TP1 - OpenFlow

F. Ferraz PG46512, J. Martins PG47916, and M. Alves PG45516

Departamento de Informática, Universidade do Minho

Abstract. The goal of this assignment is to introduce you to the OpenFlow protocol and application development on top of SDN controllers.

1 Introduction

This work can thus be divided into two exercises. The first exercise, aims to develop an application that implements a "Layer-2 self-learning switch" on OVSwitch. In the second, based on the topology presented, the goal is to develop an application that implements a "Layer-3 switch" on the OVSwitch. The packet forwarding approach could be static or dynamic.

In the development of this project, a virtual machine with Ubuntu distribution was used, where mininet (with version 2.3.0) and ryu were installed, due to the fact that it is developed in Python, and the group is more comfortable using Python than the alternative tool that was Floodlight, designed in Java. Finally, OpenFlow 1.3.0 and Wireshark were also installed, to test the behavior of packets on the network.

2 Exercise 1

The goal of this exercise is to create an application that functions as an L2 MAC self-learning switch. In this exercise, the topology to be implemented is shown as follows:

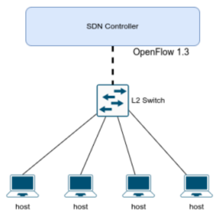
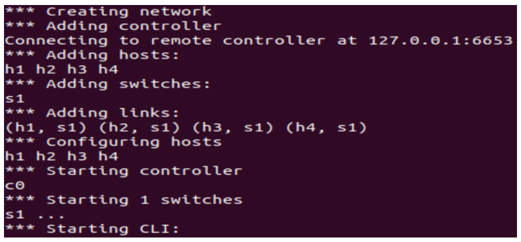


Fig. 1. Topology

This topology can be implemented using a python script, where the four nodes are created, followed by the creation of the switch, and then the links between the hosts and switch is created. This behaviour can be verified in the below image:



```
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Fig. 2. Topology Created

2.1 Strategy

Since it will be use the Ryu controller, this controller already has an example switch that with some adjustments, can accomplish the required goals of this exercise. The example switch that it will be used is the **simple_switch_13.py**. In order to make the needed changes, it was necessary to understand how does this application controller can control the OVSwitch. In order to understand its working mechanism, the first task was to divide the code into classes and functions.

The code can then be identified by having 1 class, named *SimpleSwitch13*, followed by its initiator method. In this method, it is used an attribute named **self.mac_to_port**, that will represent the MAC table of the switch. After this method, there are declared three main functions, these being:

1. switch_features_handler;
2. add_flow;
3. packet_in_handler;

This first function is during handshake between the controller and the switch and is called whenever the system is waiting to receive a SwitchFeature message. In this function an instruction was installed that sends packets that have no match in any table to the controller in order for it to be handled correctly.

The second function is the one that is used for injecting a flow into a flow table of the switch. In order to inject a flow, there are some mandatory fields that need to be completed, such as: the datapath (which can be extracted from the packet), the desired priority of the flow, the corresponding match of the packet, and finally, the action to be made to the packet.

Lastly, the third function is responsible for handling the packets that come from the switch to the controller. This function will be responsible for making the switch learn the source MAC port whenever a packet comes from the controller to the switch (PacketIN Event). This function will then compare if the destination MAC is already in the flow table, and if it is, it uses a function to get the port for that destination mac address and then sends it to that port the controller will then inject that same rule into the switch flow tables, in order to prevent the PacketIn event from being triggered again. If it is not in any table, the controller sends an order to the switch that will flood the packet to all ports.

2.2 Dry-Run

In order to run the Mininet custom topology, the first step should be to stop any controller that is still running and clean up the previous Mininet topology by using the following command:
sudo mn –clean

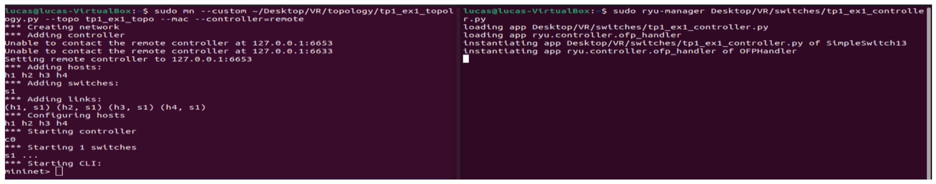
The next step is to start the mininet with the custom topology previously developed by the group. To accomplish this, it is needed to run the following command:
sudo mn –custom /Desktop/VR/topology/tp1_ex1_topology.py –topo tp1_ex1_topo –mac –controller=remote

In this command, there are some arguments that are important to mention, such as: the option –**mac**, which starts the hosts using only its MAC Address, in an ascending order; and the argument –**controller**, which will define that the controller will be initialized remotely.

Lastly, in order to start the Ryu Controller with the following command:
sudo ryu-manager Desktop/VR/switches/tp1_ex1_controller.py

2.3 Tests

In order to test the behaviour of the L2 MAC self-learning switch, it was needed to start either the mininet topology, either the controller.



