

NYCU-CN2023/Lab1-Liao-Hsiu-I

Describe each step and how to run your program

Task 2. Create a Topology

In the **topo.py** (<http://topo.py>) file, I learned how to build a network topology.

```
1      # Add hosts to a topology
2      self.addHost("h1")
3      self.addHost("h2")
4
5      # Add switches to a topology
6      self.addSwitch("s1")
7      self.addSwitch("s2")
8
9      # Add bidirectional links to a topology, and set bandwidth(Mbps)
10     self.addLink("h1", "s1", bw=2)
11     self.addLink("s1", "s2", bw=2)
12     self.addLink("s2", "h3", bw=2)
```

Therefore, I added new hosts, switches, and linkers.

```
1      def build(self):
2          # Add hosts to a topology
3          self.addHost("h1")
4          self.addHost("h2")
5          self.addHost("h3")
6          self.addHost("h4")
7
8          # Add switches to a topology
9          self.addSwitch("s1")
10         self.addSwitch("s2")
11         self.addSwitch("s3")
12
13         # Add bidirectional links to a topology, and set bandwidth(Mbps)
14         self.addLink("h1", "s1", bw=2)
15         self.addLink("h2", "s1", bw=2)
16         self.addLink("s1", "s2", bw=2)
17         self.addLink("s1", "s3", bw=2)
18         self.addLink("s2", "h3", bw=2)
19         self.addLink("s3", "h4", bw=2)
```

Task 3. Generate Flows via iPerf

topo_TCP

In the **topo.py** (<http://topo.py>) file, I learned how to establish a connection from h1 to h2 with a destination port of 7777.

```

1      # Use tcpdump to record packet in background
2      h2.cmd("tcpdump -w ../out/h2_output.pcap &")
3
4      # Create flow via iperf
5      print("create flow via iperf")
6
7      # TCP flow
8      h2.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/result_s.txt &")
9      h1.cmd("iperf -c " + str(h2.IP()) + " -i 1 -t 5 -p 7777 > ../out/result_

```

Therefore, I changed the destination port name, as well as the names of the result_s.txt and result_c.txt files, to generate two TCP flows from h1 to h3 and one TCP flow from h2 to h4. (This was advised by ChatGPT and was discovered while I was writing the Python code.

```

1      h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')
2      h3.cmd('iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3_1.txt &')
3      h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7777 > ../out/TCP_
4
5      h3.cmd('iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_2.txt &')
6      h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7778 > ../out/TCP_
7
8      h4.cmd('tcpdump -w ../out/TCP_h4.pcap &')
9      h4.cmd('iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h4.txt &')
10     h2.cmd('iperf -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7779 > ../out/TCP_

```

The following statement is inaccurate. When I call 'h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')' for the second time, it will terminate the previous instance of 'h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')'.

```

1      h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')
2      h3.cmd('iperf -s -i 1 -t 5 -p 7777 > ../out/TCP_s_h3_1.txt &')
3      h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7777 > ../out/TCP_
4
5      h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')
6      h3.cmd('iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_2.txt &')
7      h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7778 > ../out/TCP_
8
9      h4.cmd('tcpdump -w ../out/TCP_h4.pcap &')
10     h4.cmd('iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h4.txt &')
11     h2.cmd('iperf -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7779 > ../out/TCP_

```

topo_UDP

I understand that the difference between topo_UDP and topo_TCP lies in the 'type' of connections; therefore, I only made changes based on the information provided in 'iPerf' on page P.12 in lab1_2.pdf and the port numbers.

I used 'UDP' connections instead of 'TCP' connections.

Therefore, I added '-u' to the code, and the other codes are the same.

```

1 h3.cmd('tcpdump -w ../out/UDP_h3.pcap &')
2 h3.cmd('iperf -u -s -i 1 -t 5 -p 7787 > ../out/UDP_s_h3_1.txt &')
3 h1.cmd('iperf -u -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7787 > ../out,
4
5 h3.cmd('iperf -u -s -i 1 -t 5 -p 7788 > ../out/UDP_s_h3_2.txt &')
6 h1.cmd('iperf -u -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7788 > ../out,
7
8 h4.cmd('tcpdump -w ../out/UDP_h4.pcap &')
9 h4.cmd('iperf -u -s -i 1 -t 5 -p 7789 > ../out/UDP_s_h4.txt &')
10 h2.cmd('iperf -u -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7789 > ../out,

```

Task 4. Compute the Throughput

I used the following algorithms to accomplish Task 3.

