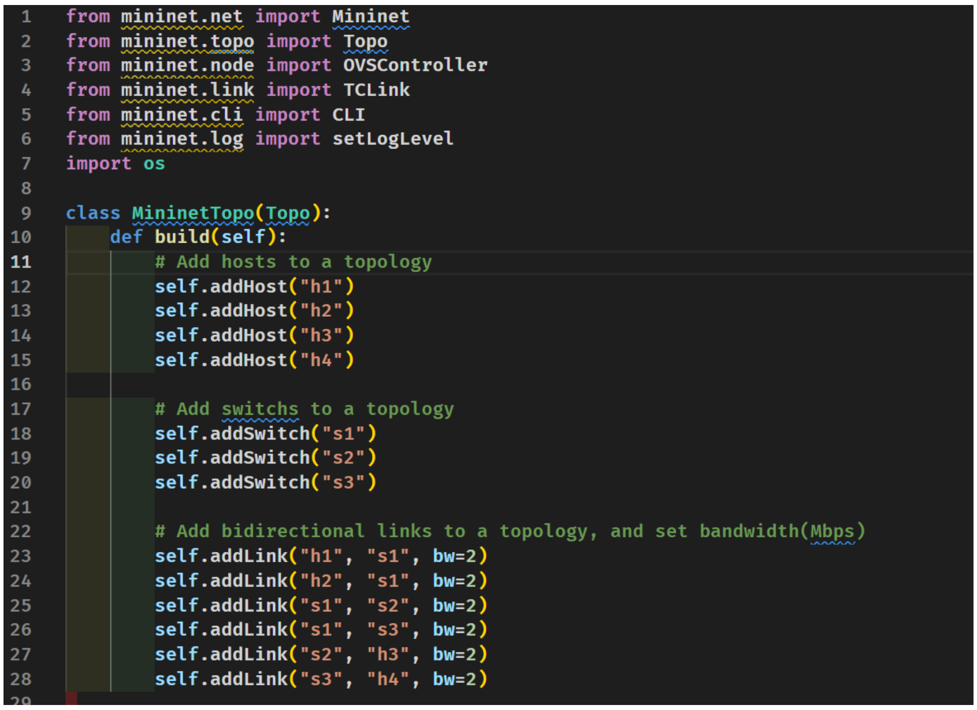
Computer Networks
Lab1

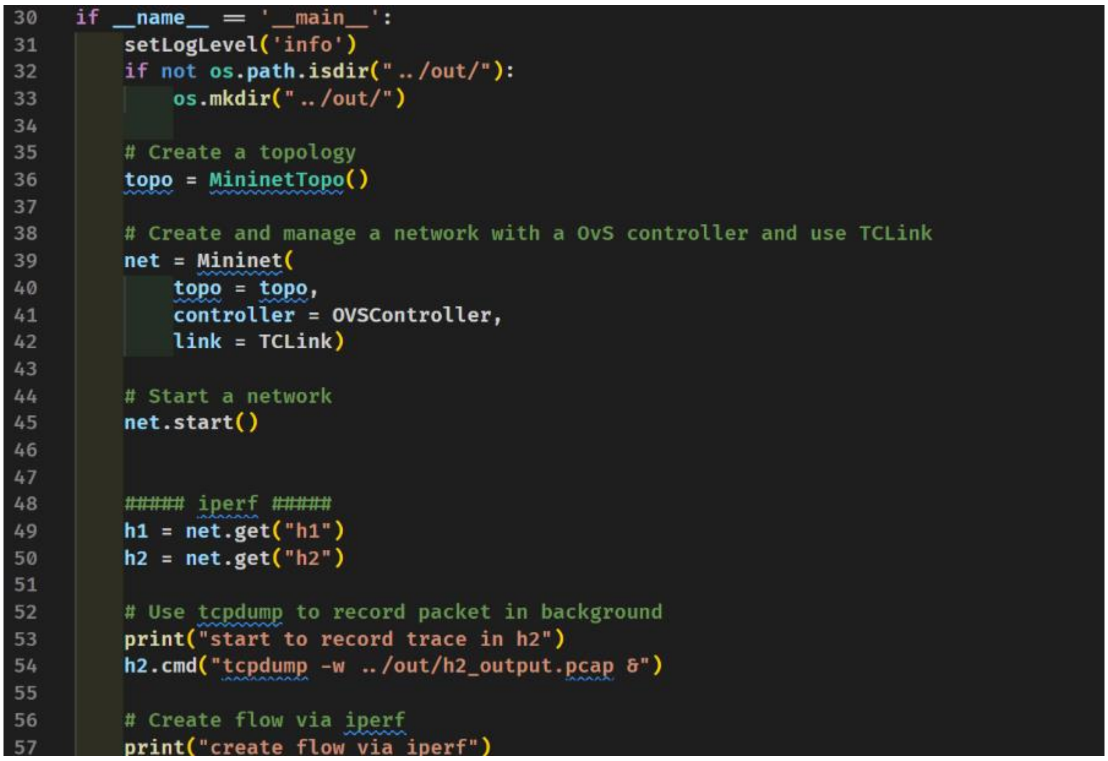
■ Each step and how to run my program：

Topo.py(Modify topo.py and create the following new network topology)

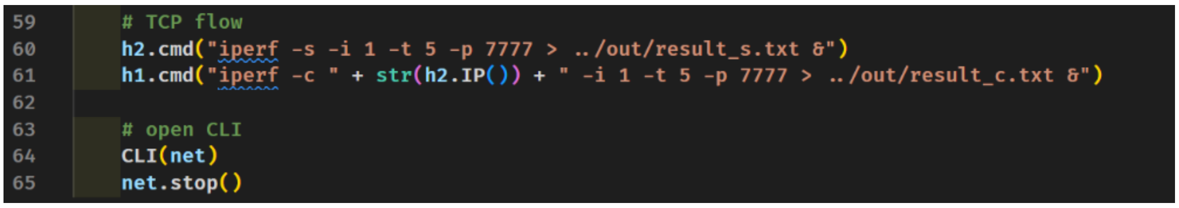
1. Import necessary Mininet modules and libraries.
2. Define a custom Mininet topology (MininetTopo) by inheriting from the Topo class and implementing the build method. This method adds hosts ("h1", "h2", "h3", "h4") and switches ("s1", "s2", "s3") to the topology and creates bidirectional links between them with specified bandwidth (2 Mbps).
3. Set the log level for Mininet.
4. Check if the "../out/" directory exists and create it if not.
5. Create an instance of the custom topology (MininetTopo).
6. Create a Mininet network with the specified topology, an OvS controller, and use TCLink for links.
7. Start the Mininet network.
8. Perform iperf measurements.
9. Open the Mininet command-line interface (CLI).
10. Stop the Mininet network.



```python
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import OVSController
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel
import os

class MininetTopo(Topo):
    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")
        self.addSwitch("s3")

        # Add bidirectional links to a topology, and set bandwidth(Mbps)
        self.addLink("h1", "s1", bw=2)
        self.addLink("h2", "s1", bw=2)
        self.addLink("s1", "s2", bw=2)
        self.addLink("s1", "s3", bw=2)
        self.addLink("s2", "h3", bw=2)
        self.addLink("s3", "h4", bw=2)
```



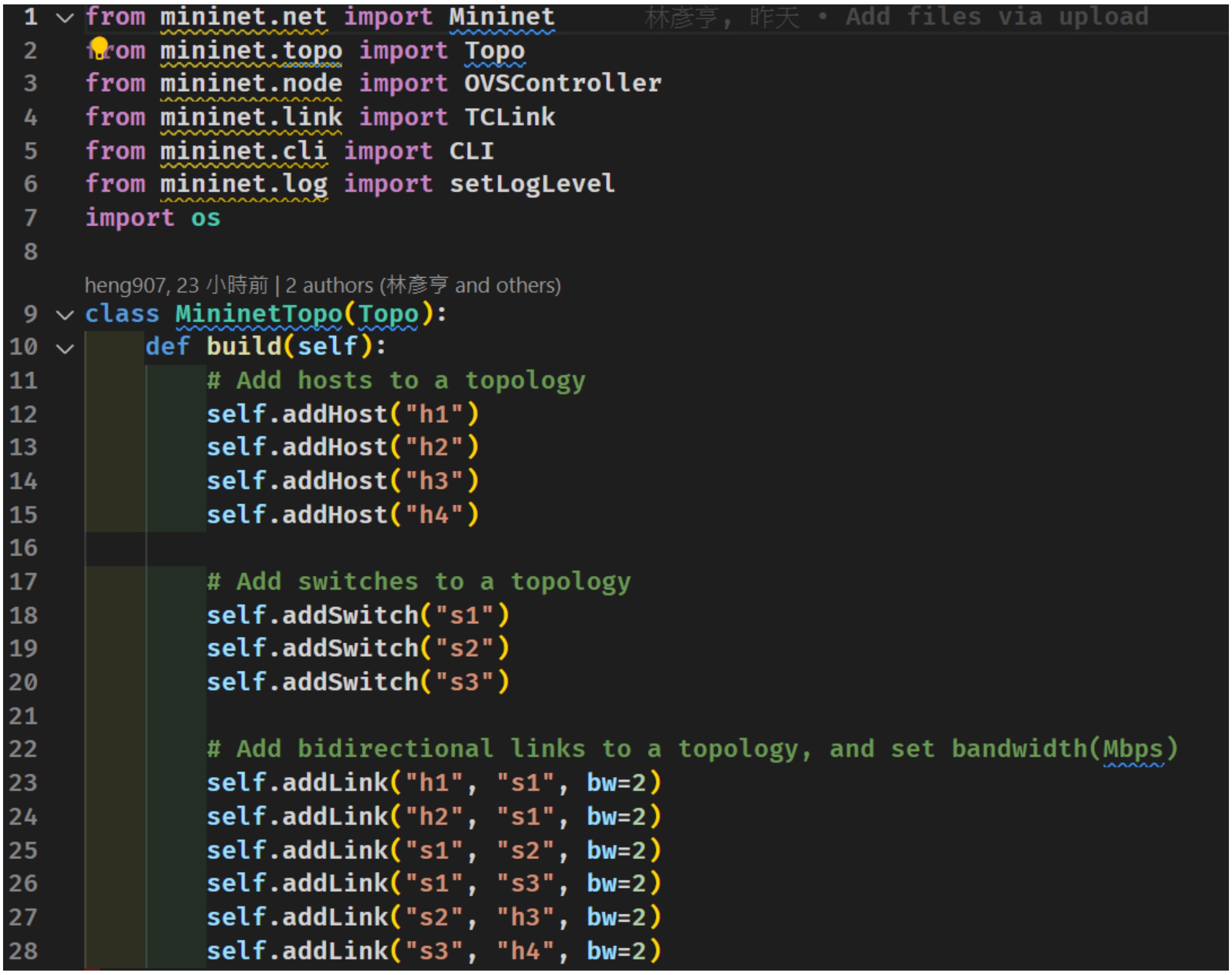
```python
if __name__ == '__main__':
    setLogLevel('info')
    if not os.path.isdir("../out/"):
        os.mkdir("../out/")

    # 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()


    ##### iperf #####
    h1 = net.get("h1")
    h2 = net.get("h2")

    # Use tcpdump to record packet in background
    print("start to record trace in h2")
    h2.cmd("tcpdump -w ../out/h2_output.pcap &")

    # Create flow via iperf
    print("create flow via iperf")
```