```
lucas@lucas-VirtualBox:~$ sudo mn --custom ~/Desktop/VR/topology/tp1_ex1_topol
ogy.py --topo tp1_ex1_topo --mac --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

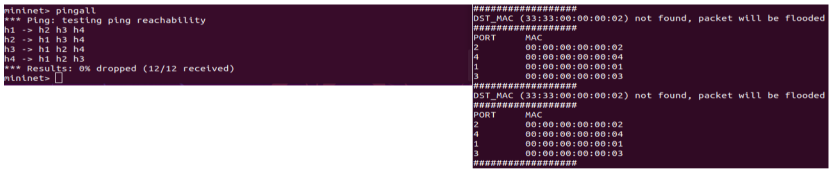
```
lucas@lucas-VirtualBox:~$ sudo ryu-manager Desktop/VR/switches/tp1_ex1_controlle
r.py
loading app Desktop/VR/switches/tp1_ex1_controller.py
loading app ryu.controller.ofp_handler
instantiating app Desktop/VR/switches/tp1_ex1_controller.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Fig. 3. Topology Startup

As soon as the test begins, the flow table is empty, so every packet that the switch receives, will be delivered to the controller, in order to for the packet to be processed. The controller first updates the MAC table with the source port and MAC address of the packet, then check if the MAC address of the destination is known, as it is the first time the packets are being exchanged, the controller sends an order to the switch to flood the packet to discover the MAC address.

When the ARP Reply reaches to the controller, the controller will update the MAC table. And after that update, there is no need for the packets with the already analysed relation to be verified by the controller, since the switch will already have the flow in their table.

The test that was performed was to test the connectivity between all the nodes, where we can see from the image below that there are no connectivity problems between them.

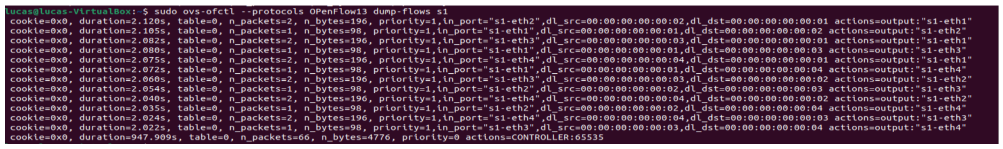


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> 
```

```
##################
DST_MAC (33:33:00:00:00:02) not found, packet will be flooded
##################
PORT      MAC
2         00:00:00:00:00:02
4         00:00:00:00:00:04
1         00:00:00:00:00:01
3         00:00:00:00:00:03
##################
DST_MAC (33:33:00:00:00:02) not found, packet will be flooded
##################
PORT      MAC
2         00:00:00:00:00:02
4         00:00:00:00:00:04
1         00:00:00:00:00:01
3         00:00:00:00:00:03
##################
```

