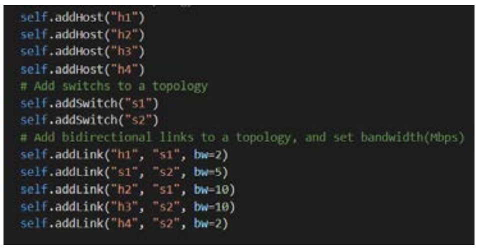
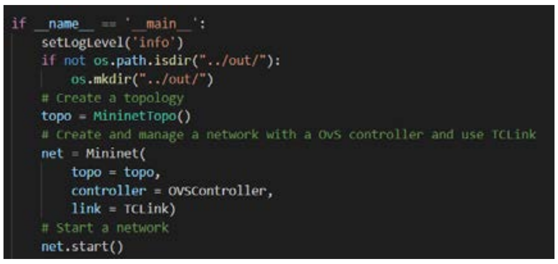
1. Describe each step and how to run your program

Topo_TCP.py



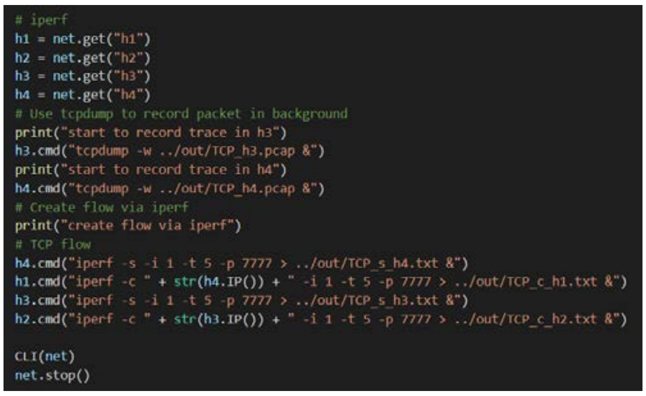
```python
self.addHost("h1")
self.addHost("h2")
self.addHost("h3")
self.addHost("h4")
# Add switchs to a topology
self.addSwitch("s1")
self.addSwitch("s2")
# Add bidirectional links to a topology, and set bandwidth(Mbps)
self.addLink("h1", "s1", bw=2)
self.addLink("s1", "s2", bw=5)
self.addLink("h2", "s1", bw=10)
self.addLink("h3", "s2", bw=10)
self.addLink("h4", "s2", bw=2)
```

設置 host 和 switch



```python
if __name__ == '__main__':
    setLogLevel('info')
    if not os.path.isdir("../out/"):
        os.mkdir("../out/")
    # Create a topology
    topo = MininetTopo()
    # Create and manage a network with a OvS controller and use TCLink
    net = Mininet(
        topo = topo,
        controller = OVSController,
        link = TCLink)
    # Start a network
    net.start()
```

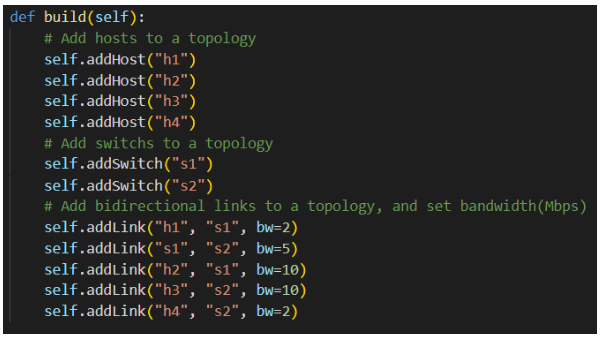
創建拓樸



```python
# iperf
h1 = net.get("h1")
h2 = net.get("h2")
h3 = net.get("h3")
h4 = net.get("h4")
# Use tcpdump to record packet in background
print("start to record trace in h3")
h3.cmd("tcpdump -w ../out/TCP_h3.pcap &")
print("start to record trace in h4")
h4.cmd("tcpdump -w ../out/TCP_h4.pcap &")
# Create flow via iperf
print("create flow via iperf")
# TCP flow
h4.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h4.txt &")
h1.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h1.txt &")
h3.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3.txt &")
h2.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h2.txt &")

CLI(net)
net.stop()
```

