• Describe each step and how to run your program

topo_TCP.py:

step 0
Need to import “**mininet**” package and run the program in **Python 2** environment.

step 1
Build **hosts**(h1, h2, h3 ,h4) and **switches**(s1, s2, s3).
Also, add **links** between hosts and switches.

```
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)
```

step 2
Create the network.

```
    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")
    h3 = net.get("h3")
    h4 = net.get("h4")
```

step 3

Use “**tcpdump**” to record trace in h3 and h4 and output to “**.pcap**” files.
Then, use “**iperf**” to generate 3 flows. Respectively,
flow 1 which come from h1 and go to h3 using port 7777,
flow 2 which come from h1 and go to h3 using port 7776,
and **flow 3 which come from h2 and go to h4 using port 7775**.
Each flow use **TCP** protocol and take **5 seconds** to transmit.
The transfer data will be outputted in the “.txt” files.
We can get all output files after 5 seconds.

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

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

# TCP flow
h3.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3_1.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h1_1.txt &")
h3.cmd("iperf -s -i 1 -t 5 -p 7776 > ../out/TCP_s_h3_2.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7776 > ../out/TCP_c_h1_2.txt &")
h4.cmd("iperf -s -i 1 -t 5 -p 7775 > ../out/TCP_s_h4.txt &")
h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7775 > ../out/TCP_c_h2.txt &")

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

topo_UDP.py:

Step 0 to step 2 are the same as the **topo_TCP.py.**
Only one difference in step 3 is add "**-u**" argument in each "iperf" command, because we want to use **UDP** protocol instead of TCP protocol for each generated flow.

```
# UDP flow
h3.cmd("iperf -s -i 1 -t 5 -p 7777 -u > ../out/UDP_s_h3_1.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 -u > ../out/UDP_c_h1_1.txt &")
h3.cmd("iperf -s -i 1 -t 5 -p 7776 -u > ../out/UDP_s_h3_2.txt &")
h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7776 -u > ../out/UDP_c_h1_2.txt &")
h4.cmd("iperf -s -i 1 -t 5 -p 7775 -u > ../out/UDP_s_h4.txt &")
h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7775 -su > ../out/UDP_c_h2.txt &")
```

computeRate.py:

step 0
Need the "**TCP_h3.pcap**", "**TCP_h4.pcap**", "**UDP_h3.pcap**" and "**UDP_h3.pcap**", generated by "**topo_TCP.py**" and **"topo_UDP.py**".
Also need to import "**scapy**" package.
The program can be ran in **Python 3**.

step1
Read ".pcap" files by **rdpcap()**

```
# read pcap
packets_h3_T = rdpcap("../out/TCP_h3.pcap")
packets_h4_T = rdpcap("../out/TCP_h4.pcap")
packets_h3_U = rdpcap("../out/UDP_h3.pcap")
packets_h4_U = rdpcap("../out/UDP_h4.pcap")
```

step2

To get throughputs, I wrote a function called “**getThroughput**”.
Two arguments, “**protocol**” and “**port**”, can select the data from the information provided by rdpcap(). (For example, we can clearly split the data of flow 1 and data of flow 2 from “TCP_h3.pcap” , because the two flows used **different port**)
After the loop that calculate total data, **we need to multiply by 8 to convert into bits and divide by 10^6 to convert into Mb**. Then, we can simply **divide by 5 to get throughput, since all these flows were set to transfer for 5 seconds.**
Finally, we can print the throughput.

```
def getThroughput(packets, protocol, port):
    total = 0
    count = 0
    if protocol == 'TCP':
        for packet in packets[TCP]:
            count += 1
        for i in range(count):
            if packets[TCP][i][2].dport == port:
                total += len(packets[TCP][i])
    else:
        for packet in packets[UDP]:
            count += 1
        for i in range(count):
            if packets[UDP][i][2].dport == port:
                total += len(packets[UDP][i])
    Mbps = (total * 8 / 1000000) / 5
    return Mbps
```

```
print("\n --- TCP --- ")
print("Flow1(h1->h3):              {} Mbps".format(getThroughput(packets_h3_T, "TCP", 7777)))
print("Flow2(h1->h3):              {} Mbps".format(getThroughput(packets_h3_T, "TCP", 7776)))
print("Flow3(h2->h4):              {} Mbps".format(getThroughput(packets_h4_T, "TCP", 7775)))
print("")
print(" --- UDP --- ")
print("Flow1(h1->h3):              {} Mbps".format(getThroughput(packets_h3_U, "UDP", 7777)))
print("Flow2(h1->h3):              {} Mbps".format(getThroughput(packets_h3_U, "UDP", 7776)))
print("Flow3(h2->h4):              {} Mbps\n".format(getThroughput(packets_h4_U, "UDP", 7775)))
```

