Report of lab2 109550027

- Execution

  Part 1 : Run Mininet and Ryu controller

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

  - Step 0 : Modify topo.py

    Add (bw, delay, loss) in self.addlink to meet the constraints in topo.png.

  - Step 1 : run Mininet topology

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

  - Step 2 : run Ryu manager with controller in another terminal

    `$ sudo ryu-manager AdaptiveController.py --observe-links`

  - Step 3: run ping command in Mininet to check the connection

    `mininet> h1 ping h2`

  - Step 4 : run iPerf command for measurement

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

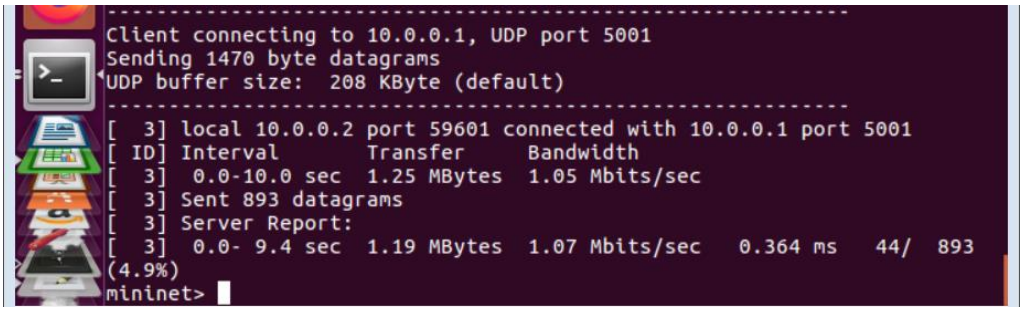
    `mininet> h2 iperf -c 10.0.0.1 -u`

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

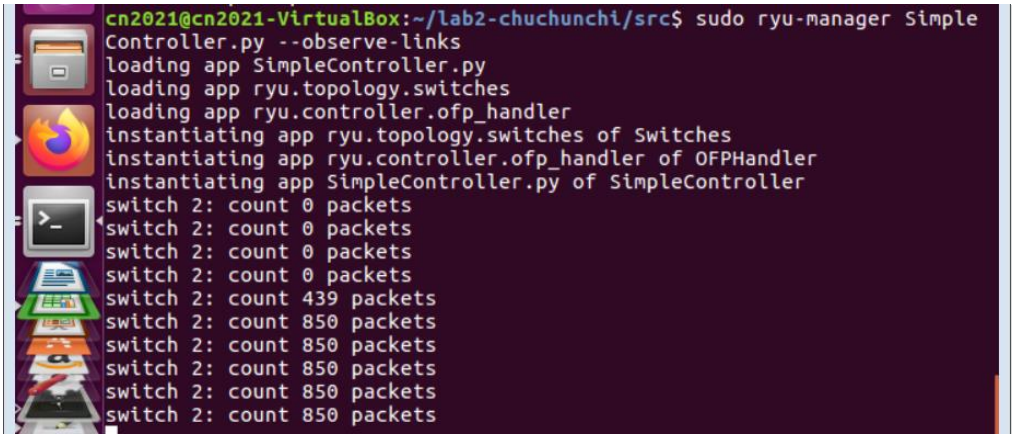
  - Mininet:
    - `mn --custom topo.py`: use self-define topology topo.py
    - `mn --topo topo`: the topology name in .py file is “topo”.
    - `mn --link tc`: speed constraint
    - `mn --controller remote`: use external controller to control the network
    - `h1 iperf -s`: server mode, server=h1.
    - `-u`: use UDP
    - `-i 1`: set the interval time to one second.
    - `-p 5566`: the server port to listen on = port 5566.
    - `./out/result1 &`: the location of output.
    - `h2 iperf -c 10.0.0.1`: client mode, client=h2, connected to a server(IP=10.0.0.1).
  - Ryu controller:
    - `ryu-manager SimpleController.py`: run SimpleController.py and use it as the controller.
    - `--observe-links`: display the message between links.