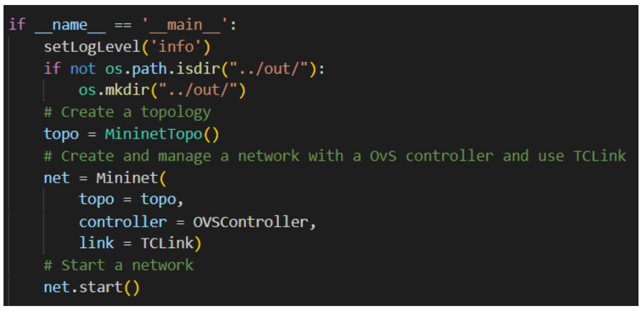
進行 iperf 計算

Topo_UDP.py



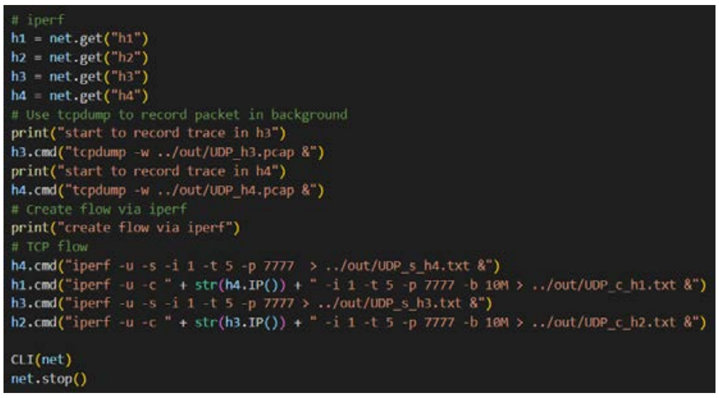
```python
def build(self):
    # Add hosts to a topology
    self.addHost("h1")
    self.addHost("h2")
    self.addHost("h3")
    self.addHost("h4")
    # Add switchs to a topology
    self.addSwitch("s1")
    self.addSwitch("s2")
    # Add bidirectional links to a topology, and set bandwidth(Mbps)
    self.addLink("h1", "s1", bw=2)
    self.addLink("s1", "s2", bw=5)
    self.addLink("h2", "s1", bw=10)
    self.addLink("h3", "s2", bw=10)
    self.addLink("h4", "s2", bw=2)
```

設置 host 和 switch



```python
if __name__ == '__main__':
    setLogLevel('info')
    if not os.path.isdir("../out/"):
        os.mkdir("../out/")
    # Create a topology
    topo = MininetTopo()
    # Create and manage a network with a OvS controller and use TCLink
    net = Mininet(
        topo = topo,
        controller = OVSController,
        link = TCLink)
    # Start a network
    net.start()
```

創建拓樸

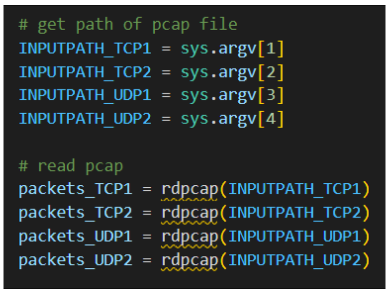


```python
# iperf
h1 = net.get("h1")
h2 = net.get("h2")
h3 = net.get("h3")
h4 = net.get("h4")
# Use tcpdump to record packet in background
print("start to record trace in h3")
h3.cmd("tcpdump -w ../out/UDP_h3.pcap &")
print("start to record trace in h4")
h4.cmd("tcpdump -w ../out/UDP_h4.pcap &")
# Create flow via iperf
print("create flow via iperf")
# TCP flow
h4.cmd("iperf -u -s -i 1 -t 5 -p 7777  > ../out/UDP_s_h4.txt &")
h1.cmd("iperf -u -c " + str(h4.IP()) + " -i 1 -t 5 -p 7777 -b 10M > ../out/UDP_c_h1.txt &")
h3.cmd("iperf -u -s -i 1 -t 5 -p 7777 > ../out/UDP_s_h3.txt &")
h2.cmd("iperf -u -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 -b 10M > ../out/UDP_c_h2.txt &")

CLI(net)
net.stop()
```