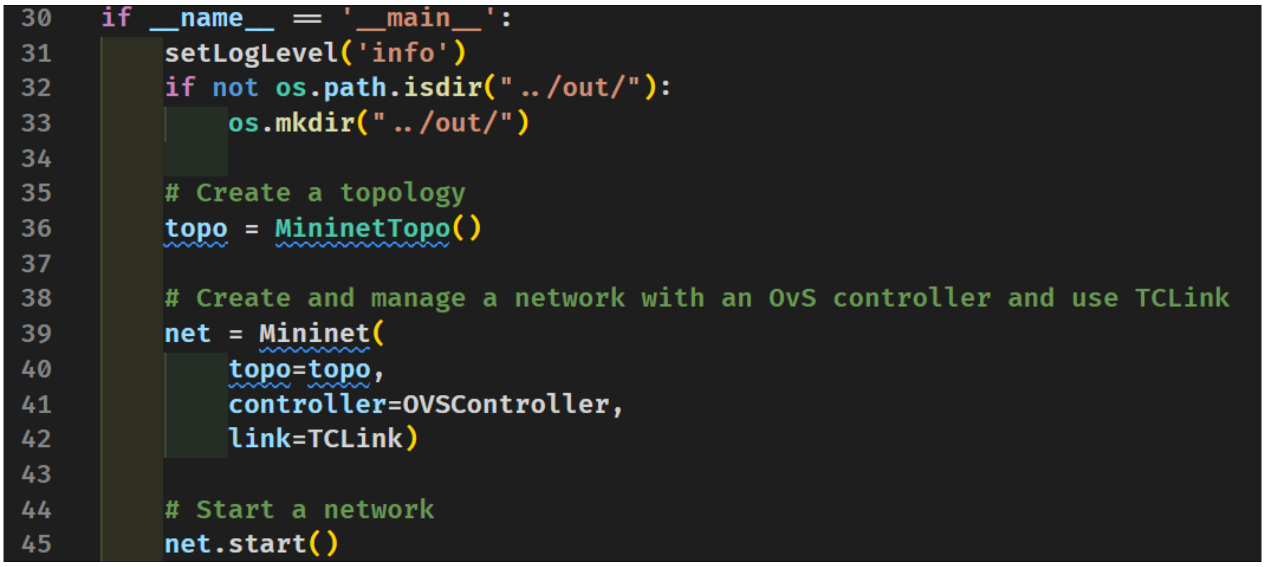
```python
    # TCP flow
    h2.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/result_s.txt &")
    h1.cmd("iperf -c " + str(h2.IP()) + " -i 1 -t 5 -p 7777 > ../out/result_c.txt &")

    # open CLI
    CLI(net)
    net.stop()
```

Topo_TCP.py：(generate two TCP flows from h1 to h3 and one TCP flow from h2 to h4)

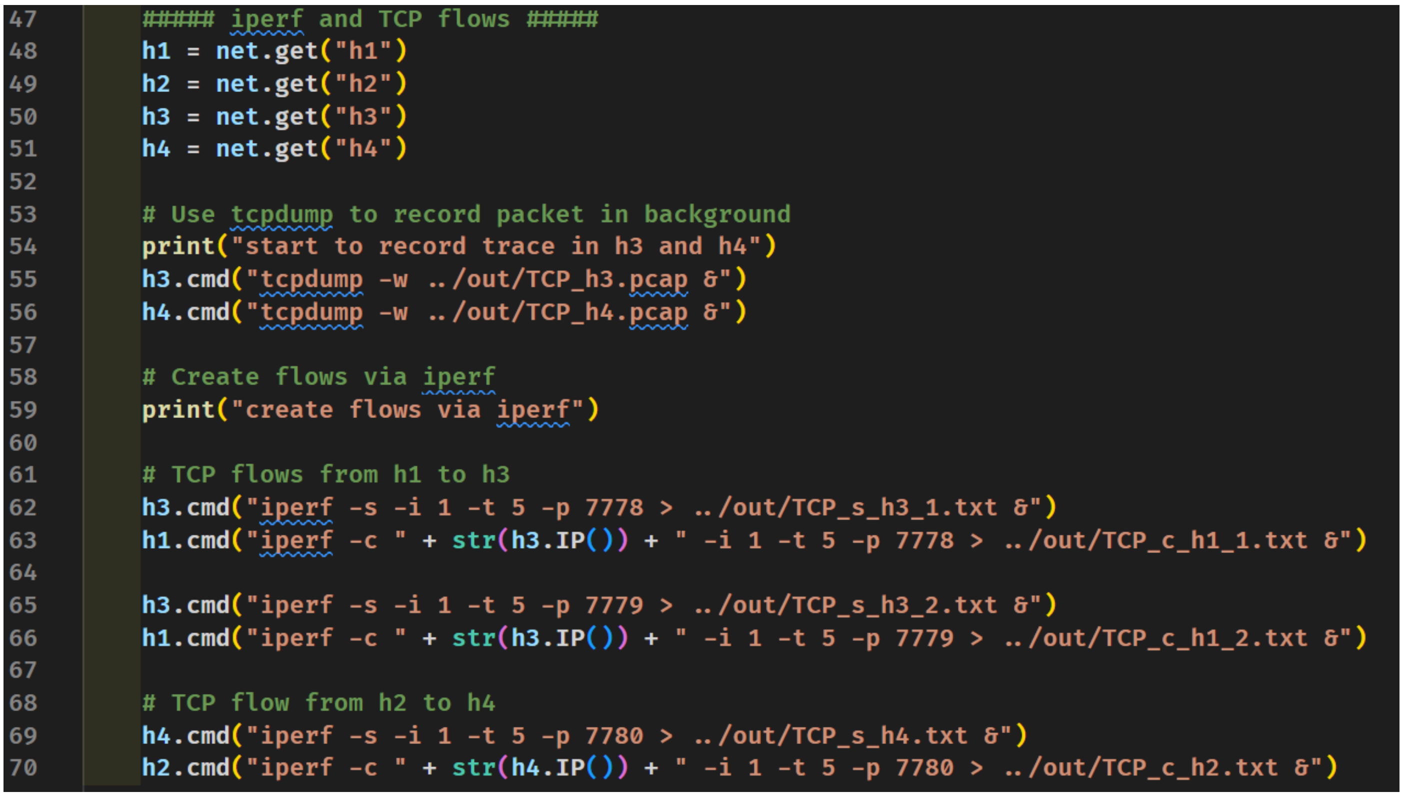
1. Import Mininet modules and libraries.
2. Define a custom Mininet topology (MininetTopo) by inheriting from the Topo class and implementing the build method. This method adds hosts ("h1", "h2", "h3", "h4") and switches ("s1", "s2", "s3") to the topology and creates bidirectional links between them with specified bandwidth (2 Mbps).
3. Set the log level for Mininet.
4. Check if the "../out/" directory exists and create it if not.
5. Create an instance of the custom topology (MininetTopo).
6. Create a Mininet network with the specified topology, an OvS controller, and use TCLink for links.
7. Start the Mininet network.
8. Perform iperf measurements and TCP flows.
   - Get references to hosts h1, h2, h3, and h4.
   - Start tcpdump on h3 and h4 to record packets.
   - Create TCP flows using iperf.
9. Open the Mininet command-line interface (CLI).
10. Stop the Mininet network.



```python
from mininet.net import Mininet        林彥亨, 昨天 • Add files via upload
from mininet.topo import Topo
from mininet.node import OVSController
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel
import os

heng907, 23 小時前 | 2 authors (林彥亨 and others)
class MininetTopo(Topo):
    def build(self):
        # Add hosts to a topology
        self.addHost("h1")
        self.addHost("h2")
        self.addHost("h3")
        self.addHost("h4")

        # Add switches to a topology
        self.addSwitch("s1")
        self.addSwitch("s2")
        self.addSwitch("s3")

        # Add bidirectional links to a topology, and set bandwidth(Mbps)
        self.addLink("h1", "s1", bw=2)
        self.addLink("h2", "s1", bw=2)
        self.addLink("s1", "s2", bw=2)
        self.addLink("s1", "s3", bw=2)
        self.addLink("s2", "h3", bw=2)
        self.addLink("s3", "h4", bw=2)
```