- Screenshots

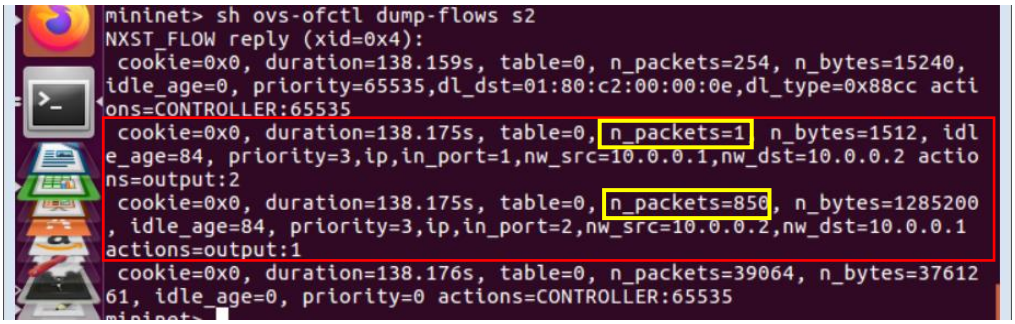
The output of iperf for SimpleController:



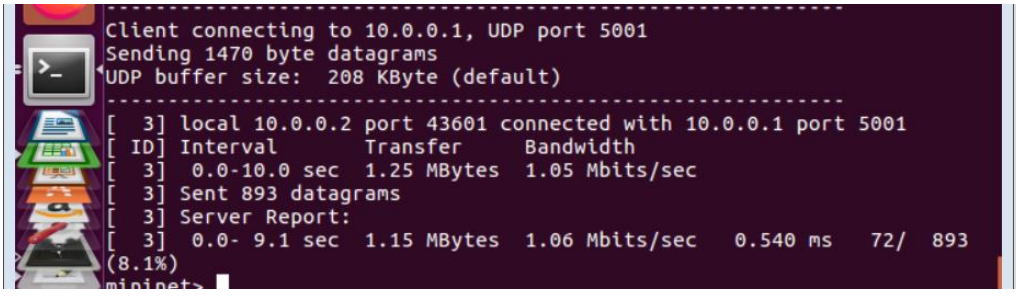
The number of packets for SimpleController:



The forwarding rules on switch 2 for SimpleController:

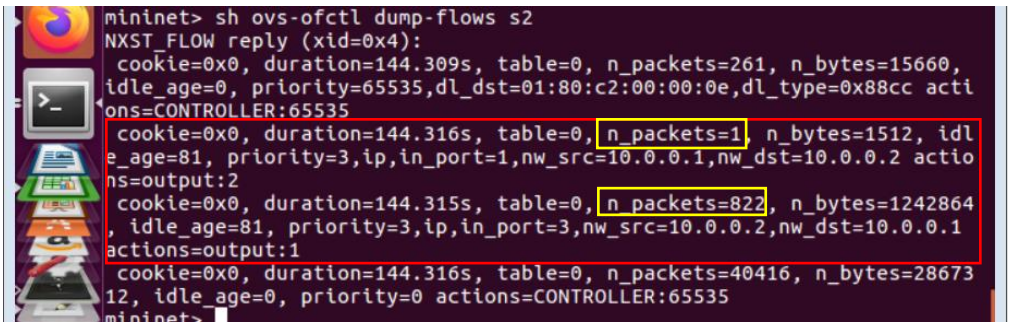


The output of iperf for controller1:

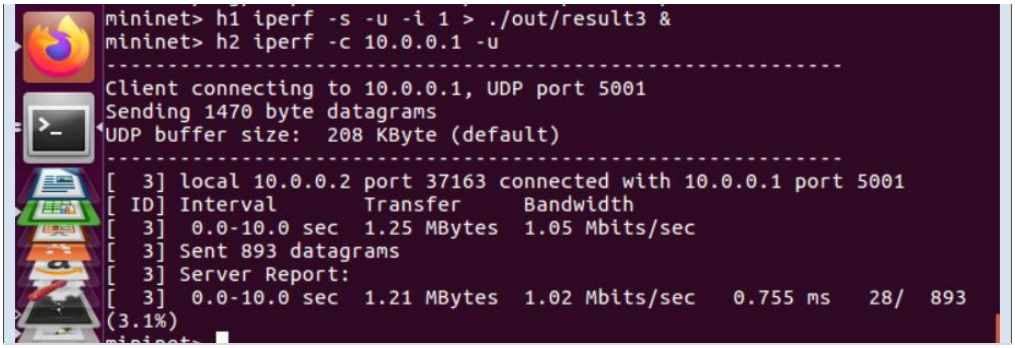


The number of packets for controller1:

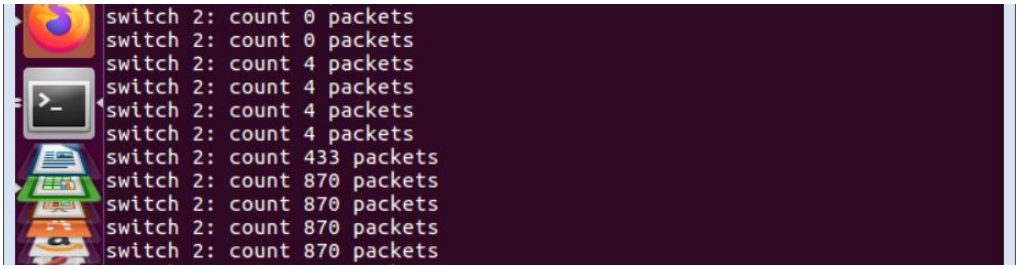
The forwarding rules on switch 2 for controller1:



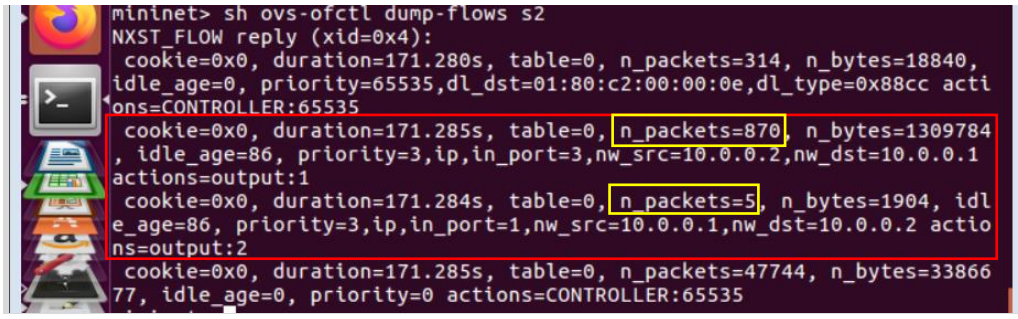
The output of iperf for controller2:



The number of packets for controller2:



The forwarding rules on switch 2 for controller2:



Part 2 : Handling flow-removed events

我先仔細地了解了 ryu function 裡所有函式，簡單筆記如下：

`_state_change_handler` 用來判斷狀態是 `MAIN_DISPATCHER`(Switch 連接中)還是 `DEAD_DISPATCHER`(Switch 中斷連接)。

`_timer` 每間隔 10 秒，向正在監測的 switch 請求取得數據。

`_request_stats` 中的 `OFPFlowStatsRequest` 取得 switch 規則資訊。

_flow_stats_reply_handler 處理剛剛接收到的規則資訊，其中包含 packet_count 這個參數。

switch_features_handler 中紀錄了很多 match 條件，以及要產生的 flow 的規則，最後利用 add_flow 產生那個 flow。

packet_in_handler pkt in 會發生是因為在 table 中沒有相對應的規則，而我們在這裡解析封包、建立規則並且觸發 action，把 pkt 送到該去的地方。

flow_removed_handler 在 msg 中找出 flow remove 的原因，並記錄(印出)flow 的數據。

研究完這些函式，再加上助教的提示，我大概想要先把 3 個 path 的 forwarding rule 加到 switch_features_handler 中，並控制 priority (flow entry 的優先權，數字大的優先跑)以及 hard_timeout (flow entry 的有效期限)。在這裡，我把 priority 分別設成(23,13,3)，把 hard timeout 設成(30,60,90)。

每次 hard_timeout 結束以後，controller 會收到 flow remove 的訊息，因此，我在 flow_remove_handler 這裡設定一個參數紀錄現在的 packet_count 以及現在的 path，實作如下：