進行 iperf 計算

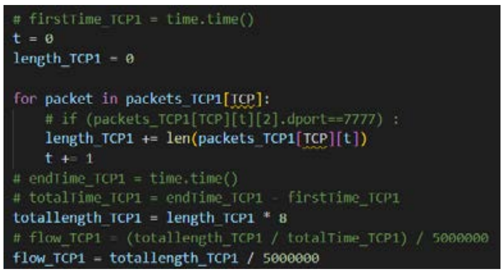
(-u:用 UDP，-b 10M:將 bandwidth 設為 10Mbps)

computRate.py



```python
# get path of pcap file
INPUTPATH_TCP1 = sys.argv[1]
INPUTPATH_TCP2 = sys.argv[2]
INPUTPATH_UDP1 = sys.argv[3]
INPUTPATH_UDP2 = sys.argv[4]

# read pcap
packets_TCP1 = rdpcap(INPUTPATH_TCP1)
packets_TCP2 = rdpcap(INPUTPATH_TCP2)
packets_UDP1 = rdpcap(INPUTPATH_UDP1)
packets_UDP2 = rdpcap(INPUTPATH_UDP2)
```

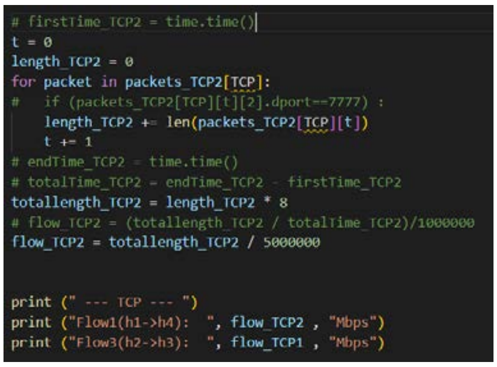
讀取 TCP_h3.pcap、TCP_h4.pcap、UDP_h3.pcap、UDP_h4.pcap



```python
# firstTime_TCP1 = time.time()
t = 0
length_TCP1 = 0

for packet in packets_TCP1[TCP]:
    # if (packets_TCP1[TCP][t][2].dport==7777) :
    length_TCP1 += len(packets_TCP1[TCP][t])
    t += 1
# endTime_TCP1 = time.time()
# totalTime_TCP1 = endTime_TCP1 - firstTime_TCP1
totallength_TCP1 = length_TCP1 * 8
# flow_TCP1 = (totallength_TCP1 / totalTime_TCP1) / 5000000
flow_TCP1 = totallength_TCP1 / 5000000
```

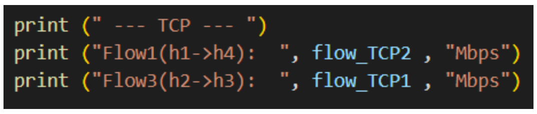
在 TCP_h3.pcap 中，先用 for 迴圈將 TCP layer 的所有 packet size，再將其*8 變成 bit，接著除以 1000000 再除以 5 秒，這就是題目所要求的

flow(Mbps)