Fig. 4. Connectivity Test

In order to check that the flows have been correctly inserted into the flows table of the switch node, we can use the command that is shown in the following image:



```
lucas@lucas-VirtualBox:~$ sudo ovs-ofctl --protocols OPenFlow13 dump-flows s1
 cookie=0x0, duration=2.120s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=2.105s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=2.082s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=2.080s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=2.075s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=2.072s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=2.060s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=2.054s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=2.040s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=2.035s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=2.024s, table=0, n_packets=2, n_bytes=196, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=2.022s, table=0, n_packets=1, n_bytes=98, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=947.909s, table=0, n_packets=66, n_bytes=4776, priority=0 actions=CONTROLLER:65535
```

Fig. 5. Flow Table on Switch

3 Exercise 2

The goal for this exercise is to develop a Layer-3 forwarder/switch. It's not exactly a router, because it won't decrements the IP TTL (Time-to-Live) and it won't recompute the checksum at each hop and this is why the traceroute won't work. However, it will match on masked IP prefix ranges, just like a real router. This ”router” can be completely static or dynamic. The following figure shows the topology that this exercise will follow. The idea is to use the self-learning application done on the previous exercise for the L2 Switches, and only develop a new application for the L3 Switch.

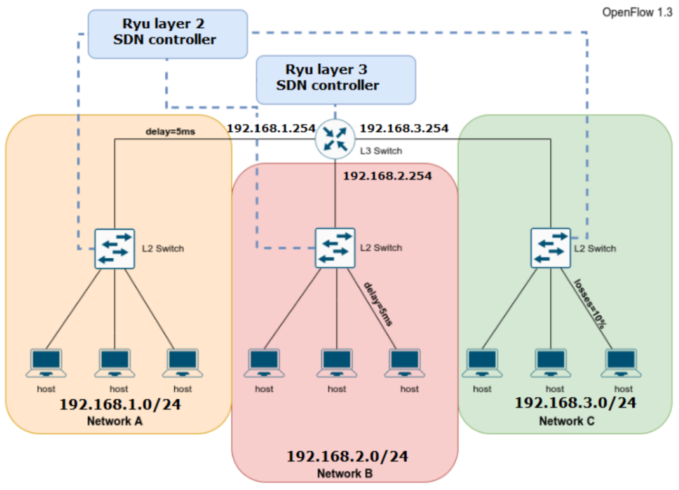


Fig. 6. Topology

As was done in the previous exercise, the topology presented above was replicated using a python script. This script will then create the hosts, and then create the switches, as well as the links between all the nodes, including the loss and delay characteristics of these links.

In this practical exercise it was also necessary to add the default gateway on each node, and since we are going to use two different communication ports, one for each controller, it was also necessary to define an IPv4 address for switch L3. The ports we will use to communicate with the L2 switches is port 6633 and the port for switch L3 is 6653, which are the default mininet ports for the controllers. In this practical exercise, one of the objectives that has to be met is for the L3 switch to be able to respond to ARP Request, that is, to be able to mount an ARP Reply and forward it to the correct port, without needing to go to the controller. Another objective is that the your app may receive ICMP echo (ping) requests for the router, which it should respond to. Lastly, packets for unreachable subnets should be answered with ICMP network unreachable messages.

For this the following structure is needed:

– Routing Table, where all the infromation is going to be statically allocated;
– Content Addressable Memory Table - associative memory table;
– Message Queue, while the router waits for the ARP Response.

3.1 Strategy

The strategy used for this exercise was as follows: the first step to be implemented was to add attributes so that it is possible to store the IP address and MAC of every L3 switch that the controller is connected to, that attribute is called **ip_to_MAC**. It was also necessary to create variables to hold the relationships between the MAC and Port of the L3 switch and the relationship between IP and Port, and for this they were named according to the relationship they hold, thus being the variables **L3_mac_to_port** and **L3_ip_to_port**.

The variable **L3_mac_to_port** is a dynamic variable, and in the course of the script there is a function called **send_port_desc_stats_request**, which is responsible for asking the ports for information about them, which includes the MAC Address. Next, there is a function called **port_desc_stats_reply_handler**, which is responsible for replying with port information from the router to the controller. In order to store the IPv4 associated to the L3 switch it is used a variable named **interface_port_to_ip**.