•Describe your observations from the results in this lab

In TCP cases, I found that flow 1 and flow 2 shared the bandwidth(2 Mb) while flow 3 could use almost all bandwidth.
On the other hand, the throughputs of each UDP flow were similar. The total throughput of flow 1 and flow 2 is higher than 2 Mbps and the throughput of flow 3 is almost half of the bandwidth.
For the result of UDP cases, I believe it is because the UDP protocol is connectionless or “best-effort delivery”. Every UDP flow tried its best to transfer data, however, only about 1 Mbps of data could be successfully delivered.

•Answer the following question in short:

•What does each iPerfcommand you used mean?

```
    # TCP flow
    h3.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3_1.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h1_1.txt
&")
    h3.cmd("iperf -s -i 1 -t 5 -p 7776 > ../out/TCP_s_h3_2.txt &")
    h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7776 > ../out/TCP_c_h1_2.txt
&")
    h4.cmd("iperf -s -i 1 -t 5 -p 7775 > ../out/TCP_s_h4.txt &")
    h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7775 > ../out/TCP_c_h2.txt
&")
```

1. iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3_1.txt &
Running iperf in server mode, generate flow using TCP protocol and port 7777. Also, setting the time in 5 seconds to transmit for and the interval time in 1 seconds between periodic bandwidth, jitter, and loss reports. Finally, save the records in “../out/TCP_s_h3_1.txt”.

2. iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 > ../out/TCP_c_h1_1.txt &
Running iperf in client mode and setting destination to ip of h3, generate flow using TCP protocol and port 7776. Also, setting the time in 5 seconds to transmit for and the interval time in 1 seconds between periodic bandwidth, jitter, and loss reports. Finally, save the records in “../out/TCP_s_h1_1.txt”.

3. iperf -s -i 1 -t 5 -p 7776 > ../out/TCP_s_h3_2.txt &
Everything is the same as (1) but port change to 7776 and output path to “../out/TCP_s_h3_2.txt”.

4. iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7776 > ../out/TCP_c_h1_2.txt &
Everything is the same as (2) but port change to 7776 and output path to “../out/TCP_s_h1_2.txt”.

5. iperf -s -i 1 -t 5 -p 7775 > ../out/TCP_s_h4.txt &
Everything is the same as (1) but port change to 7775 and output path to "../out/TCP_s_h4.txt".

6. iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7775 -su > ../out/UDP_c_h2.txt &
Everything is the same as (2) but port change to 7775 and output path to "../out/TCP_s_h2.txt".

```
   # UDP flow
   h3.cmd("iperf -s -i 1 -t 5 -p 7777 -u > ../out/UDP_s_h3_1.txt &")
   h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7777 -u >
../out/UDP_c_h1_1.txt &")
   h3.cmd("iperf -s -i 1 -t 5 -p 7776 -u > ../out/UDP_s_h3_2.txt &")
   h1.cmd("iperf -c " + str(h3.IP()) + " -i 1 -t 5 -p 7776 -u >
../out/UDP_c_h1_2.txt &")
   h4.cmd("iperf -s -i 1 -t 5 -p 7775 -u > ../out/UDP_s_h4.txt &")
   h2.cmd("iperf -c " + str(h4.IP()) + " -i 1 -t 5 -p 7775 -su >
../out/UDP_c_h2.txt &")
```

All six commands are identical to the six commands above but the UDP version.

