Lab2 report

109550005 資工 13 張可晴

Execution

Part1 : Run Mininet and Ryu controller

- Steps for running mininet and Ryu controller to ping successfully from host to host.

I followed the instruction of ppt to finish these tasks. First of all, I set up the environment. Secondly, I run the original topo.py and SimpleController.py on two terminal to test whether it is correct. After that, I modified topo.py by adding those bandwidth, delay and loss rate in addLink function to build the topology through mininet to meet the need of topo.png. Then, I duplicated SimpleController.py twice , naming it as controller1.py and controller2.py, respectively, modifying some port number ,adding some new switches in switch_features_handler function to satisfy the forwarding rules in page 25 and page 27. As for measurement, I measured the three codes by running them in different terminal to confirm that ICMP and APR packets can successfully reach the destination. Next, I created a folder “out” in src to record the statics. Entering iPerf command in mininet CLI, enabling me to measure my bandwidth and store the result to out/result1, 2, and 3. Thus, we can observe the number of packets that switch 2 received in the terminal running the controller. Finally, I used the command presented on ppt to output the forwarding rules on s2 (switch 2). Last but not least, doing these measurement steps three times to measure the different forwarding rules.

- What is the meaning of the executing command (both Mininet and Ryu controller)?

sudo mn --custom topo.py --topo topo --link tc --controller remote

mn --custom topo.py : choose topo.py file by custom parameter.

--topo topo : according to the name of the last line of python.

--link tc : users can setup via connection.

--controller remote : setup controller, remote = controlled by outside controller.

sudo ryu-manager SimpleController.py --observe-links

ryu-manager: loads Ryu application and run it.

--observe-links: manifest the message between connection.

mn-c: clean up the mininet or RTNETLINK

h1 ping h2: test the connection between h1 and h2

h1 iperf -s -u -i 1 > ./out/result1 &:

-s: run h1 as server.

-u: use UDP.

-i 1:it will report in every one second.

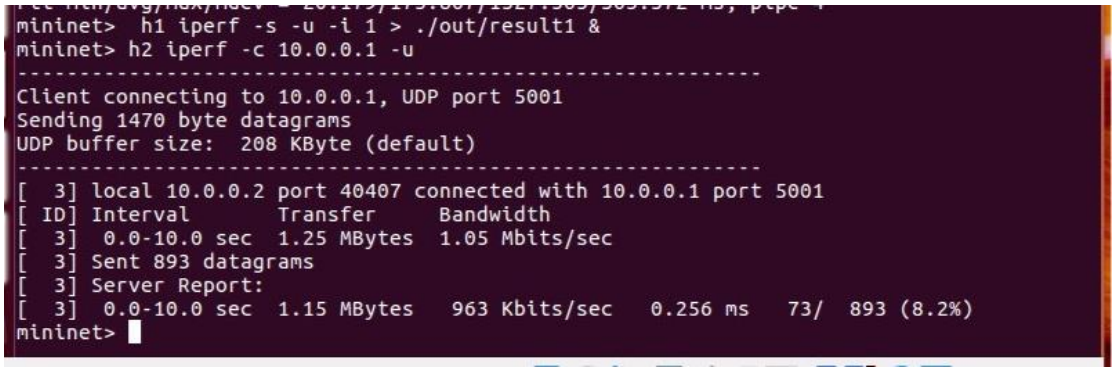
> ./out/result1 &:output and save data in result1.

h2 iperf -c 10.0.0.1 -u:

-c: run h2 as client, connecting to 10.0.0.1.