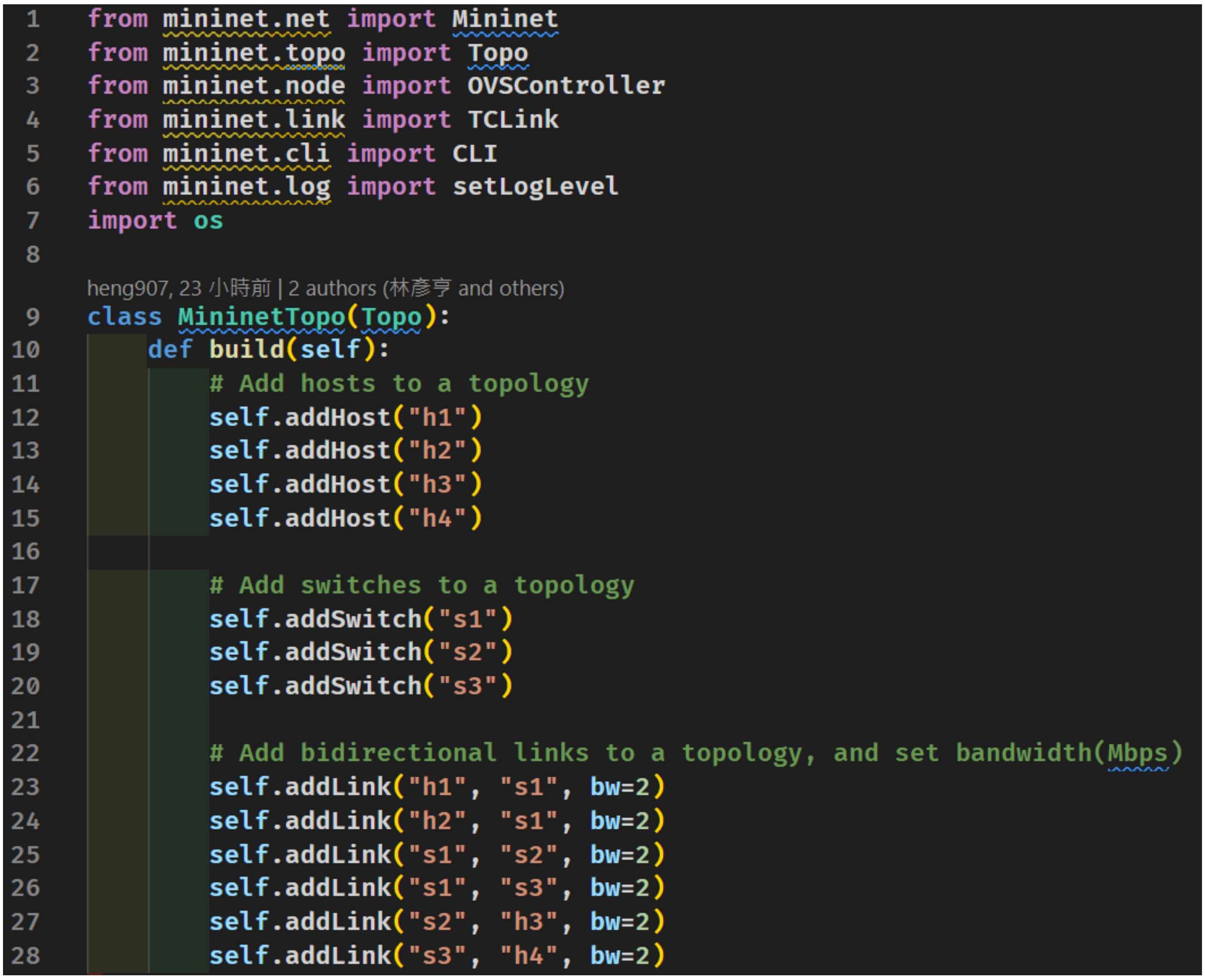
```python
if __name__ == '__main__':
    setLogLevel('info')
    if not os.path.isdir("../out/"):
        os.mkdir("../out/")

    # Create a topology
    topo = MininetTopo()

    # Create and manage a network with an OvS controller and use TCLink
    net = Mininet(
        topo=topo,
        controller=OVSController,
        link=TCLink)

    # Start a network
    net.start()
```



```python
    ##### iperf and TCP flows #####
    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 and h4")
    h3.cmd("tcpdump -w ../out/TCP_h3.pcap &")
    h4.cmd("tcpdump -w ../out/TCP_h4.pcap &")

    # Create flows via iperf
    print("create flows via iperf")

    # TCP flows from h1 to h3
    h3.cmd("iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_1.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7778 > ../out/TCP_c_h1_1.txt &")

    h3.cmd("iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h3_2.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7779 > ../out/TCP_c_h1_2.txt &")

    # TCP flow from h2 to h4
    h4.cmd("iperf -s -i 1 -t 5 -p 7780 > ../out/TCP_s_h4.txt &")
    h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7780 > ../out/TCP_c_h2.txt &")
```

Topo_UDP.py(generate two UDP flows from h1 to h3 and one UDP flow from h2 to h4. (three flows in total))：

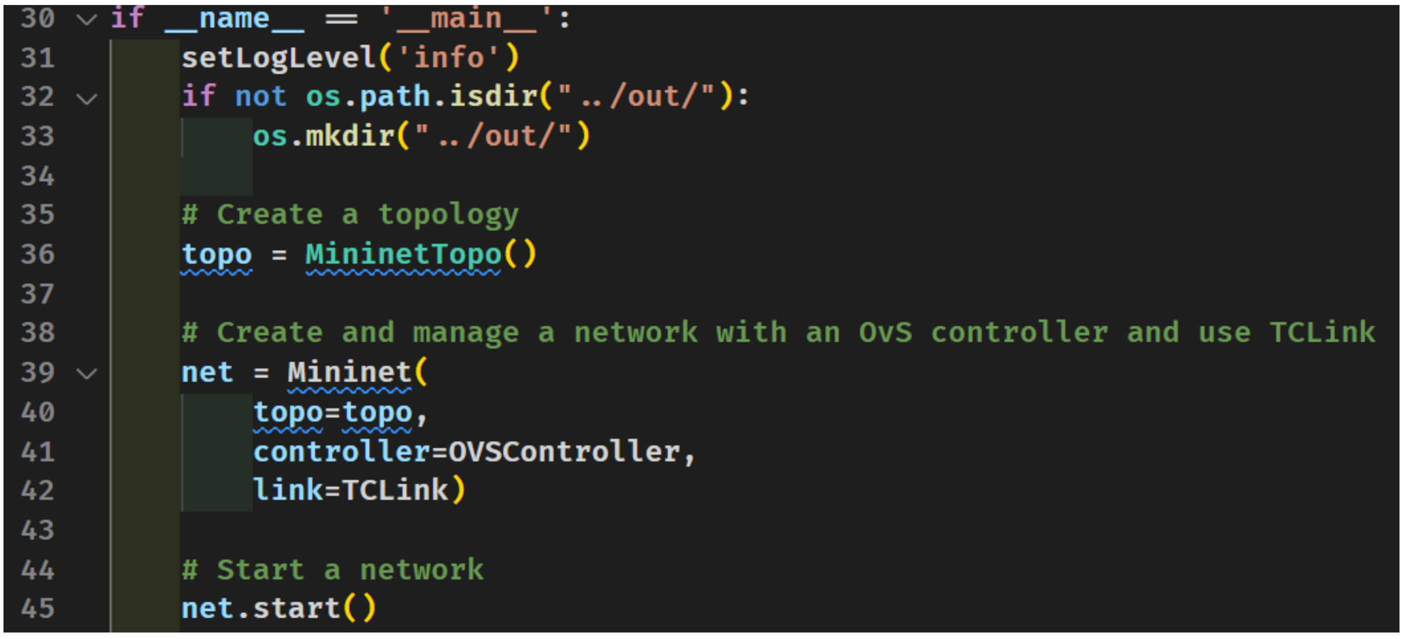
1. Import Mininet modules and libraries.
2. Define a custom Mininet topology (MininetTopo) by inheriting from the Topo class and implementing the build method. This method adds hosts ("h1", "h2", "h3", "h4") and switches ("s1", "s2", "s3") to the topology and creates bidirectional links between them with specified bandwidth (2 Mbps).
3. Set the log level for Mininet.
4. Check if the "../out/" directory exists and create it if not.
5. Create an instance of the custom topology (MininetTopo).
6. Create a Mininet network with the specified topology, an OvS controller, and use TCLink for links.
7. Start the Mininet network.
8. Perform iperf measurements and TCP flows.
   - Get references to hosts h1, h2, h3, and h4.
   - Start tcpdump on h3 and h4 to record packets.
   - Create UDP flows using iperf.
9. Open the Mininet command-line interface (CLI).
10. Stop the Mininet network.