The last attribute that was created was called **queue**, and its objective is to store the packets and the information, that when the packet is ready to be sent, it needs to reassemble it.

Now moving on to the actual behavior of the application:

The first step that the application performs, is to check which protocol of the packet was received at the controller from the switch. And, depending on the type of packet, the behavior can be as follows:

1. If the received packet is **LLDP** (Link Layer Discovery Protocol), the application will just ignore the packet.
2. If the received packet is **ARP** (Address Resolution Protocol), there's a function named **arp_handler** that will be triggered. And this routine has the following behaviour:
   The first statement of the function is to check what type of ARP request the controller has received.
   – If the received packet is an **ARP Request** and the destination is the L3 switch, the controller will reply and send an ARP Request. If the ARP Request destination isn't the L3 switch, then the packet will be ignored.
   – If the received packet is an **ARP Reply**, the controller will verify if the queue is empty, and if there is no packets in the queue, the packet will be ignored. If the queue has any packets in it, the controller will search to check if any packets on the queue, match the reply that was received. If there is no match, the packet will be ignored.

In the case, that there are packets on the queue and there is a match with a packet inside it, then it means that the ARP Reply that was received by the controller, was request by himself. And, for that reason, the MAC and IP table will be updated, the flow will be added to the flow table, and the packet is forwarded to the destination. Once the packet is forwarded, the packet is deleted from the queue.

After this function, the **flood_arp** is called, this function is responsible for handling packets in which the destination in unknown. In this case, the controller will check if the subnet is known, and if this condition is true, the packet will be flooded on that same port of that subnet.

3. If the received packet is **IP** (Internet Protocol), the following behaviour will be performed:
   - The tables MAC and IP are updated with the information extracted from the packet, and if the destination IP is known, the flow will be added to the flow table and the packet will be forwarded.
   - If the Destination IP is not in the IP table, the controller will verify if the destination in the L3 switch itself. If it's true, the controller will verify if the packet received is a ICMP Request, and if it is, then the controller will reply with an ICMP Reply to the host who sent the ICMP Request. If it isn't an ICMP Request, the packet will be ignored.
   - In the case, that the Destination IP is not on any IP Table and it isn't the IP of the L3 Switch, the controller will call the function above explained, the **flood_arp**. Which in this case, this function will place the packet, on the queue and search for the right port to send the ARP request, in order to search for the destination on the right subnet.

3.2 Dry-Run

In order to run the Mininet custom topology, the first step should be to stop any controller that is still running and clean up the previous Mininet topology by using the following command:

```
sudo mn –c
```

The next step is to start the mininet with the custom topology previously developed by the group. To accomplish this, it is needed to run the following command:

```
sudo python3 Desktop/VR/topology/tp1_ex2_topology.py
```

Lastly, it is needed to start both the Ryu Controller for the L2 switch and the L3 switch. Just like it was said before, the ports that were used was the 6633 for the Layer 2 switch and the 6653 for the Layer 3 switch. To start the layer 2 switch controller, the following command is used:

```
sudo ryu-manager Desktop/VR/switches/tp1_ex1_controller.py –ofp-tcp-listen.port 6633
```

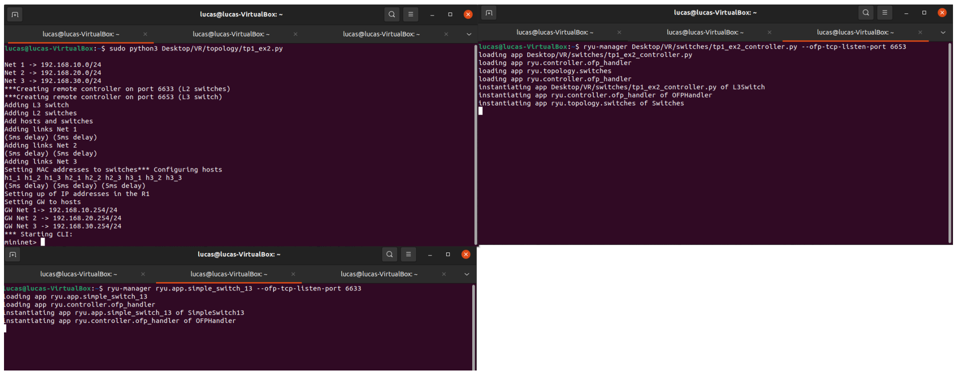
To start the layer 3 switch controller, the following command is used:

```
sudo ryu-manager Desktop/VR/switches/tp1_ex2_controller.py –ofp-tcp-listen.port 6653
```