1. Read the file
2. Read the pcap file
3. Count the total_transmit_bits for packets with the desired dst_port
4. Calculate throughput

I would provide more details on how to obtain this information.

In **parser.py** (<http://parser.py>), it explains how to retrieve the dst_port and packet_size of packet[0].

```

1 print ("***Get data of this packet***")
2 print (f"      src IP: {packets[TCP][0][1].src}")      # in IP layer
3 print (f"      dst IP: {packets[TCP][0][1].dst}")      # in IP layer
4 print (f"      src port: {packets[TCP][0][2].sport}")  # in TCP layer
5 print (f"      dst port: {packets[TCP][0][2].dport}")  # in TCP layer
6 print (f"packet size: {len(packets[TCP][0])} bytes")

```

So, I applied the same concept, but I didn't specify [0] since I scanned all possible cases rather than a specific packet. For example, let's take a look at how to calculate the throughput of Flow1(h1->h3).

```

1  # read the file
2  INPUTPATH = '/home/cn2023-lab1/Desktop/out/TCP_h3.pcap'
3
4  # Read the pcap file
5  packets = rdpcap(INPUTPATH)
6
7  # Initialize the variable to store total transmit bits for port 7777
8  total_transmit_bits = 0
9
10 # Iterate through all TCP packets
11 for packet in packets[TCP]:
12     # Get the source and destination ports
13     src_port = packet[2].sport
14     dst_port = packet[2].dport
15
16     # Check if the source port is 7777
17     if dst_port == 7777:
18         # Calculate the transmit bits for the current packet
19         transmit_bits = len(packet)
20
21         # Accumulate the transmit bits to the total
22         total_transmit_bits += transmit_bits
23
24 # Output the total transmit bits for port 7777
25 print(f"Flow1(h1->h3): {total_transmit_bits * 8 / 5000000} Mbps")

```

Similarly, I changed the file name and destination port to calculate the throughput of the other.

Describe your observations from the results in this lab

```

cn2023-lab1@cn2023lab1-VirtualBox:~/Desktop/Lab1-Liao-Hsiu-I/src$ python3 computeRate.py
--- TCP ---
Flow1(h1->h3): 0.9994624 Mbps
Flow2(h1->h3): 0.9993568 Mbps
Flow3(h2->h4): 1.9754208 Mbps
--- UDP ---
Flow1(h1->h3): 1.0813824 Mbps
Flow2(h1->h3): 1.0838016 Mbps
Flow3(h2->h4): 1.0813824 Mbps

```

1. Two TCP connections share the bandwidth.
2. Two UDP connections do not share the bandwidth.
3. Any throughput is limited by the bottleneck.

What does each iPerf command you used mean?

iPerf Command line options

- -s: (Server) Run iPerf in server mode
- -c: (Client) Run iPerf in client mode, connecting to an iPerf server running on host
- -i: (Interval) Sets the interval time in seconds between periodic bandwidth, jitter, and loss reports
- -t: (Time) The time in seconds to transmit for
- -p: (Port) The server port for the server to listen on and the client to connect to
- -u: (UDP) Use UDP.
- -b: (bandwidth) Set target bandwidth to n bits/sec (default 1 Mbit/sec for UDP, unlimited for TCP)

Let't take a look at tcp_flow1.

```

h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')
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_1.1

```

- tcpdump: A command-line tool used for monitoring, analyzing, and capturing network packets.
- -w ../out/TCP_h3.pcap: Specifies the file path and name where the captured packet information will be written. In this case, it is the file TCP_h3.pcap located in the out folder one level above the current directory.
- &: Appending this symbol at the end of the command instructs it to run in the background, allowing the terminal to continue executing other commands without being blocked.
- iperf: A tool for measuring network bandwidth, commonly used for server-client testing.
- -s: Indicates server mode.
- -i 1: Specifies to display updates every second.
- -t 5: Specifies the test duration as 5 seconds.
- -p 7777: Specifies the port on which the server listens (port 7777).
- > ../out/TCP_s_h3_1.txt: Redirects the output of iperf to a file named TCP_s_h3_1.txt, located in the out folder one level above the current directory.
- -c ' + str(h3.IP()) + ': Specifies the client to connect to the IP address of host h3.
- > ../out/TCP_c_h1_1.txt: Redirects the output of iperf to a file named TCP_c_h1_1.txt, located in the out folder one level above the current directory.