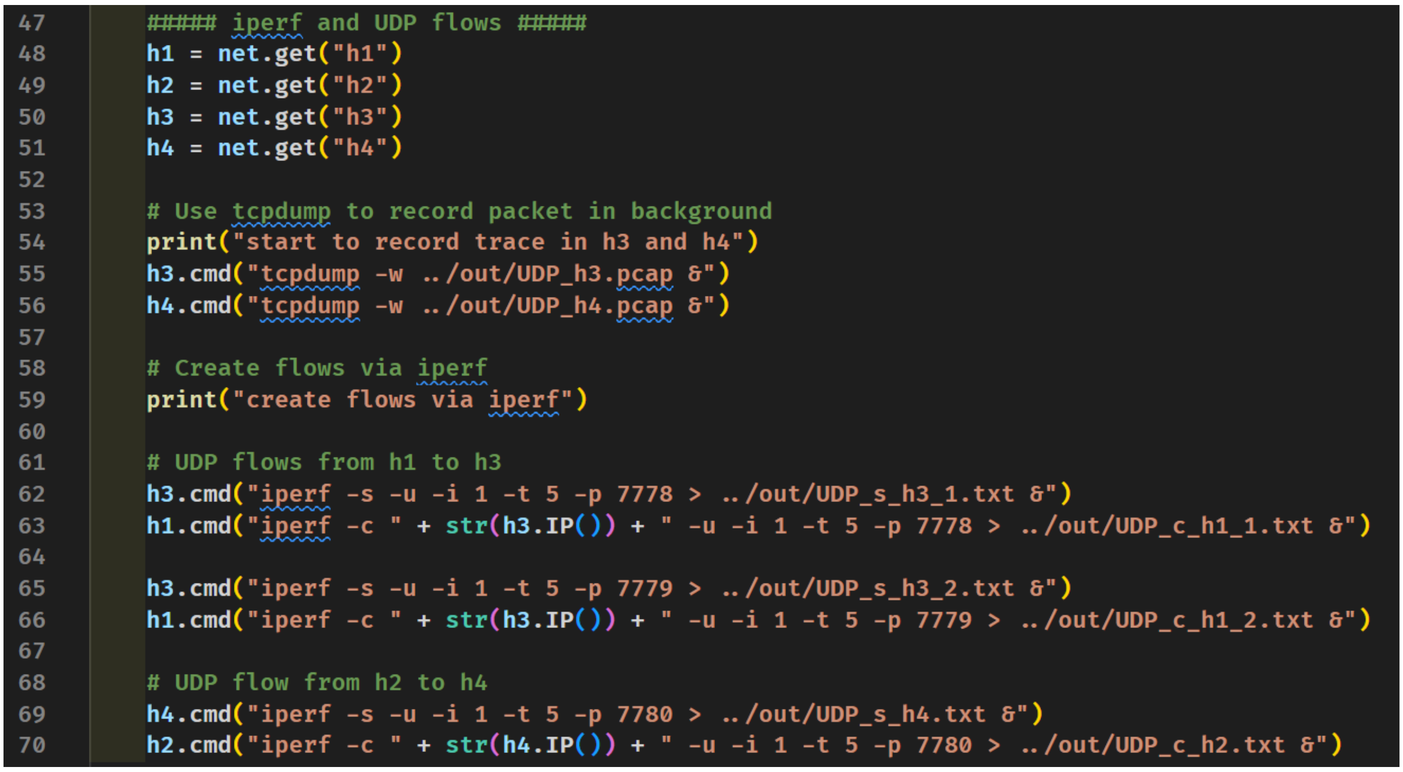
```python
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import OVSController
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel
import os

```

heng907, 23 小時前 | 2 authors (林彥亨 and others)

```python
class MininetTopo(Topo):
    def build(self):
        # Add hosts to a topology
        self.addHost("h1")
        self.addHost("h2")
        self.addHost("h3")
        self.addHost("h4")

        # Add switches to a topology
        self.addSwitch("s1")
        self.addSwitch("s2")
        self.addSwitch("s3")

        # Add bidirectional links to a topology, and set bandwidth(Mbps)
        self.addLink("h1", "s1", bw=2)
        self.addLink("h2", "s1", bw=2)
        self.addLink("s1", "s2", bw=2)
        self.addLink("s1", "s3", bw=2)
        self.addLink("s2", "h3", bw=2)
        self.addLink("s3", "h4", bw=2)
```



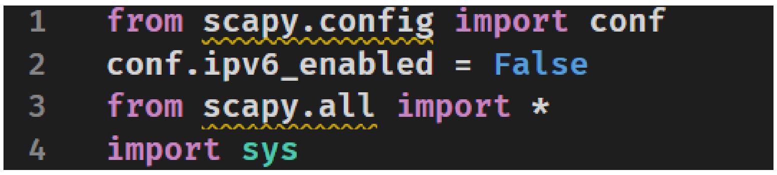
```python
if __name__ == '__main__':
    setLogLevel('info')
    if not os.path.isdir("../out/"):
        os.mkdir("../out/")

    # Create a topology
    topo = MininetTopo()

    # Create and manage a network with an OvS controller and use TCLink
    net = Mininet(
        topo=topo,
        controller=OVSController,
        link=TCLink)

    # Start a network
    net.start()
```



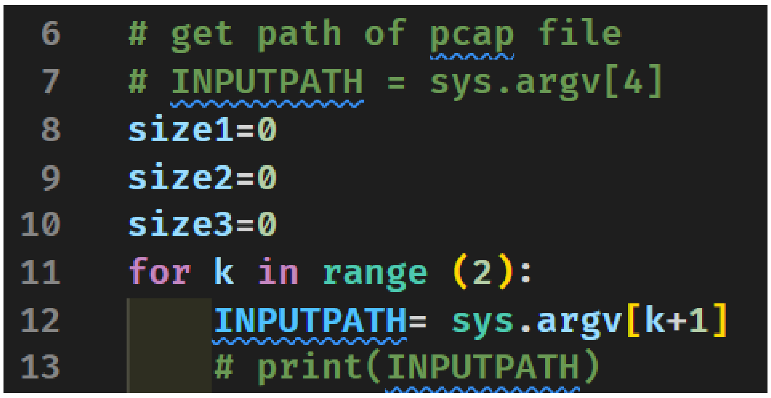
```python
##### iperf and UDP flows #####
h1 = net.get("h1")
h2 = net.get("h2")
h3 = net.get("h3")
h4 = net.get("h4")

# Use tcpdump to record packet in background
print("start to record trace in h3 and h4")
h3.cmd("tcpdump -w ../out/UDP_h3.pcap &")
h4.cmd("tcpdump -w ../out/UDP_h4.pcap &")

# Create flows via iperf
print("create flows via iperf")

# UDP flows from h1 to h3
h3.cmd("iperf -s -u -i 1 -t 5 -p 7778 > ../out/UDP_s_h3_1.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -u -i 1 -t 5 -p 7778 > ../out/UDP_c_h1_1.txt &")

h3.cmd("iperf -s -u -i 1 -t 5 -p 7779 > ../out/UDP_s_h3_2.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -u -i 1 -t 5 -p 7779 > ../out/UDP_c_h1_2.txt &")

# UDP flow from h2 to h4
h4.cmd("iperf -s -u -i 1 -t 5 -p 7780 > ../out/UDP_s_h4.txt &")
h2.cmd("iperf -c " + str(h4.IP()) + " -u -i 1 -t 5 -p 7780 > ../out/UDP_c_h2.txt &")
```