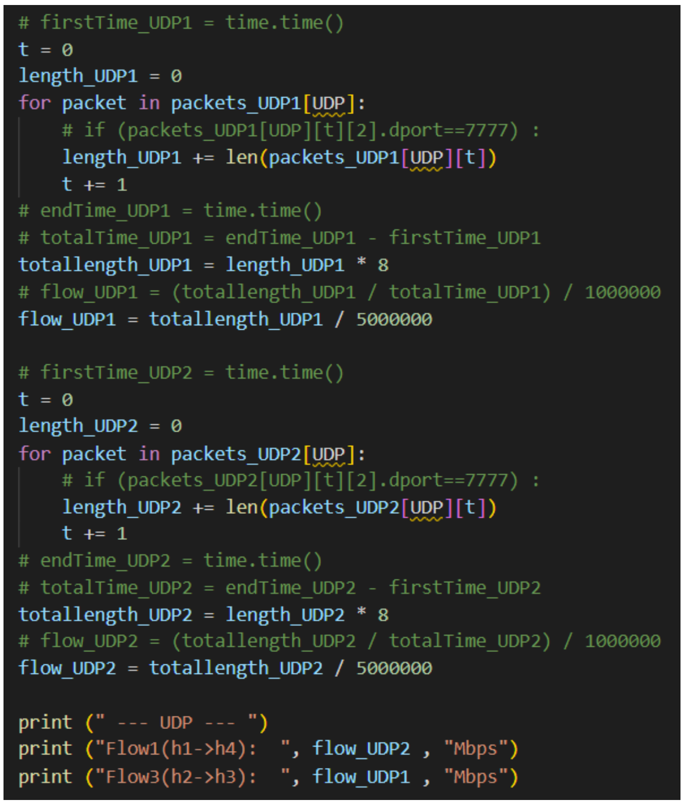
```python
# firstTime_TCP2 = time.time()
t = 0
length_TCP2 = 0
for packet in packets_TCP2[TCP]:
#   if (packets_TCP2[TCP][t][2].dport==7777) :
    length_TCP2 += len(packets_TCP2[TCP][t])
    t += 1
# endTime_TCP2 = time.time()
# totalTime_TCP2 = endTime_TCP2 - firstTime_TCP2
totallength_TCP2 = length_TCP2 * 8
# flow_TCP2 = (totallength_TCP2 / totalTime_TCP2)/1000000
flow_TCP2 = totallength_TCP2 / 5000000


print (" --- TCP --- ")
print ("Flow1(h1->h4):  ", flow_TCP2 , "Mbps")
print ("Flow3(h2->h3):  ", flow_TCP1 , "Mbps")
```

TCP_h4.pcap 重複上述步驟



```python
print (" --- TCP --- ")
print ("Flow1(h1->h4):  ", flow_TCP2 , "Mbps")
print ("Flow3(h2->h3):  ", flow_TCP1 , "Mbps")
```

Print 出 TCP 答案



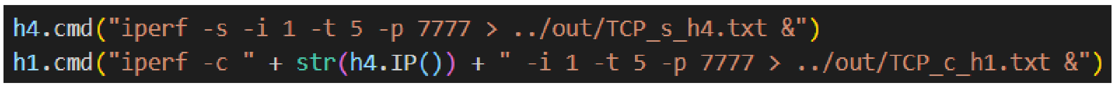
```python
# firstTime_UDP1 = time.time()
t = 0
length_UDP1 = 0
for packet in packets_UDP1[UDP]:
    # if (packets_UDP1[UDP][t][2].dport==7777) :
    length_UDP1 += len(packets_UDP1[UDP][t])
    t += 1
# endTime_UDP1 = time.time()
# totalTime_UDP1 = endTime_UDP1 - firstTime_UDP1
totallength_UDP1 = length_UDP1 * 8
# flow_UDP1 = (totallength_UDP1 / totalTime_UDP1) / 1000000
flow_UDP1 = totallength_UDP1 / 5000000

# firstTime_UDP2 = time.time()
t = 0
length_UDP2 = 0
for packet in packets_UDP2[UDP]:
    # if (packets_UDP2[UDP][t][2].dport==7777) :
    length_UDP2 += len(packets_UDP2[UDP][t])
    t += 1
# endTime_UDP2 = time.time()
# totalTime_UDP2 = endTime_UDP2 - firstTime_UDP2
totallength_UDP2 = length_UDP2 * 8
# flow_UDP2 = (totallength_UDP2 / totalTime_UDP2) / 1000000
flow_UDP2 = totallength_UDP2 / 5000000

print (" --- UDP --- ")
print ("Flow1(h1->h4):  ", flow_UDP2 , "Mbps")
print ("Flow3(h2->h3):  ", flow_UDP1 , "Mbps")
```

UDP_h3.pcap 和 UDP_h4.pcap 也重複上述三步驟

2. Describe your observations from the results in the lab
   i. TCP 和 UDP 兩者的 flow 數值接近
   ii. 因為 UDP 不具有 congestion controller，從 wireshock 可以明顯看到有封包丟失
   iii. 在 UDP 中，h1->h4 的頻寬小於 h2->h3 的頻寬，因此丟失的封包也較多

3. Answer the following question in short:
   i. What does each iPerfcommand you used mean?

TCP



```
h4.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h4.txt &")
h1.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h1.txt &")
```