3.3 Tests

In order to test the functionality of this exercise, both the topology and the controllers must be running.

Once the topology and both controllers start, both controllers are waiting for packets to be transmitted, so they can start filling in the flow tables.



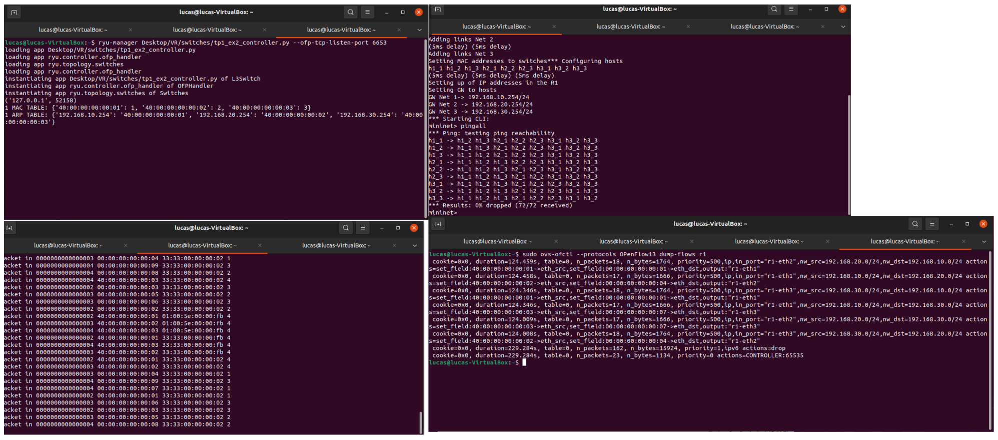
```
lucas@lucas-VirtualBox:~$ sudo python3 Desktop/VR/topology/tp1_ex2.py

Net 1 -> 192.168.10.0/24
Net 2 -> 192.168.20.0/24
Net 3 -> 192.168.30.0/24
***Creating remote controller on port 6633 (L2 switches)
***Creating remote controller on port 6653 (L3 switch)
Adding L3 switch
Adding L2 switches
Add hosts and switches
Adding links Net 1
(5ms delay) (5ms delay)
Adding links Net 2
(5ms delay) (5ms delay)
Adding links Net 3
Setting MAC addresses to switches*** Configuring hosts
h1_1 h1_2 h1_3 h2_1 h2_2 h2_3 h3_1 h3_2 h3_3
(5ms delay) (5ms delay) (5ms delay)
Setting up of IP addresses in the R1
Setting GW to hosts
GW Net 1-> 192.168.10.254/24
GW Net 2 -> 192.168.20.254/24
GW Net 3 -> 192.168.30.254/24
*** Starting CLI:
mininet>
```

```
lucas@lucas-VirtualBox:~$ ryu-manager Desktop/VR/switches/tp1_ex2_controller.py --ofp-tcp-listen-port 6653
loading app Desktop/VR/switches/tp1_ex2_controller.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app Desktop/VR/switches/tp1_ex2_controller.py of L3Switch
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
```

```
lucas@lucas-VirtualBox:~$ ryu-manager ryu.app.simple_switch_13 --ofp-tcp-listen-port 6633
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Fig. 7. Topology and Controllers Starting Point

The test that was done on this topology was a connectivity test between all the hosts present in the topology. The results of the below image, show us that all the nodes can communicate with the hosts inside the same network and with the hosts that belong to other networks.