computeRate.py(to compute throughput of each flow in Task 3)：

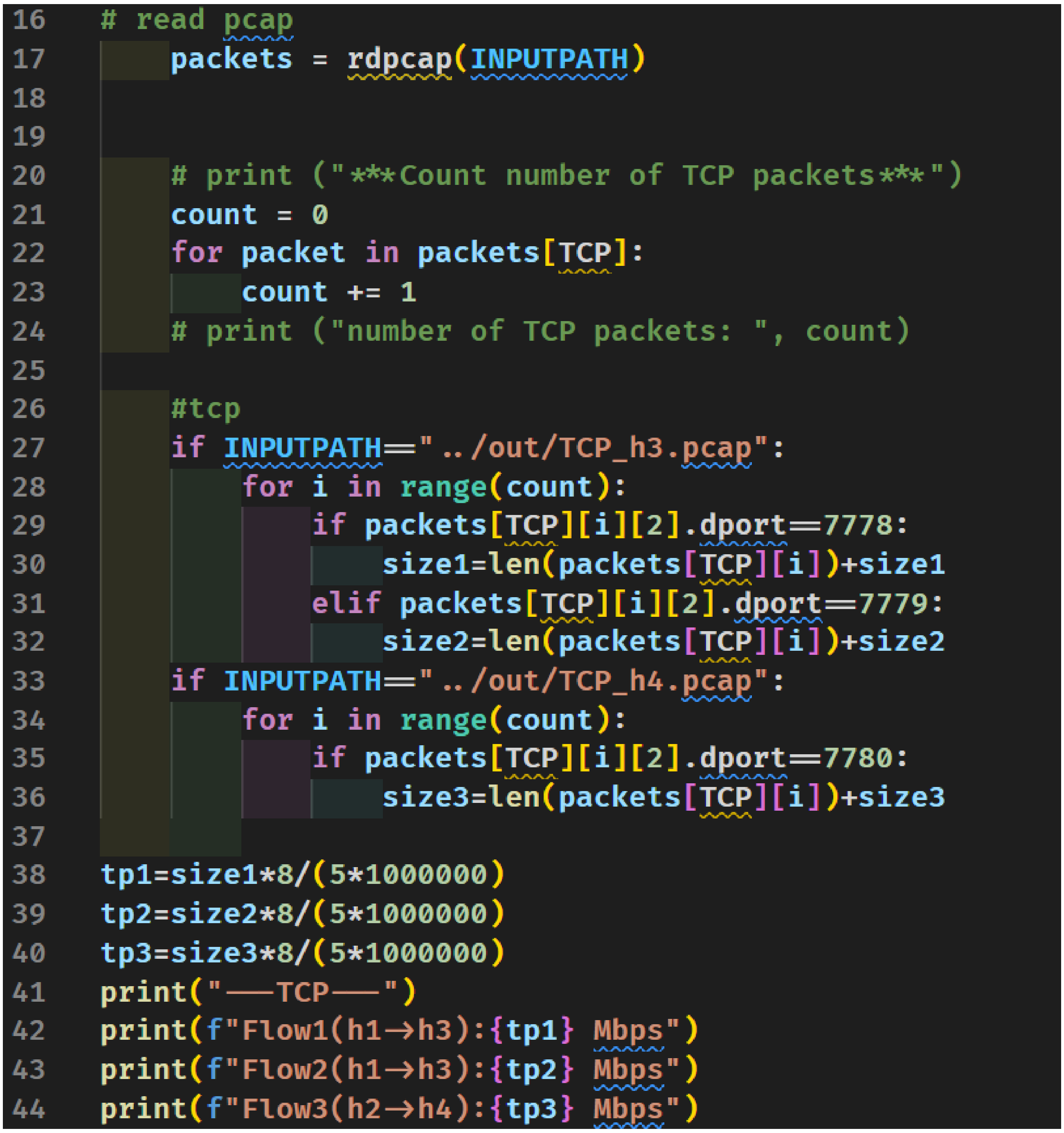
1. Initialization: The script initializes variables to store packet sizes for TCP flows (size1, size2, size3).
2. Loop for TCP flows: The script loops over the TCP pcap files (../out/TCP_h3.pcap and ../out/TCP_h4.pcap). For each file, it reads the pcap and counts the number of TCP packets.
3. TCP Flow Analysis: Based on the file path, the script analyzes the TCP flows. It checks the destination ports and accumulates the packet sizes for different flows.
4. Compute TCP Throughputs: Using the accumulated packet sizes, the script calculates the throughput for each TCP flow (tp1, tp2, tp3).
5. Repeat for UDP Flows: The script then reinitializes variables and repeats the process for UDP flows (../out/UDP_h3.pcap and ../out/UDP_h4.pcap).
6. Compute UDP Throughputs: Similar to TCP flows, the script calculates the throughput for each UDP flow (tp1, tp2, tp3).
7. Print Results: Finally, the script prints the computed throughput values for TCP and UDP flows.



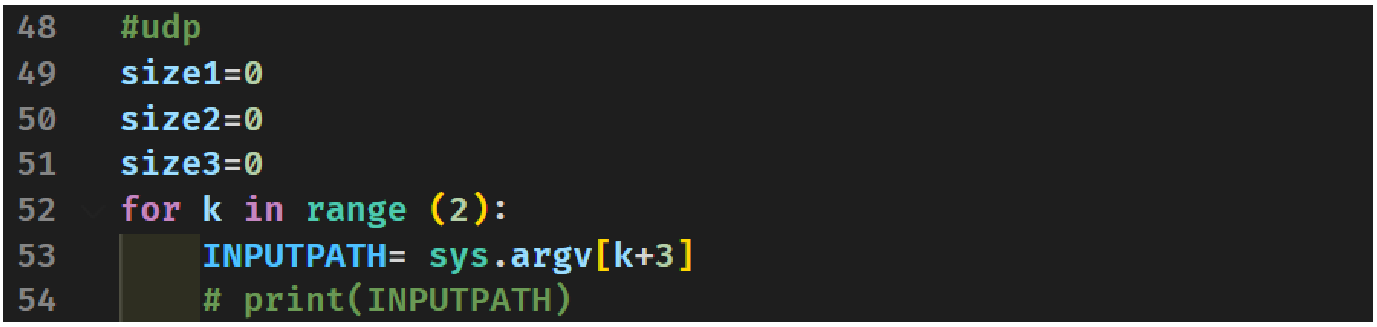
```python
from scapy.config import conf
conf.ipv6_enabled = False
from scapy.all import *
import sys
```



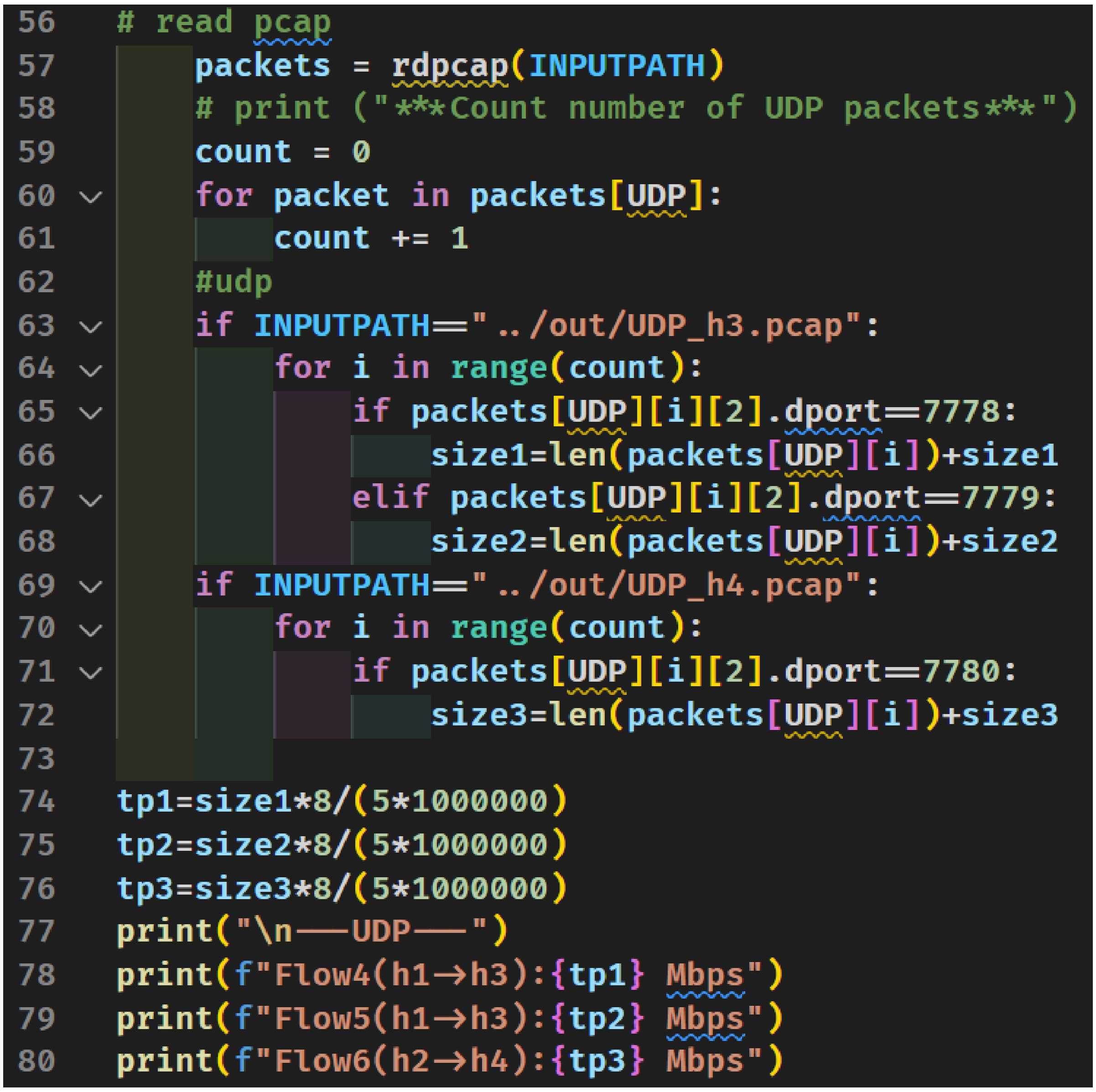
```python
# get path of pcap file
# INPUTPATH = sys.argv[4]
size1=0
size2=0
size3=0
for k in range (2):
    INPUTPATH= sys.argv[k+1]
    # print(INPUTPATH)
```



```python
# read pcap
    packets = rdpcap(INPUTPATH)


    # print ("***Count number of TCP packets***")
    count = 0
    for packet in packets[TCP]:
        count += 1
    # print ("number of TCP packets: ", count)

    #tcp
    if INPUTPATH=="../out/TCP_h3.pcap":
        for i in range(count):
            if packets[TCP][i][2].dport==7778:
                size1=len(packets[TCP][i])+size1
            elif packets[TCP][i][2].dport==7779:
                size2=len(packets[TCP][i])+size2
    if INPUTPATH=="../out/TCP_h4.pcap":
        for i in range(count):
            if packets[TCP][i][2].dport==7780:
                size3=len(packets[TCP][i])+size3

tp1=size1*8/(5*1000000)
tp2=size2*8/(5*1000000)
tp3=size3*8/(5*1000000)
print("——TCP——")
print(f"Flow1(h1->h3):{tp1} Mbps")
print(f"Flow2(h1->h3):{tp2} Mbps")
print(f"Flow3(h2->h4):{tp3} Mbps")
```



```python
#udp
size1=0
size2=0
size3=0
for k in range (2):
    INPUTPATH= sys.argv[k+3]
    # print(INPUTPATH)
```



```python
# read pcap
    packets = rdpcap(INPUTPATH)
    # print ("***Count number of UDP packets***")
    count = 0
    for packet in packets[UDP]:
        count += 1
    #udp
    if INPUTPATH=="../out/UDP_h3.pcap":
        for i in range(count):
            if packets[UDP][i][2].dport==7778:
                size1=len(packets[UDP][i])+size1
            elif packets[UDP][i][2].dport==7779:
                size2=len(packets[UDP][i])+size2
    if INPUTPATH=="../out/UDP_h4.pcap":
        for i in range(count):
            if packets[UDP][i][2].dport==7780:
                size3=len(packets[UDP][i])+size3

tp1=size1*8/(5*1000000)
tp2=size2*8/(5*1000000)
tp3=size3*8/(5*1000000)
print("\n---UDP---")
print(f"Flow4(h1->h3):{tp1} Mbps")
print(f"Flow5(h1->h3):{tp2} Mbps")
print(f"Flow6(h2->h4):{tp3} Mbps")
```