-s:以 server 模式執行 iperf

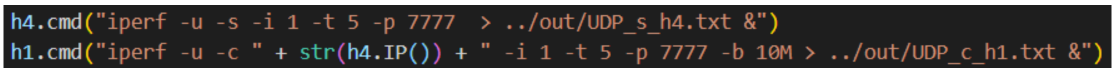
-i 1:指定輸出數據的時間間隔為 1

-t 5:指定傳輸設據測試時間為 5

-p 7777:指定 port 為 7777

-c:以 client 模式執行 iperf

UDP



```
h4.cmd("iperf -u -s -i 1 -t 5 -p 7777  > ../out/UDP_s_h4.txt &")
h1.cmd("iperf -u -c " + str(h4.IP()) + " -i 1 -t 5 -p 7777 -b 10M > ../out/UDP_c_h1.txt &")
```

-u:使用 UDP 協定

-b 10M:target bandwidth 設置為 10M bits/s

ii. What is your command to filter each flow in Wireshark?

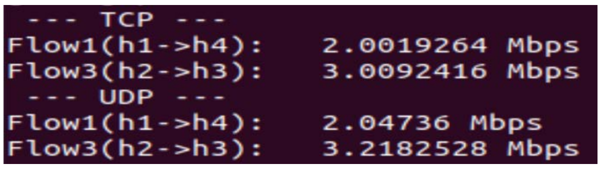
TCP_h3.pcap:ip.src==10.0.0.2 and ip.dst==10.0.0.3

TCP_h4.pcap:ip.src==10.0.0.1 and ip.dst==10.0.0.4

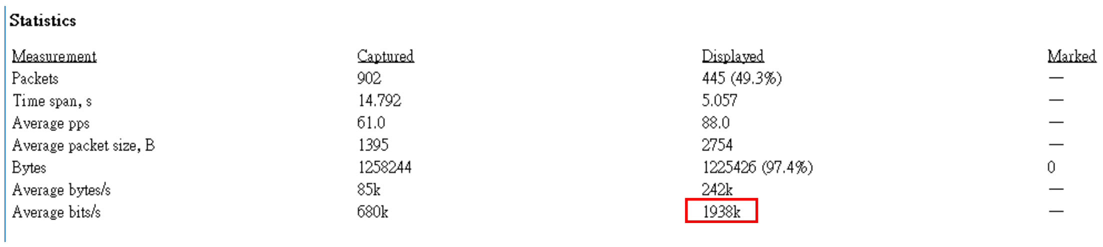
UDP_h3.pcap:ip.src==10.0.0.2 and ip.dst==10.0.0.3

UDP_h4.pcap:ip.src==10.0.0.1 and ip.dst==10.0.0.4

iii. Show the results of computeRate.py and statistics of Wireshark



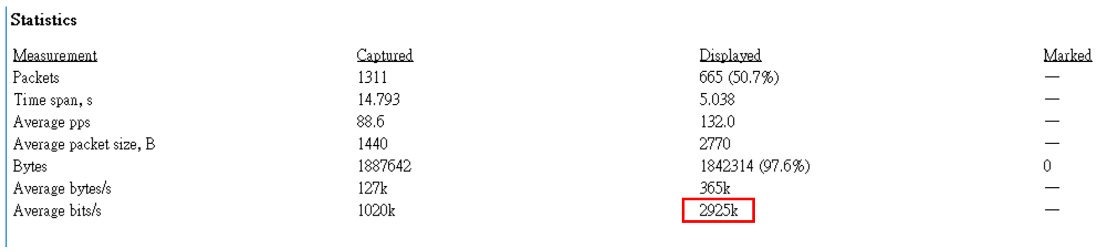
```
 --- TCP ---
Flow1(h1->h4):    2.0019264 Mbps
Flow3(h2->h3):    3.0092416 Mbps
 --- UDP ---
Flow1(h1->h4):    2.04736 Mbps
Flow3(h2->h3):    3.2182528 Mbps
```



Statistics

Measurement	Captured	Displayed	Marked
Packets	902	445 (49.3%)	—
Time span, s	14.792	5.057	—
Average pps	61.0	88.0	—
Average packet size, B	1395	2754	—
Bytes	1258244	1225426 (97.4%)	0
Average bytes/s	85k	242k	—
Average bits/s	680k	1938k	—