•Screenshots

SimpleController



Controller1

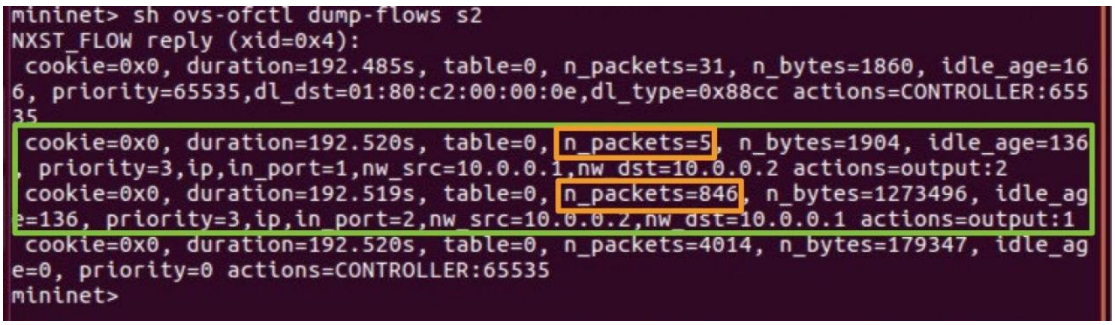
```
mininet> h1 iperf -s -u -i 1 > ./out/result2 &
mininet>  h2 iperf -c 10.0.0.1 -u
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 59835 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.5 sec  1.13 MBytes   904 Kbits/sec  31.932 ms   85/  893 (9.5%)
mininet>
```

```
cn2021@cn2021-VirtualBox:~$ cd lab2-cassie0206/src
cn2021@cn2021-VirtualBox:~/lab2-cassie0206/src$ sudo  ryu-manager controller1.py
 --observe-links
[sudo] password for cn2021:
loading app controller1.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app controller1.py of SimpleController
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
switch 2: count 0 packets
switch 2: count 1 packets
switch 2: count 5 packets
switch 2: count 5 packets
switch 2: count 5 packets
switch 2: count 534 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
switch 2: count 814 packets
```

```
mininet> sh ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=232.355s, table=0, n_packets=62, n_bytes=3720, idle_age=165, pri
ority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=232.412s, table=0, n_packets=7, n_bytes=2100, idle_age=166, prio
rity=3,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
 cookie=0x0, duration=232.412s, table=0, n_packets=814, n_bytes=1223698, idle_age=166,
 priority=3,ip,in_port=3,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=232.415s, table=0, n_packets=5746, n_bytes=311080, idle_age=0, p
riority=0 actions=CONTROLLER:65535
mininet>
```

Controller2

```
mininet> h1 iperf -s -u -i 1 > ./out/result3 &
mininet> h2 iperf -c 10.0.0.1 -u
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 34382 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.18 MBytes   989 Kbits/sec   0.765 ms   52/  893 (5.8%)
mininet>
```



Part2 : Handling flow-removed events

In terms of task6, I will firstly duplicate SimpleController.py ,naming it AdaptiveController.py. Second of all, modifying the switch_features_handler function making the forwarding rule satisfying three orange paths(h2->h1) , separating three different paths by making the priority in add_flow function of SimpleController, Controller1, and Controller2 to be 1, 2, and 3, respectively. Moreover, I let the hard_timeout of Controller1 to be 20, Controller2 to be 40, SimpleController to be 60. In this way, I can measure the bandwidth of those three pathes. After three bandwidth finish measurement, I will do comparison and print the greater one's path number and its python file's name in packet_in_handler. Moreover, I added self.add_flow and set its priority to be the corresponding one, its hard_timeout to be 0, which allows the best path to run continually.

Part3 : Problems encountered

• Problems you met while doing this lab

1. There are many trivial problems such as forgetting something learned from lab0 and lab1, not familiar with the operation of using Mininet and Ryu

simultaneously, confused about which step need to operate on which terminal.

2. The first time I saw the continuous occurrence of “switch 2: count 0 packets” regarding them as error, so I will clean up the terminal. Not until I’m doing task5, I know the reason why this situation will happen.
3. When I do step x-3 of task5, I found the number of packets will greater than the number of the difference of packet be sent and packet loss. I think is because the command “h1 ping h2” accumulating the number of packets.
4. When I do task1 through task5, most of time, I will encounter error is because I don’t exit or clean up last step.
5. For task6, I will forget to let the bandwidth to be global variable, preventing me from doing comparison.
6. Because I’m not familiar with python and those function, making the best path run continuously by adding self.add_flow function and some parameters, which cost me lots of time to debug.

Discussion

1. Describe the differences between packet-in and packet-out in detail
   Packet-in and packet-out come from the command of the controller. If OpenFlow switch receives packet-in, then it will send the packet to the controller. In contrast, if receiving packet-out, it will send the packet from the controller to a specified port.
2. What is “table-miss” in SDN?
   A packet that it can’t find a matched flow entry in a flow table.
3. Why is “(app_manager.RyuApp)” adding after the declaration of class in SimpleController.py?
   When writing a Ryu application, you need to let your application inherit RyuApp , which is an important base class.
4. What is the meaning of “datapath” in SimpleController.py?
   The operation of OpenFlow switches in topology and flow tables come from the object of datapath.
5. Why need to set “eth_type=0x0800” in the flow entry?
   Set the IPv4 protocol.
6. Compare the differences between the iPerf results of SimpleController.py, controller1.py and controller2.py. Which forwarding rule is better? Why?
   Controller2.py.
   Because the loss rate of Controller2 is smaller.
   Controller1(9.5%) > SimpleController(8.2%) > Controller2(5.8%)