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

tcp.port == 7777 for TCP flow 1
tcp.port == 7776 for TCP flow 2
tcp.port == 7775 for TCP flow 3
udp.port == 7777 for UDP flow 1
udp.port == 7776 for UDP flow 2
udp.port == 7775 for UDP flow 3

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

result of computeRate.py

```
 --- TCP ---
Flow1(h1->h3):              0.9898783999999999 Mbps
Flow2(h1->h3):              0.994512 Mbps
Flow3(h2->h4):              1.9651039999999997 Mbps

 --- UDP ---
Flow1(h1->h3):              1.0717056 Mbps
Flow2(h1->h3):              1.0789632 Mbps
Flow3(h2->h4):              1.0813824 Mbps
```

TCP flow 1

Statistics

Measurement	Captured	Displayed	Marked
Packets	941	423 (45.0%)	—
Time span, s	52.464	5.110	—
Average pps	17.9	82.8	—
Average packet size, B	1359	1493	—
Bytes	1278974	631738 (49.4%)	0
Average bytes/s	24 k	123 k	—
Average bits/s	195 k	989 k	—

TCP flow 2

Statistics

Measurement	Captured	Displayed	Marked
Packets	941	424 (45.1%)	—
Time span, s	52.464	5.098	—
Average pps	17.9	83.2	—
Average packet size, B	1359	1497	—
Bytes	1278974	634688 (49.6%)	0
Average bytes/s	24 k	124 k	—
Average bits/s	195 k	995 k	—

TCP flow 3

Statistics

Measurement	Captured	Displayed	Marked
Packets	927	833 (89.9%)	—
Time span, s	54.513	5.059	—
Average pps	17.0	164.7	—
Average packet size, B	1366	1505	—
Bytes	1266442	1253914 (99.0%)	0
Average bytes/s	23 k	247 k	—
Average bits/s	185 k	1982 k	—

UDP flow 1

Statistics

Measurement	Captured	Displayed	Marked
Packets	1021	461 (45.2%)	—
Time span, s	68.635	5.465	—
Average pps	14.9	84.4	—
Average packet size, B	1366	1496	—
Bytes	1394853	689656 (49.4%)	0
Average bytes/s	20 k	126 k	—
Average bits/s	162 k	1009 k	—

UDP flow 2

Statistics

Measurement	Captured	Displayed	Marked
Packets	1021	460 (45.1%)	—
Time span, s	68.635	5.463	—
Average pps	14.9	84.2	—
Average packet size, B	1366	1504	—
Bytes	1394853	691832 (49.6%)	0
Average bytes/s	20 k	126 k	—
Average bits/s	162 k	1013 k	—

UDP flow 3

Statistics

Measurement	Captured	Displayed	Marked
Packets	550	448 (81.5%)	—
Time span, s	72.743	4.987	—
Average pps	7.6	89.8	—
Average packet size, B	1256	1512	—
Bytes	690825	677376 (98.1%)	0
Average bytes/s	9496	135 k	—
Average bits/s	75 k	1086 k	—

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

For TCP cases, Yes, the total throughput of flow 1 and flow 2 and the throughput of flow both close but not excess 2 Mbps.
For UDP cases, No, the throughputs of each flow are all about only half of the bottleneck throughput.

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

No, TCP can approximately speed up to the limit, while UDP is stable at approximately 1 Mbps even if the bandwidth is 2 Mbps.

Bonus:

•What have you learned from this lab?

I learned how to use mininet to build a SDN and use Wireshark to analyze the ".pcap" files.
After this lab, I obtained the ability to create and test a network which I believe is important to web development.

•What difficulty have you met in this lab?

In the beginning, I didn't know how to design computeRate.py because I had no idea how to distinguish between flow 1 and flow 2 in TCP/UDP_h3.pcap.
The iperf commands are synchronized, so we can't use the timestamp to deal with the problem.
Finally, I came up with the idea that I could design different flows using different ports, thus I could make use of this property to differentiate all flows!