```
lucas@lucas-VirtualBox:~$ ryu-manager Desktop/VR/switches/tp1_ex2_controller.py --ofp-tcp-listen-port 6653
loading app Desktop/VR/switches/tp1_ex2_controller.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app Desktop/VR/switches/tp1_ex2_controller.py of L3Switch
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
('127.0.0.1', 52158)
1 MAC TABLE: {'40:00:00:00:00:01': 1, '40:00:00:00:00:02': 2, '40:00:00:00:00:03': 3}
1 ARP TABLE: {'192.168.10.254': '40:00:00:00:00:01', '192.168.20.254': '40:00:00:00:00:02', '192.168.30.254': '40:00
:00:00:00:03'}
```

```
Adding links Net 2
(5ms delay) (5ms delay)
Adding links Net 3
Setting MAC addresses to switches*** Configuring hosts
h1_1 h1_2 h1_3 h2_1 h2_2 h2_3 h3_1 h3_2 h3_3
(5ms delay) (5ms delay) (5ms delay)
Setting up of IP addresses in the R1
Setting GW to hosts
GW Net 1-> 192.168.10.254/24
GW Net 2 -> 192.168.20.254/24
GW Net 3 -> 192.168.30.254/24
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1_1 -> h1_2 h1_3 h2_1 h2_2 h2_3 h3_1 h3_2 h3_3
h1_2 -> h1_1 h1_3 h2_1 h2_2 h2_3 h3_1 h3_2 h3_3
h1_3 -> h1_1 h1_2 h2_1 h2_2 h2_3 h3_1 h3_2 h3_3
h2_1 -> h1_1 h1_2 h1_3 h2_2 h2_3 h3_1 h3_2 h3_3
h2_2 -> h1_1 h1_2 h1_3 h2_1 h2_3 h3_1 h3_2 h3_3
h2_3 -> h1_1 h1_2 h1_3 h2_1 h2_2 h3_1 h3_2 h3_3
h3_1 -> h1_1 h1_2 h1_3 h2_1 h2_2 h2_3 h3_2 h3_3
h3_2 -> h1_1 h1_2 h1_3 h2_1 h2_2 h2_3 h3_1 h3_3
h3_3 -> h1_1 h1_2 h1_3 h2_1 h2_2 h2_3 h3_1 h3_2
*** Results: 0% dropped (72/72 received)
mininet>
```

```
acket in 0000000000000003 00:00:00:00:00:04 33:33:00:00:00:02 1
acket in 0000000000000004 00:00:00:00:00:09 33:33:00:00:00:02 3
acket in 0000000000000004 00:00:00:00:00:08 33:33:00:00:00:02 2
acket in 0000000000000004 40:00:00:00:00:03 33:33:00:00:00:02 4
acket in 0000000000000002 00:00:00:00:00:03 33:33:00:00:00:02 3
acket in 0000000000000003 00:00:00:00:00:05 33:33:00:00:00:02 2
acket in 0000000000000003 00:00:00:00:00:06 33:33:00:00:00:02 3
acket in 0000000000000002 00:00:00:00:00:02 33:33:00:00:00:02 2
acket in 0000000000000002 40:00:00:00:00:01 01:00:5e:00:00:fb 4
acket in 0000000000000003 40:00:00:00:00:02 01:00:5e:00:00:fb 4
acket in 0000000000000004 40:00:00:00:00:03 01:00:5e:00:00:fb 4
acket in 0000000000000002 40:00:00:00:00:01 33:33:00:00:00:fb 4
acket in 0000000000000004 40:00:00:00:00:03 33:33:00:00:00:fb 4
acket in 0000000000000003 40:00:00:00:00:02 33:33:00:00:00:fb 4
acket in 0000000000000002 40:00:00:00:00:01 33:33:00:00:00:02 4
acket in 0000000000000003 40:00:00:00:00:02 33:33:00:00:00:02 4
acket in 0000000000000003 00:00:00:00:00:04 33:33:00:00:00:02 1
acket in 0000000000000004 00:00:00:00:00:09 33:33:00:00:00:02 3
acket in 0000000000000004 00:00:00:00:00:07 33:33:00:00:00:02 1
acket in 0000000000000002 00:00:00:00:00:01 33:33:00:00:00:02 1
acket in 0000000000000003 00:00:00:00:00:06 33:33:00:00:00:02 3
acket in 0000000000000002 00:00:00:00:00:03 33:33:00:00:00:02 3
acket in 0000000000000003 00:00:00:00:00:05 33:33:00:00:00:02 2
acket in 0000000000000004 00:00:00:00:00:08 33:33:00:00:00:02 2
```

