Report 109550031 李旻融

Execution

-Part 1: Run Mininet and Ryu controller

一開始先在 topo.py 中加入一些參數，像是 bandwidth, delay, loss，這樣即可完成 mininet 的 topology。再來要複製兩份 SimpleController.py，分別更改其 switch_features_handler 這個函式的內容來產生 controller1.py 和 controller2.py，要更改的地方大致如下: msg.datapath.id 是 switch 的代號，in_port 則是傳入的 switch 的 port number，actions = [parser.OFPActionOutput()] 則是該 switch 要傳出去 port number。最後就可以來測量三個 iperf 的 output、s2 上符合 forwarding rules 的 packet 數量和其 flow，分成以下三個檔案來說明。

*** SimpleController.py**

1. 先在一個 terminal 上跑 topo.py。

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

2. 開啟另一個 terminal 跑 SimpleController.py。

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

3. 在 Mininet CLI 中執行 ping 的指令，以確保 ICMP 和 APR 有成功送達。

   ```
   $ h1 ping h2
   ```

4. 在 Mininet CLI 中執行 iperf 指令，測量 bandwidth。

   ```
   $ h1 iperf -s -u -i 1 > ./out/result1 &
   $ h2 iperf -c 10.0.0.1 -u -i 1
   ```

```
Files
cting 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 36774 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  6.0- 7.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  9.0-10.0 sec   128 KBytes  1.05 Mbits/sec
[  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  1.19 MBytes  1000 Kbits/sec   0.333 ms   45/  893 (5%)
```

5. 確認 packet 的數量。

```
switch 2: count 0 packets
switch 2: count 58 packets
switch 2: count 849 packets
switch 2: count 849 packets
switch 2: count 849 packets
```

6. 輸出 s2 的 forwarding rules。

$ sh ovs-ofctl dump-flows s2

```
 cookie=0x0, duration=61.734s, table=0, n_packets=87, n_bytes=5220, idle_age=0,
priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x0, duration=61.736s, table=0, n_packets=1, n_bytes=1512, idle_age=36,
priority=3,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
 cookie=0x0, duration=61.735s, table=0, n_packets=849, n_bytes=1283688, idle_age
=36, priority=3,ip,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=61.736s, table=0, n_packets=3733, n_bytes=262661, idle_age
=0, priority=0 actions=CONTROLLER:65535
```

*** controller1.py**

1. 先在一個 terminal 上跑 topo.py。

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

2. 開啟另一個 terminal 跑 controller1.py。

$ sudo ryu-manager controller1.py --observe-links

3. 在 Mininet CLI 中執行 ping 的指令，以確保 ICMP 和 APR 有成功送達。

$ h1 ping h2

4. 在 Mininet CLI 中執行 iperf 指令，測量 bandwidth。

$ h1 iperf -s -u -i 1 > ./out/result2 &

$ h2 iperf -c 10.0.0.1 -u -i 1

```
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 50206 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  6.0- 7.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  9.0-10.0 sec   128 KBytes  1.05 Mbits/sec
[  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  1.16 MBytes   976 Kbits/sec   0.152 ms   66/  893 (7.4%)
```

5. 確認 packet 的數量。

```
switch 2: count 0 packets
switch 2: count 142 packets
switch 2: count 828 packets
switch 2: count 828 packets
switch 2: count 828 packets
```

6. 輸出 s2 的 forwarding rules。

```
$ sh ovs-ofctl dump-flows s2
```

```
 cookie=0x0, duration=119.143s, table=0, n_packets=152, n_bytes=9120, idle_age=1
, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:6553
5
 cookie=0x0, duration=119.146s, table=0, n_packets=1, n_bytes=1512, idle_age=93,
 priority=3,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
 cookie=0x0, duration=119.146s, table=0, n_packets=828, n_bytes=1251936, idle_ag
e=93, priority=3,ip,in_port=3,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=119.147s, table=0, n_packets=22022, n_bytes=1012396, idle_
age=0, priority=0 actions=CONTROLLER:65535
```

*** controller2.py**

1. 先在一個 terminal 上跑 topo.py。

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

2. 開啟另一個 terminal 跑 controller2.py。

```
$ sudo ryu-manager controller2.py --observe-links
```

3. 在 Mininet CLI 中執行 ping 的指令，以確保 ICMP 和 APR 有成功送達。

```
$ h1 ping h2
```

4. 在 Mininet CLI 中執行 iperf 指令，測量 bandwidth。

```
$ h1 iperf -s -u -i 1 > ./out/result3 &
$ h2 iperf -c 10.0.0.1 -u -i 1
```

```
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 55218 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  6.0- 7.0 sec   129 KBytes  1.06 Mbits/sec
[  3]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec
[  3]  9.0-10.0 sec   128 KBytes  1.05 Mbits/sec
[  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  1.21 MBytes  1.01 Mbits/sec   0.221 ms   33/  893 (3.7%)
```