Observation：

In this lab1 provides a comprehensive example of creating, configuring, and analyzing a Mininet network with traffic generation and capture.It would be beneficial to include more comments in the code to enhance readability and clarify the purpose of each section. Additionally, error handling and validation for command-line arguments could be improved to handle unexpected input more gracefully.

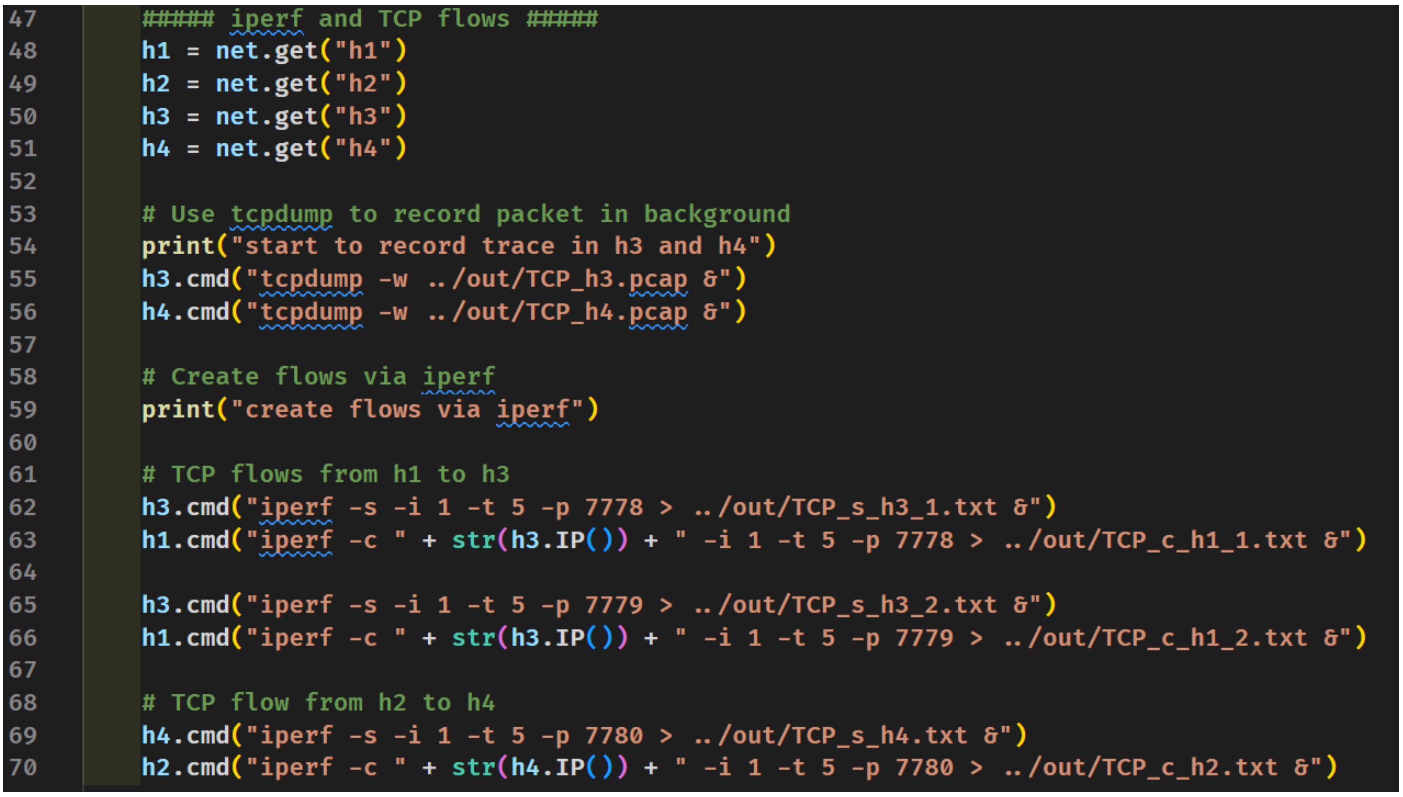
Questions：

- What does each iPerf command you used mean?

ANS：

In topo_TCP.py：

1. This starts tcpdump on hosts h3 and h4 to capture and record the network packets. The captured packets will be saved in the specified pcap files (../out/TCP_h3.pcap and ../out/TCP_h4.pcap).
2. For TCP flows from h1 to h3:
   - h3 acts as the server (iperf -s) with port 7778, and the output is redirected to ../out/TCP_s_h3_1.txt.
   - h1 acts as the client (iperf -c) connecting to h3's IP address and port 7778. The client's output is redirected to ../out/TCP_c_h1_1.txt.
   - The same process is repeated with a different port (7779) and output file names for a second TCP flow.
3. For the TCP flow from h2 to h4:
   - Similar to the previous flows, h4 acts as the server (iperf -s) with port 7780, and the output is redirected to ../out/TCP_s_h4.txt.
   - h2 acts as the client (iperf -c) connecting to h4's IP address and port 7780, with the client's output redirected to ../out/TCP_c_h2.txt.



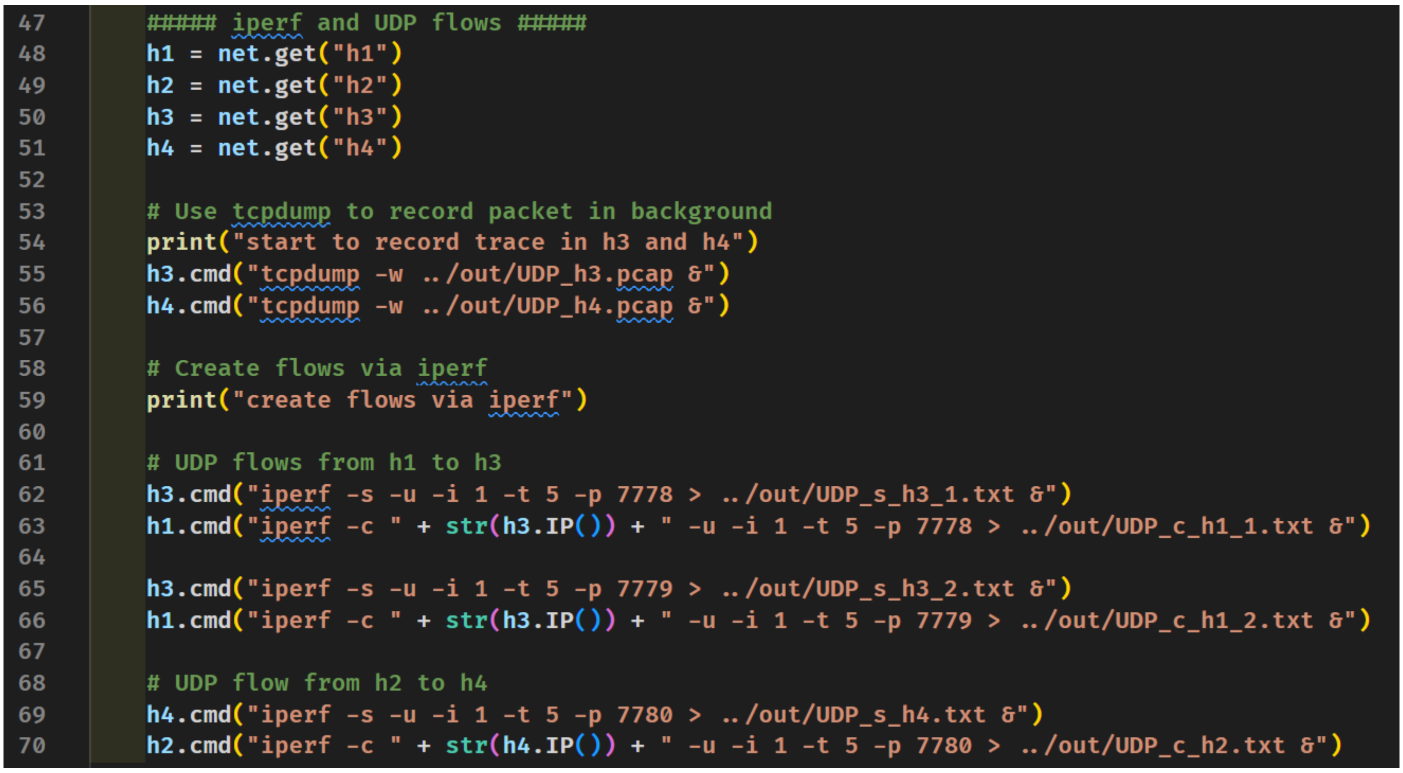
```
    ##### iperf and TCP flows #####
    h1 = net.get("h1")
    h2 = net.get("h2")
    h3 = net.get("h3")
    h4 = net.get("h4")

    # Use tcpdump to record packet in background
    print("start to record trace in h3 and h4")
    h3.cmd("tcpdump -w ../out/TCP_h3.pcap &")
    h4.cmd("tcpdump -w ../out/TCP_h4.pcap &")

    # Create flows via iperf
    print("create flows via iperf")

    # TCP flows from h1 to h3
    h3.cmd("iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_1.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7778 > ../out/TCP_c_h1_1.txt &")

    h3.cmd("iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h3_2.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7779 > ../out/TCP_c_h1_2.txt &")

    # TCP flow from h2 to h4
    h4.cmd("iperf -s -i 1 -t 5 -p 7780 > ../out/TCP_s_h4.txt &")
    h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7780 > ../out/TCP_c_h2.txt &")
```