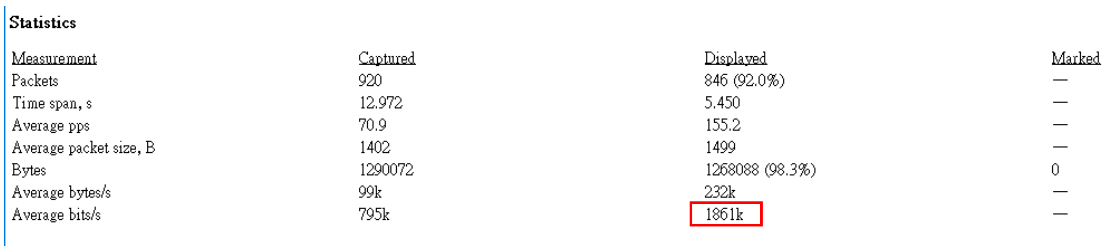
Flow1(TCP)



Statistics

Measurement	Captured	Displayed	Marked
Packets	1311	665 (50.7%)	—
Time span, s	14.793	5.038	—
Average pps	88.6	132.0	—
Average packet size, B	1440	2770	—
Bytes	1887642	1842314 (97.6%)	0
Average bytes/s	127k	365k	—
Average bits/s	1020k	2925k	—

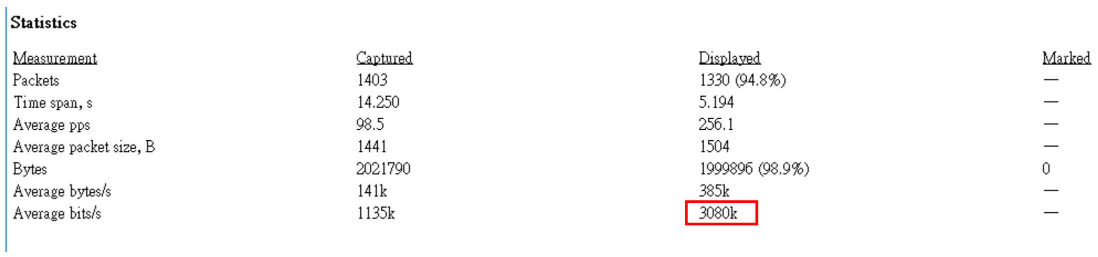
Flow3(TCP)



Statistics

Measurement	Captured	Displayed	Marked
Packets	920	846 (92.0%)	—
Time span, s	12.972	5.450	—
Average pps	70.9	155.2	—
Average packet size, B	1402	1499	—
Bytes	1290072	1268088 (98.3%)	0
Average bytes/s	99k	232k	—
Average bits/s	795k	1861k	—

Flow1(UDP)



Statistics

Measurement	Captured	Displayed	Marked
Packets	1403	1330 (94.8%)	—
Time span, s	14.250	5.194	—
Average pps	98.5	256.1	—
Average packet size, B	1441	1504	—
Bytes	2021790	1999896 (98.9%)	0
Average bytes/s	141k	385k	—
Average bits/s	1135k	3080k	—

Flow3(UDP)

iv. Does the throughput match the bottleneck throughput of the path?

由上方截圖可知，兩者結果吻合

v. Do you observe the same throughput from TCP and UDP? Can both flows equally share the bandwidth?

TCP 和 UDP 的結果相近。可以共用，因為 Flow1 加 Flow3 皆為 5

4. What you have learned from this lab?

在 lab 的操作過程中，對於 TCP 和 UDP 的執行過程和差異更加了解，經過一

次次的錯誤嘗試，清楚各種語法和指令的用途，同時，因為和同學不斷討論，進而發現自己的觀念漏洞。

5. What difficulty you have met in this lab?

第一次接觸到跟 iperf 有關的東西，因此對於指令的使用十分不熟悉，沒有搞清楚題目的要求導致前期花很多時間做白工，後來在 computRate 也因為觀念不清楚亂寫一堆判斷式，幸虧有同學和室友的提點，我才能順利完成。