5. 確認 packet 的數量。

```
switch 2: count 0 packets
switch 2: count 4 packets
switch 2: count 4 packets
switch 2: count 397 packets
switch 2: count 865 packets
switch 2: count 865 packets
```

6. 輸出 s2 的 forwarding rules。

$ sh ovs-ofctl dump-flows s2

```
 cookie=0x0, duration=1915.599s, table=0, n_packets=2733, n_bytes=163980, idle_a
ge=0, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:
65535
 cookie=0x0, duration=1915.601s, table=0, n_packets=5, n_bytes=1904, idle_age=18
72, priority=3,ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
 cookie=0x0, duration=1915.601s, table=0, n_packets=865, n_bytes=1302224, idle_a
ge=1872, priority=3,ip,in_port=3,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:
1
 cookie=0x0, duration=1915.601s, table=0, n_packets=251286, n_bytes=13095922, id
le_age=0, priority=0 actions=CONTROLLER:65535
```

*** meaning of the executing command**

Mininet

-sudo

--custom : 使用自己定義的 topology(python 檔)

--topo : python 檔中 topology 的檔名

--link tc : TCLink，使用--link 可以設置網路參數

--controller remote: 使用外部的 controller 控制網路

-iperf

-s : server

-u : UDP

-i : interval (以秒為單位顯示的間隔)

-p : port (指定伺服器的 port number)

-c : client

h1 iperf -s -u -i 1 > ./out/result1 & : h1 是 server(-s)，使用 UDP 協議(-u)，每隔一秒監視結果(-i)，並在後台運行(&)，將結果輸出至./out/result1。

$ h2 iperf -c 10.0.0.1 -u -i 1 : h2 是 client(-c)，指定 server 的地址 10.0.0.1，一樣使用 UDP 協議(-u)，每隔一秒監視結果(-i)。

Ryu controller

ryu-manager+ .py : 執行該程式作為 controller

--observe-links : 顯示 link 間的訊息

- Part2: Handling flow-removed events

1. 一開始先把這三個檔案(SimpleController.py、controller1.py、controller2.py)在 switch_features_handler 這個函式裡所有的 forwarding rule 都加進來，再分別給他們 priority 和 hard_timeout。path 1 (controller1.py) 的 priority 是 1，path 2 (controller2.py) 的 priority 是 2，path 3 (SimpleController.py) 的 priority 是 3，因為 priority 值越大的會越先跑，所以 path 3 跑完才輪到 path 2，最後才是 path 1，也因為 hard_timeout 是累加上去的，所以在設定的時候，分別給 path 3 的 hard_timeout 是 20、path 2 的 hard_timeout 是 40、path 1 的 hard_timeout 是 60，這意味著他們在多久後會被 remove 掉。

2. 當時間一到，該條 path 就會被 remove，進而跑進 flow_removed_handler 這個函式。首先，我先宣告三個全域變數(bw1,bw2,bw3)，在此函式裡，我先判斷進來的 priority 是 1 還 2 還 3，進而加進 bw1、bw2、bw3，這個部分我是利用收到 byte 的數量(byte_count)來計算，因為三個的時間皆是二十，所以可以不用理會，在同樣的時間裡，收到的 byte 數量越多，基本上意味著頻寬越大。在前二十秒內，因為 priority = 3 的會優先，所以前二十秒算的皆會是屬於 bw3 的 byte 數，等二十秒一到，他們會全數被 remove，接下來的二十秒則輪到 priority = 2，接下來算的是屬於 bw2 的 byte 數，四十秒一到，也會被 remove，最後二十秒則是屬於 bw1 的 byte 數。

3. 當他們(bw1、bw2、bw3)都不是零的時候，就開始進行比較，進而找出頻寬最大的那條路並印出來，並將頻寬最大的用 add_flow 的函式加進去，hard_timeout 設定為零，這樣就可以一直跑最大頻寬的那條。

-Part3: Problems encountered

1. 在寫 controller1.py 的時候，他一直跑出 error，找了很久也問了同學，但都不知道原因出在哪，後來突然發現是縮排，有一行不小心多了一格，那一個空格好小，根本不會發現，還好後來問題解決了。

2. 在 task5 的時候，遇到一個問題是一直跑不出封包數，無論我等多久都只有 loading…和 instantiating…那六行，就連我把 terminal 甚至是 VM 關掉重開也都無法解決，後來不知道為什麼亂試一試，又突然好了，最後發現在 Mininet exit 後，打個 mn -c，就能解決這個問題。