In topo_UDP.py：

1. UDP flow from h1 to h3:
   - h3.cmd("iperf -s -u -i 1 -t 5 -p 7778 > ../out/UDP_s_h3_1.txt &"): This command starts iperf in server mode (-s) with UDP (-u) on host h3, listening on port 7778. It sets the reporting interval to 1 second (-i 1), runs for 5 seconds (-t 5), and redirects the output to a file named "UDP_s_h3_1.txt" in the "../out/" directory. The & at the end runs the command in the background.
   - h1.cmd("iperf -c " + str(h3.IP()) + " -u -i 1 -t 5 -p 7778 > ../out/UDP_c_h1_1.txt &"): This command starts iperf in client mode (-c) on host h1, connecting to the IP address of h3 (str(h3.IP())) on port 7778 with UDP. The other options are similar to the server side, and the output is redirected to "UDP_c_h1_1.txt" in the "../out/" directory.
2. Another UDP flow from h1 to h3 (different port):
   - Similar to the first flow, but using a different port (7779) and output file ("UDP_s_h3_2.txt" and "UDP_c_h1_2.txt").
3. UDP flow from h2 to h4:
   - h4.cmd("iperf -s -u -i 1 -t 5 -p 7780 > ../out/UDP_s_h4.txt &"): This command starts iperf in server mode on host h4, listening on port 7780 for UDP traffic, with similar options as before. The output is

redirected to "UDP_s_h4.txt".

- h2.cmd("iperf -c " + str(h4.IP()) + " -u -i 1 -t 5 -p 7780 > ../out/UDP_c_h2.txt &"): This command starts iperf in client mode on host h2, connecting to the IP address of h4 on port 7780 with UDP. The output is redirected to "UDP_c_h2.txt".



```
##### iperf and UDP flows #####
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 and h4")
h3.cmd("tcpdump -w ../out/UDP_h3.pcap &")
h4.cmd("tcpdump -w ../out/UDP_h4.pcap &")

# Create flows via iperf
print("create flows via iperf")

# UDP flows from h1 to h3
h3.cmd("iperf -s -u -i 1 -t 5 -p 7778 > ../out/UDP_s_h3_1.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -u -i 1 -t 5 -p 7778 > ../out/UDP_c_h1_1.txt &")

h3.cmd("iperf -s -u -i 1 -t 5 -p 7779 > ../out/UDP_s_h3_2.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -u -i 1 -t 5 -p 7779 > ../out/UDP_c_h1_2.txt &")

# UDP flow from h2 to h4
h4.cmd("iperf -s -u -i 1 -t 5 -p 7780 > ../out/UDP_s_h4.txt &")
h2.cmd("iperf -c " + str(h4.IP()) + " -u -i 1 -t 5 -p 7780 > ../out/UDP_c_h2.txt &")
```

- What is your command to filter each flow in Wireshark?

ANS：

I used following command to filter each flow in wireshark.

“tcp.port == 7778” to filter the TCP flow1 in TCP_h3.

“tcp.port == 7779” to filter the TCP flow2 in TCP_h3.

“tcp.port == 7780” to filter the TCP flow3 in TCP_h4.

“udp.port == 7778” to filter the UDP flow1 in UDP_h3.

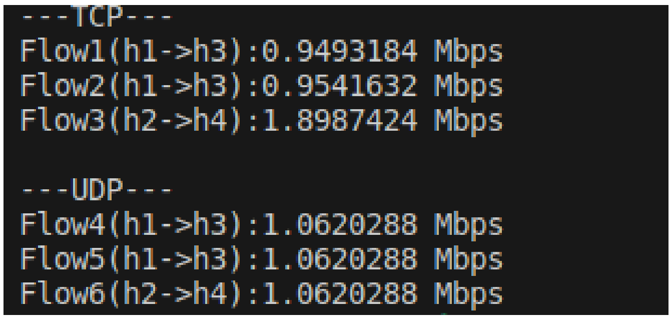
“udp.port == 7779” to filter the UDP flow2 in UDP_h3.

“udp.port == 7780” to filter the UDP flow3 in UDP_h4.

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

ANS：

Result of computeRate.py

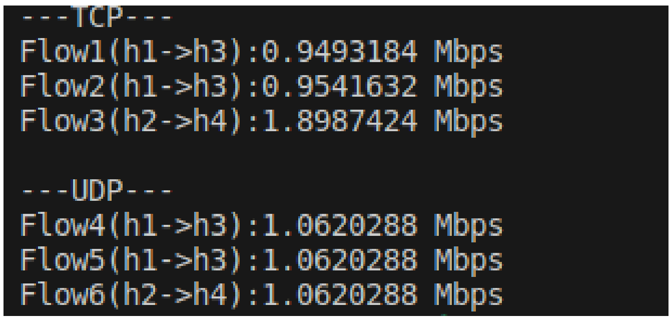


```
---TCP---
Flow1(h1->h3):0.9493184 Mbps
Flow2(h1->h3):0.9541632 Mbps
Flow3(h2->h4):1.8987424 Mbps

---UDP---
Flow4(h1->h3):1.0620288 Mbps
Flow5(h1->h3):1.0620288 Mbps
Flow6(h2->h4):1.0620288 Mbps
```

- Does the throughput match the bottleneck throughput of the path?

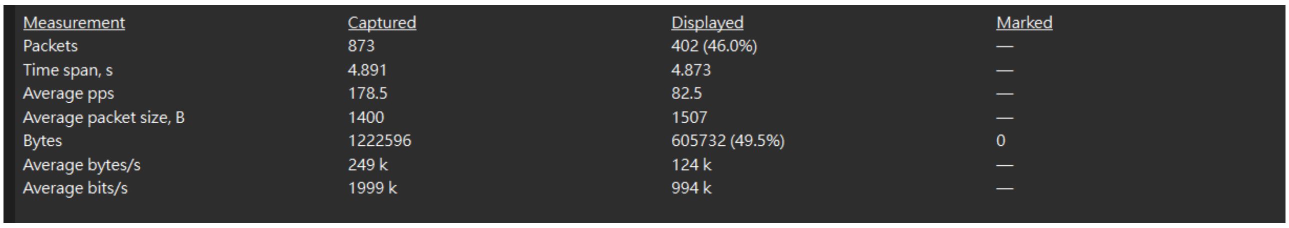
ANS：yes.

Result of computeRate.py



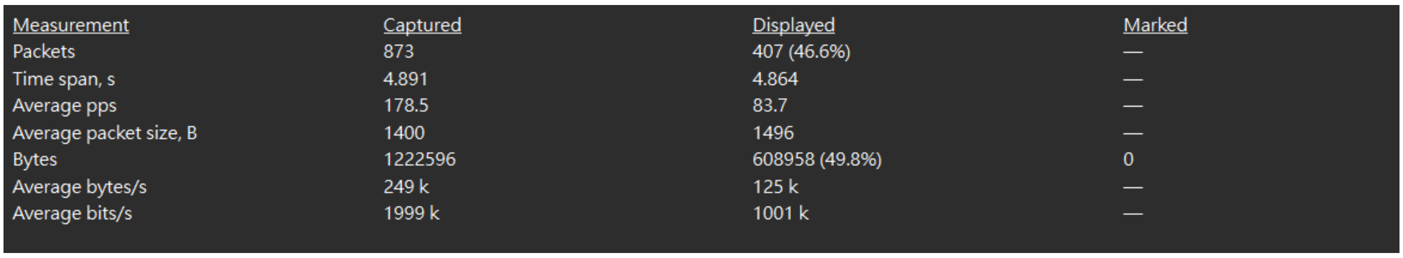
```
---TCP---
Flow1(h1->h3):0.9493184 Mbps
Flow2(h1->h3):0.9541632 Mbps
Flow3(h2->h4):1.8987424 Mbps

---UDP---
Flow4(h1->h3):1.0620288 Mbps
Flow5(h1->h3):1.0620288 Mbps
Flow6(h2->h4):1.0620288 Mbps
```