Let's take a look at udp_flow1.

```
h4.cmd('tcpdump -w ../out/UDP_h4.pcap &')
h4.cmd('iperf -u -s -i 1 -t 5 -p 7789 > ../out/UDP_s_h4.txt &')
h2.cmd('iperf -u -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7789 > ../out/UDP_c_h2.
```

- iperf -u: Indicates UDP mode.

What is your command to filter each flow in Wireshark?

- TCP flow1(h1->h3): ip.src == 10.0.0.1 and tcp.port == 7787
- TCP flow2(h1->h3): ip.src == 10.0.0.1 and tcp.port == 7788

- TCP flow3(h2->h4): ip.src == 10.0.0.2 and tcp.port == 7789
- UDP flow1(h1->h3): ip.src == 10.0.0.1 and udp.port == 7787
- UDP flow2(h1->h3): ip.src == 10.0.0.1 and udp.port == 7788
- UDP flow3(h2->h4): ip.src == 10.0.0.2 and udp.port == 7789

Show the results of computeRate.py and statistics of Wireshark

computeRateResult

```

cn2023-lab1@cn2023lab1-VirtualBox:~/Desktop/Lab1-Liao-Hsiu-I/src$ python3 computeRate.py
--- TCP ---
Flow1(h1->h3): 0.9994624 Mbps
Flow2(h1->h3): 0.9993568 Mbps
Flow3(h2->h4): 1.9754208 Mbps

--- UDP ---
Flow1(h1->h3): 1.0813824 Mbps
Flow2(h1->h3): 1.0838016 Mbps
Flow3(h2->h4): 1.0813824 Mbps

```

TCP flow1(h1->h3)

The screenshot shows the Wireshark interface with a packet capture filter set to `ip.src == 10.0.0.1 and tcp.port == 7777`. The packet list shows several packets from source 10.0.0.1 to destination 10.0.0.1. The packet details pane shows the selected packet (Frame 6) with details: Ethernet II, Src: a6:bc:18:8b:75:82, Internet Protocol Version 4, Src: 10.0.0.1, and Transmission Control Protocol, Src Port: 7777. The packet statistics pane shows the following statistics:

Measurement	Captured	Displayed	Marked
Packets	887	228 (25.7%)	—
Time span, s	7.891	5.116	—
Average pps	112.4	44.6	—
Average packet size, B	1442	2740	—
Bytes	1279439	624664 (48.8%)	0
Average bytes/s	162 k	122 k	—
Average bits/s	1297 k	976 k	—

The packet capture file properties pane shows the following details:

- Details:
 - File (SHA1): 71a0b9e01a10372010b0c0120031e1a0e0201
 - Format: Wireshark/tcpdump/... - pcap
 - Encapsulation: Ethernet
 - Snapshot length: 262144
- Time:
 - First packet: 2023-11-14 23:14:24
 - Last packet: 2023-11-14 23:14:32
 - Elapsed: 00:00:07
- Capture:
 - Hardware: Unknown
 - OS: Unknown
 - Application: Unknown
- Interfaces:

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Unknown	Unknown	Unknown	Ethernet	262144 bytes
- Statistics:

Measurement	Captured	Displayed	Marked
Packets	887	228 (25.7%)	—
Time span, s	7.891	5.116	—
Average pps	112.4	44.6	—
Average packet size, B	1442	2740	—
Bytes	1279439	624664 (48.8%)	0
Average bytes/s	162 k	122 k	—
Average bits/s	1297 k	976 k	—

TCP flow2(h1->h3)