3. 在測量頻寬時，想了很久到底要不要指定 port 5566，還是就讓他自己跑到 port 5001 就好，後來看到助教回覆這次的 lab 沒有指定 port，就決定也不指定了。還有在做 iperf 的 output 一直都只有一行 0-10 秒的 bandwidth，後來發現在 h2 那邊我忘了加上-i 1，讓他的 interval 變一秒，加了之後終於得到我要的結果了。

4. 一開始總封包數減掉 loss 的封包一直差 1，也不知道怎麼解決，後來發現有其他同學也有這個問題，然後助教有回覆說這是因為 iperf 內部會有一些 hasdshake，所以會有差 1 的現象產生，不過如果把 client 的指令中的- i 1 拿掉就不會差了。

5. Task6 要如何下手，真的是一個很難的問題，想了好幾天還是沒有頭緒，和同學討論了很久也都沒有結果，有點不太確定教授和助教要我們做的究竟是什麼，最後我自己理解出了一個方法(part 2)，助教好像有說會 remove 一條 link 或是 add flow，但我不確定我自己測試的方法是否正確。

6. 在做 Task6 時遇到 hard_timeout 的問題，原本規定 hard_timeout 要是 5，但因為 5 秒過完 path 3 可能還不會收到封包，就換到 path2 了，於是就將 hard_timeout 延長至 20。

7. 在 Task6 裡遇到另一個問題是我的判斷條件是用 bw1、bw2、bw3 不等於零就輸出最佳 path，這樣的話，在第四十秒的地方就會先輸出，於是我想改用 hard_timeout 來判斷，但它一直說宣告錯誤，我也查詢不到應如何宣告，於是決定還是用一樣的方式，但 40-60 秒的輸出並不是真的最佳 path，要等到 60 秒後印出的才是對的。

Discussion

1. **Describe the differences between packet-in and packet-out in detail.**
   Packet-in 是將接收到的封包，轉送至 controller，packet-out 則是將接收到從 controller 送來的封包，轉送至指定的 port。首先 Switch 要先利用 packet-in 來學習 MAC 的 address，接著 controller 也要利用 packet-in 接收來自 switch 的封包，然後進行分析，得到連接 host 的 MAC address 還有一些 port 的相關資料，學習完之後要將封包進行轉送，在那些學習過的 host 資料裡找封包的目的地，如果已經有紀錄，那就利用 packet-out 來轉送至對應的 port，如果沒紀錄的話，就利用 packet-out 來達到 flooding。

2. **What is “table-miss” in SDN?**
   如果在某個 Flow Table 都找不到相對應的 Flow Entry，就稱為 table-miss。如果發生這種情況的話，可能會直接把封包丟掉，或是轉至其他的 Flow Table 繼續 pipeline 處理過程，又或者是轉送至 controller。

3. **Why is “(app_manager.RyuApp)” adding after the declaration of class in SimpleController.py?**
   如果在 module 中定義了兩個以上的 Ryu 程式，則 app_manager 是用名稱來排序，而且還會用第一個來當作這個 module 中的 Ryu 程式，並且執行。
4. **What is the meaning of “datapath” in SimpleController.py?**
   datapath 是指 topology 中使用 openflow 的 switch，每次要新增一個 flow 到 flow table 時，都要藉由 datapath 取得 switch 的資料。

5. **Why need to set “eth_type=0x0800” in the flow entry?**
   eth_type=0x0800 是十六進制的 IPv4 protocol，因為有很多種類的 protocols 會通過 Ethernet，所以在 flow entry 中需要設定，這樣才能確保在兩終端間能夠傳遞訊息。

6. **Compare the differences between the iPerf results of SimpleController.py, controller1.py and controller2.py. Which forwarding rule is better? Why?**

   SimpleController.py:
   * Average bandwidth: 1000 Kbits/sec
   * Delay: 0.333 ms
   * Loss: 5%

   controller1.py:
   * Average bandwidth: 976 Kbits/sec
   * Delay: 0.152 ms
   * Loss: 7.4%

   controller2.py:
   * Average bandwidth: 1.01 Mbits/sec
   * Delay: 0.221 ms
   * Loss: 3.7%

   Compare

* Average bandwidth: controller2.py > SimpleController.py > controller1.py
* Delay: SimpleController.py > controller2.py > controller1.py
* Loss: controller1.py > SimpleController.py > controller2.py

以 Delay 來看，controller1.py 的 Delay 是最少的，再來是 controller2.py，最後才是 SimpleController.py，接著，無論是觀察 Loss 的比率或是平均的頻寬皆是 controller2.py 最佳，再者是 SimpleController.py，最後才是 controller2.py。這和我原先預想的有點不大相同，原本以為無論是 Delay、Loss 的比率或是平均的頻寬，都應該是 controller2.py 最佳，但因為 Delay 的差距不算到太大，所以 Loss 的比率才是重點，所以綜合下來，我認為還是 controller2.py 最佳，雖然他可能傳送的比較久，但它的遺失率比其他都還低。