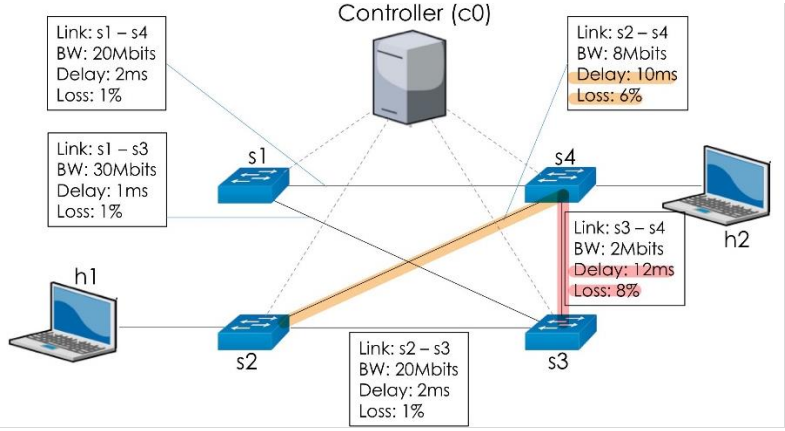
在 flow_remove_handler 中，利用當前的 priority 大小就可以知道現在是哪條 path，另外，我設一個 list pktcount 來記錄三個 path 的 packet count 數目-> pktcount[path-1] = msg.packet_count

最後，我印出最多 packet 的 path：pktcount.index(max(pktcount))+1，接著再根據 path，重新 add flow(利用 if 判斷要新增哪個 path)。在這裡的 add flow 和前面不同的是我把 hard timeout 都設成 30，而不設成 0 是因為我怕中途會有偵測不到的 flow remove event，所以固定 30 秒重新讓他跑一次。

Part 3 : Problems encountered

超級無敵多…，真的謝謝助教們一直解答我的問題。一開始的問題是沒有方向可言，後來就先看 RYU BOOK，還有一個有用的 GITHUB，認真把每個 function 都搞懂(再加上每天滑討論區看別人的問題)以後好像就比較知道要怎麼做，然後後來還有遇到問題像是 packet 全部都是 0，我同學跟我說是我的 timeout 設太短，還來不及傳，把 timeout 拉長以後就改善。還有一些 python 語法問題就先省略，總之，看到他跑出一條 path 出來我真的超級無敵感動…。雖然我也不確定答案是不是完全符合題目要求，但希望助教在跑的時候等他久一點，可能我 timeout 又設太長，但跑久一點真的會有東西跑出來啦…！

- Discussion
  1. **Describe the differences between packet-in and packet-out in detail**
     Packet-in: The process that a switch receives a packet, and then transmit it to the controller.
     Packet-out: Initialize by the controller, the process that switch receive packets from the controller and transmit it to the port.
  2. **What is "table-miss" in SDN?**
     Table-miss means you can't find the corresponding flow entry in the flow table. Solution: Ignore the packet, send to other flow table or send to the controller.
  3. **Why is "(app_manager.RyuApp)" adding after the declaration of class in SimpleController.py?**
     SimpleController.py has to inherit app_manager.RyuApp such that it can call the function in it.
  4. **What is the meaning of "datapath" in SimpleController.py?**
     The datapath class here is used to deal with the important massage from the switch. For example, the communication with the OpenFlow switch. So, at the end of many function, we use `datapath.send_msg()` to update datapath.
  5. **Why need to set "eth_type=0x0800" in the flow entry?**
     eth_type=0x0800: IPv4 protocal, show the IP address of endpoint.
     Why? Can exchange message between two endpoint.
  6. **Compare the differences between the iPerf results of SimpleController.py, controller1.py and controller2.py. Which forwarding rule is better? Why?**
     Controller1:
     avg bandwidth = 1.06 Mbits/sec, delay = 0.54 ms, packet loss rate = 8.1%
     Controller2:
     avg bandwidth = 1.02 Mbits/sec, delay = 0.755 ms, packet loss rate = 3.1%
     SimpleController:
     avg bandwidth = 1.07 Mbits/sec, delay = 0.364 ms, packet loss rate = 4.9%



From the iperf result above, we can observe that controller1 has the worst performance , while controller2 and SimpleController's performance are roughly the same.

Also, by the topo we define, we can see that link s3-s4 was the worst link (bw is low and huge delay&packet loss), and link s2-s4 is the second worth one. So, controller1, which runs through link s3-s4, has the worth performance. On the other hand, controller2, which neither run in s3-s4 and s2-s4, is expected to have the best performance.