TCP flow1：



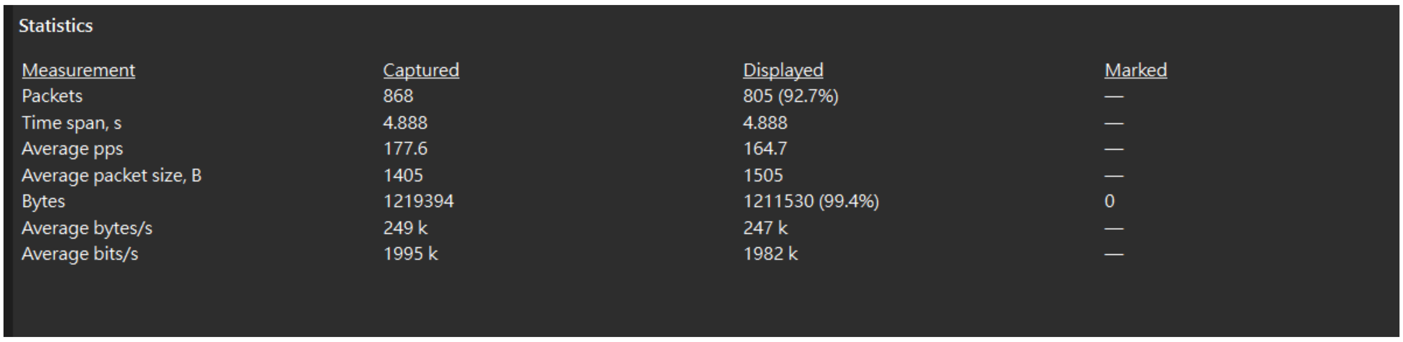
Measurement	Captured	Displayed	Marked
Packets	873	402 (46.0%)	—
Time span, s	4.891	4.873	—
Average pps	178.5	82.5	—
Average packet size, B	1400	1507	—
Bytes	1222596	605732 (49.5%)	0
Average bytes/s	249 k	124 k	—
Average bits/s	1999 k	994 k	—

TCP flow2：



Measurement	Captured	Displayed	Marked
Packets	873	407 (46.6%)	—
Time span, s	4.891	4.864	—
Average pps	178.5	83.7	—
Average packet size, B	1400	1496	—
Bytes	1222596	608958 (49.8%)	0
Average bytes/s	249 k	125 k	—
Average bits/s	1999 k	1001 k	—

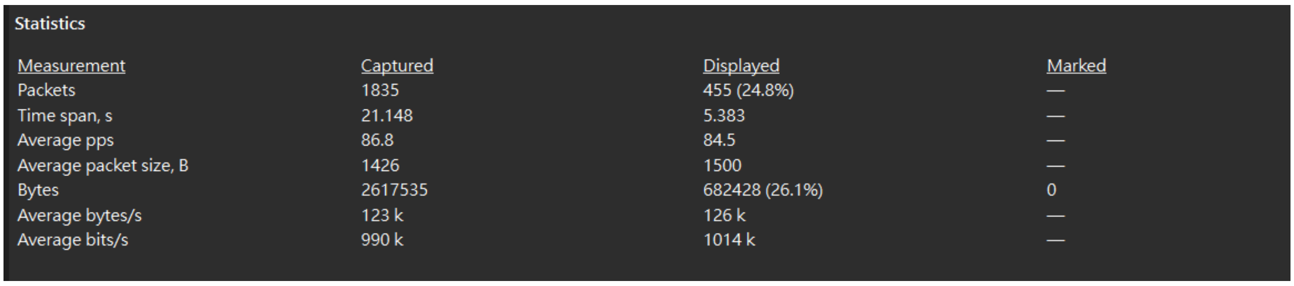
TCP flow3：



Statistics

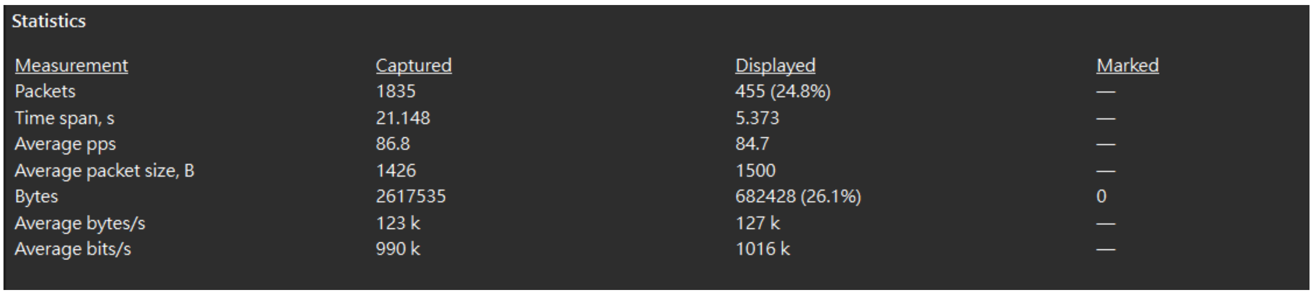
Measurement	Captured	Displayed	Marked
Packets	868	805 (92.7%)	—
Time span, s	4.888	4.888	—
Average pps	177.6	164.7	—
Average packet size, B	1405	1505	—
Bytes	1219394	1211530 (99.4%)	0
Average bytes/s	249 k	247 k	—
Average bits/s	1995 k	1982 k	—

UDP flow1：


Statistics

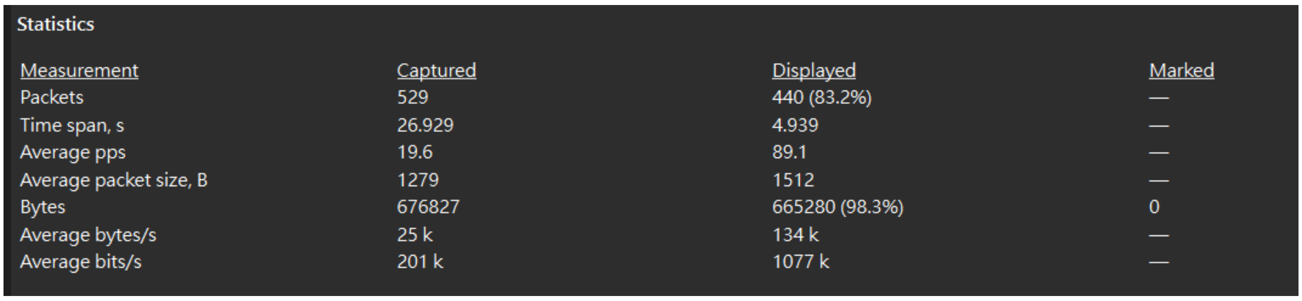
Measurement	Captured	Displayed	Marked
Packets	1835	455 (24.8%)	—
Time span, s	21.148	5.383	—
Average pps	86.8	84.5	—
Average packet size, B	1426	1500	—
Bytes	2617535	682428 (26.1%)	0
Average bytes/s	123 k	126 k	—
Average bits/s	990 k	1014 k	—

UDP flow2：


Statistics

Measurement	Captured	Displayed	Marked
Packets	1835	455 (24.8%)	—
Time span, s	21.148	5.373	—
Average pps	86.8	84.7	—
Average packet size, B	1426	1500	—
Bytes	2617535	682428 (26.1%)	0
Average bytes/s	123 k	127 k	—
Average bits/s	990 k	1016 k	—

UDP flow3：

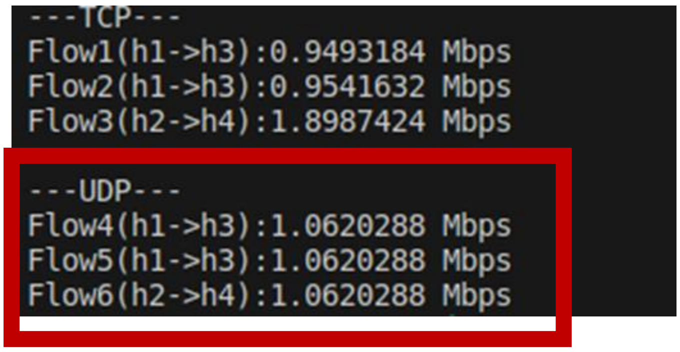

Statistics

Measurement	Captured	Displayed	Marked
Packets	529	440 (83.2%)	—
Time span, s	26.929	4.939	—
Average pps	19.6	89.1	—
Average packet size, B	1279	1512	—
Bytes	676827	665280 (98.3%)	0
Average bytes/s	25 k	134 k	—
Average bits/s	201 k	1077 k	—

- Do you observe the same throughput from TCP and UDP?

ANS：

I observe the same throughput from UDP.UDP has Flow1, Flow2, and Flow3 have the same throughput.


```
---TCP---
Flow1(h1->h3):0.9493184 Mbps
Flow2(h1->h3):0.9541632 Mbps
Flow3(h2->h4):1.8987424 Mbps

---UDP---
Flow4(h1->h3):1.0620288 Mbps
Flow5(h1->h3):1.0620288 Mbps
Flow6(h2->h4):1.0620288 Mbps
```

- What have you learned from this lab?

ANS：

The most impressive part I learned in this lab is “Throughput Computation” and “Packet Analysis”.In throughput computation part, I spent a lot of time to let my idea to implement in python code; In packet analysis part, I understand how packets transmission work and comprehend how to filter the packets you want to analyze.

- What difficulty have you met in this lab?

ANS：

I got into touble to figure out the computeRate.py.Because I still not accustomed to python. Further more, I have to understand the logic behind the transmission in this lab and to refer to the parser.py to create the computeRate.py to compute the throughput.