Wireshark · Capture File Properties · TCP_h3.pcap

Details

Format: Wireshark/tcpdump/... - pcap

Encapsulation: Ethernet

Snapshot length: 262144

Time

First packet: 2023-11-14 23:14:24

Last packet: 2023-11-14 23:14:32

Elapsed: 00:00:07

Capture

Hardware: Unknown

OS: Unknown

Application: Unknown

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Unknown	Unknown	Unknown	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	887	227 (25.6%)	—
Time span, s	7.891	5.127	—
Average pps	112.4	44.3	—
Average packet size, B	1442	2752	—
Bytes	1279439	624598 (48.8%)	0
Average bytes/s	162 k	121 k	—
Average bits/s	1297 k	974 k	—

Capture file comments

Help Refresh Copy To Clipboard Close Save Comments

ip.src == 10.0.0.1 and tcp.port == 7778

No.	Time	Source
8	0.126571	10.0.0.1
20	0.189365	10.0.0.1
21	0.189370	10.0.0.1
23	0.195471	10.0.0.1
25	0.201443	10.0.0.1
31	0.237234	10.0.0.1
33	0.250309	10.0.0.1
39	0.285695	10.0.0.1
42	0.298757	10.0.0.1
48	0.335534	10.0.0.1
50	0.347672	10.0.0.1
56	0.383833	10.0.0.1
58	0.395957	10.0.0.1
66	0.438213	10.0.0.1
68	0.444421	10.0.0.1
70	0.456398	10.0.0.1
77	0.492997	10.0.0.1

> Frame 8: 74 bytes on wire (592 bits), 74 captured (592 bits) on 0
> Ethernet II, Src: a6:bc:18:8b:75:82 (a6:bc:18:8b:75:82), Dst: 02:00:00:00:00:00, Protocol: Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
> Transmission Control Protocol, Src Port: 7778, Dst Port: 7778

TCP flow3(h2->h4)

Wireshark · Capture File Properties · TCP_h4.pcap

Details

Format: Wireshark/tcpdump/... - pcap

Encapsulation: Ethernet

Snapshot length: 262144

Time

First packet: 2023-11-14 23:14:24

Last packet: 2023-11-14 23:14:32

Elapsed: 00:00:07

Capture

Hardware: Unknown

OS: Unknown

Application: Unknown

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Unknown	Unknown	Unknown	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	876	431 (49.2%)	—
Time span, s	7.879	5.131	—
Average pps	111.2	84.0	—
Average packet size, B	1445	2865	—
Bytes	1266188	1234638 (97.5%)	0
Average bytes/s	160 k	240 k	—
Average bits/s	1285 k	1925 k	—

Capture file comments

Help Refresh Copy To Clipboard Close Save Comments

ip.src == 10.0.0.2 and tcp.port == 7779

No.	Time	Source
4	0.018424	10.0.0.2
8	0.104427	10.0.0.2
9	0.108078	10.0.0.2
11	0.109823	10.0.0.2
13	0.114748	10.0.0.2
15	0.126434	10.0.0.2
17	0.138196	10.0.0.2
19	0.150909	10.0.0.2
21	0.163319	10.0.0.2
22	0.163326	10.0.0.2
24	0.175554	10.0.0.2
26	0.187640	10.0.0.2
29	0.199561	10.0.0.2
31	0.212511	10.0.0.2
33	0.224458	10.0.0.2
36	0.237472	10.0.0.2
38	0.243546	10.0.0.2

> Frame 4: 74 bytes on wire (592 bits), 74 captured (592 bits) on 0
> Ethernet II, Src: ae:0b:6c:ef:e0:38 (ae:0b:6c:ef:e0:38), Dst: 02:00:00:00:00:00, Protocol: Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
> Transmission Control Protocol, Src Port: 7779, Dst Port: 7779

<https://hackmd.io/KmyQrVmNRCEzmD5AMsvYA?view>

7/23

UDP flow1(h1->h3)

Wireshark · Capture File Properties · UDP_h3.pcap

Details

Format:Wireshark/tcpdump/... - pcap

Encapsulation:Ethernet

Snapshot length:262144

Time

First packet:2023-11-14 23:19:11

Last packet:2023-11-14 23:19:23

Elapsed:00:00:11

Capture

Hardware:Unknown

OS:Unknown

Application:Unknown

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Unknown	Unknown	Unknown	Ethernet	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	959	454 (47.3%)	—
Time span, s	11.715	5.439	—
Average pps	81.9	83.5	—
Average packet size, B	1454	1498	—
Bytes	1394058	679994 (48.8%)	0
Average bytes/s	119 k	125 k	—
Average bits/s	952 k	1000 k	—