```
lucas@lucas-VirtualBox:~$ sudo ovs-ofctl --protocols OPenFlow13 dump-flows r1
 cookie=0x0, duration=124.459s, table=0, n_packets=18, n_bytes=1764, priority=500,ip,in_port="r1-eth2",nw_src=192.168.20.0/24,nw_dst=192.168.10.0/24 action
s=set_field:40:00:00:00:00:01->eth_src,set_field:00:00:00:00:00:01->eth_dst,output:"r1-eth1"
 cookie=0x0, duration=124.458s, table=0, n_packets=17, n_bytes=1666, priority=500,ip,in_port="r1-eth1",nw_src=192.168.10.0/24,nw_dst=192.168.20.0/24 action
s=set_field:40:00:00:00:00:02->eth_src,set_field:00:00:00:00:00:04->eth_dst,output:"r1-eth2"
 cookie=0x0, duration=124.346s, table=0, n_packets=18, n_bytes=1764, priority=500,ip,in_port="r1-eth3",nw_src=192.168.30.0/24,nw_dst=192.168.10.0/24 action
s=set_field:40:00:00:00:00:01->eth_src,set_field:00:00:00:00:00:01->eth_dst,output:"r1-eth1"
 cookie=0x0, duration=124.346s, table=0, n_packets=17, n_bytes=1666, priority=500,ip,in_port="r1-eth1",nw_src=192.168.10.0/24,nw_dst=192.168.30.0/24 action
s=set_field:40:00:00:00:00:03->eth_src,set_field:00:00:00:00:00:07->eth_dst,output:"r1-eth3"
 cookie=0x0, duration=124.009s, table=0, n_packets=17, n_bytes=1666, priority=500,ip,in_port="r1-eth2",nw_src=192.168.20.0/24,nw_dst=192.168.30.0/24 action
s=set_field:40:00:00:00:00:03->eth_src,set_field:00:00:00:00:00:07->eth_dst,output:"r1-eth3"
 cookie=0x0, duration=124.008s, table=0, n_packets=18, n_bytes=1764, priority=500,ip,in_port="r1-eth3",nw_src=192.168.30.0/24,nw_dst=192.168.20.0/24 action
s=set_field:40:00:00:00:00:02->eth_src,set_field:00:00:00:00:00:04->eth_dst,output:"r1-eth2"
 cookie=0x0, duration=229.284s, table=0, n_packets=162, n_bytes=15924, priority=1,ipv6 actions=drop
 cookie=0x0, duration=229.284s, table=0, n_packets=23, n_bytes=1134, priority=0 actions=CONTROLLER:65535
lucas@lucas-VirtualBox:~$
```

Fig. 8. Results of Pings and Flows

As it can be seen from the image above, the layer 2 switch received the packets, where it is displayed the information of the MAC Address of the Source and the MAC Address of the Destination. On the router 1, the flows table can shows us its contents, where it can be seen the actions performed to the packets. Finally, on the L3 Switch, in the MAC Table, it has the information about the MAC Address and the port, respectively. In the ARP Table, the MAC Address is associated with the IP Address of that same port.

4 Conclusion

To conclude this work, the objective was to develop a layer2 switch for the first exercise where the group managed to do it by analyzing a already existing simple switch example from ryu and edit it to the exercise needs, the second exercise proved more of a challenge for the group due to our overall lack of programming knowledge but in the end the group managed to complete it. Overall this work gave the group more knowledge about how SDN's work and how to create example of switches to be implemented on a virtualized network.