Capture file comments

Help

Refresh

Copy To Clipboard

Close

Save Comments

ip.src == 10.0.0.1 and udp.port == 7787

No.	Time	Source
4	0.073628	10.0.0.1
9	0.099803	10.0.0.1
10	0.105794	10.0.0.1
12	0.117632	10.0.0.1
13	0.123721	10.0.0.1
14	0.129561	10.0.0.1
16	0.141539	10.0.0.1
18	0.153403	10.0.0.1
20	0.166457	10.0.0.1
22	0.178382	10.0.0.1
24	0.190389	10.0.0.1
26	0.202273	10.0.0.1
28	0.214189	10.0.0.1
31	0.227348	10.0.0.1
33	0.239353	10.0.0.1
35	0.251365	10.0.0.1
37	0.263147	10.0.0.1

> Frame 4: 1512 bytes on wire (12096 bytes captured) on interface 0

> Ethernet II, Src: 36:4e:ac:2f:fe:64, Dst: 02:00:00:00:00:00

> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.1

> User Datagram Protocol, Src Port: 5555, Dst Port: 7787

> Data (1470 bytes)

UDP_h3.pcap

UDP flow2(h1->h3)

Wireshark · Capture File Properties · UDP_h3.pcap

Details

File

Name:/Users/origamye/Deskto/lab1-Liao-Hsiu-I/out/UDP_h3.pcap

Length:1409 kB

Hash (SHA256):2b7ebf5a8d03c684a661f2f90d12afd99a8589b0c41bb45dddec2c744414a10ef

Hash (RIPEMD160):3d402b88165ccfe0cb8054fd7276800b13352206

Hash (SHA1):f1a55977d571f7ac923688f7cf34c4b965863328

Format:Wireshark/tcpdump/... - pcap

Encapsulation:Ethernet

Snapshot length:262144

Time

First packet:2023-11-14 23:19:11

Last packet:2023-11-14 23:19:23

Elapsed:00:00:11

Capture

Hardware:Unknown

OS:Unknown

Application:Unknown

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
Unknown	Unknown	Unknown	Ethernet	262144 bytes

Statistics

Measurement	Captured	Dislaved	Marked
-------------	----------	----------	--------

Capture file comments

Help

Refresh

Copy To Clipboard

Close

Save Comments

ip.src == 10.0.0.1 and udp.port == 7788

No.	Time	Source
5	0.075846	10.0.0.1
6	0.087156	10.0.0.1
7	0.091374	10.0.0.1
8	0.097293	10.0.0.1
11	0.111717	10.0.0.1
15	0.135553	10.0.0.1
17	0.147464	10.0.0.1
19	0.160439	10.0.0.1
21	0.172482	10.0.0.1
23	0.184466	10.0.0.1
25	0.196328	10.0.0.1
27	0.208256	10.0.0.1
30	0.221403	10.0.0.1
32	0.233316	10.0.0.1
34	0.245250	10.0.0.1
36	0.257201	10.0.0.1
38	0.268977	10.0.0.1

> Frame 5: 1512 bytes on wire (12096 bytes captured) on interface 0

> Ethernet II, Src: 36:4e:ac:2f:fe:64, Dst: 02:00:00:00:00:00

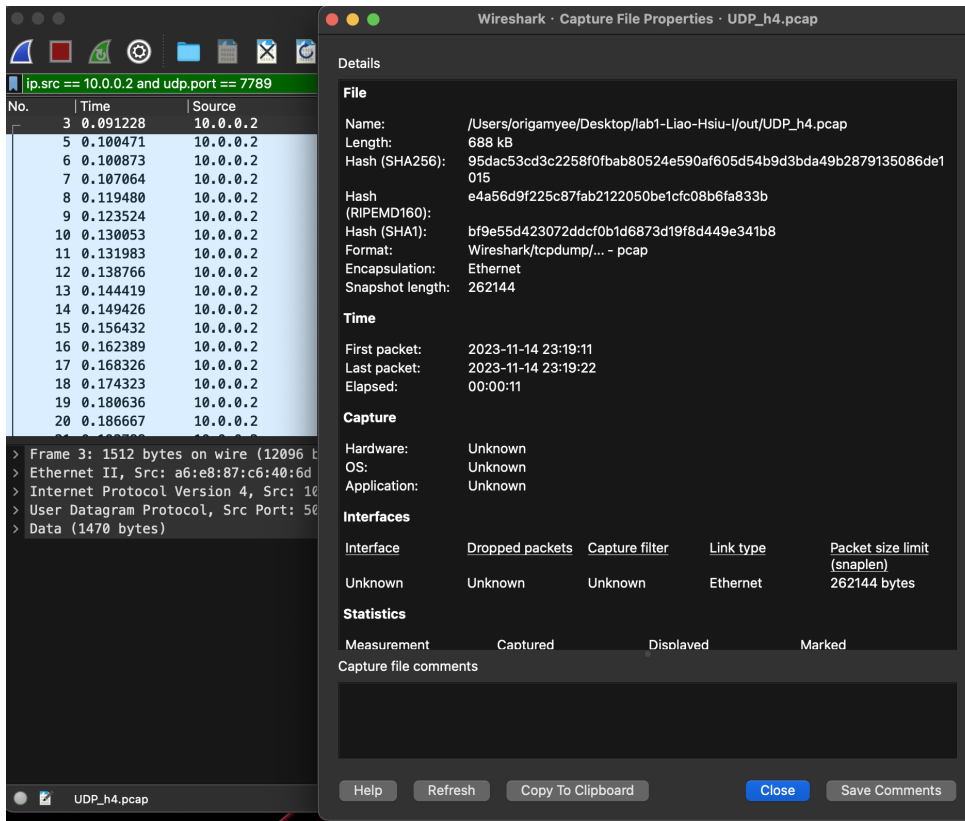
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.1

> User Datagram Protocol, Src Port: 3456, Dst Port: 7788

> Data (1470 bytes)

UDP_h3.pcap

UDP flow3(h2->h4)



Does the throughput match the bottleneck throughput of the path?

- Yes, bottleneck throughput of the path = 2Mbps, all of them are smaller than 2Mbps.

Do you observe the same throughput from TCP and UDP?

- No, as discovered previously:
 1. Two TCP connections share the bandwidth.
 2. Two UDP connections do not share the bandwidth.
 3. Additionally, the UDP throughput is slightly higher than the combined throughput of the shared connections.
 4. UDP has bigger packet size.

Bonus

What have you learned from this lab?

I learned

- How to Use an OVA File on UTM

- It aligns with the description provided in the "What difficulties have you encountered in this lab?" section.
- how to use ChatGPT to explain concepts I haven't learned and debug code.
- How to establish a network topology and write code for TCP and UDP connections
- how to use Wireshark
 - refer to **DisplayFilters** (<https://wiki.wireshark.org/DisplayFilters>)

What difficulty have you met in this lab?

- How to Use an OVA File on UTM

After trying the following methods, they didn't work:

1. Install VirtualBox 7.0.X beta version
 - Some features are not supported.
2. Attend a 24-hour open computer class to complete the homework
 - Some features are not supported in version 6.3.X.
3. Try to open the OVA file directly in UTM and VMware
 - Some features are not supported.
4. Build a nested VM
 - Some features are not supported.
5. Some features are not supported.
 - Seek assistance from ChatGPT and Google; however, I couldn't find a solution to the previous issues.

After asking my classmate, he provided me with a solution:

1. Convert the OVA file to the qcow2 format.
2. Modify the settings.
3. It runs very slowly on UTM, taking about 3 seconds for each operation.

reference ([https://medium.com/@hitoshi.shimomae/convert-ova-to-qcow2-and-start-it-with-utm-](https://medium.com/@hitoshi.shimomae/convert-ova-to-qcow2-and-start-it-with-utm-13fa3fc4c3db)

[13fa3fc4c3db](https://medium.com/@hitoshi.shimomae/convert-ova-to-qcow2-and-start-it-with-utm-13fa3fc4c3db)),