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.

```
        # Add hosts to a topology
        self.addHost("h1")
        self.addHost("h2")

        # 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=2)
        self.addLink("s2", "h3", bw=2)
```

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

```
    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)
```

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.

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

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

    # 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
```

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.

```
    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/TCI

    h3.cmd('iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_2.txt &')
    h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7778 > ../out/TCI

    h4.cmd('tcpdump -w ../out/TCP_h4.pcap &')
    h4.cmd('iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h4.txt &')
    h2.cmd('iperf -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7779 > ../out/TCI
```

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 &’)’.

```
    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/TCI

    h3.cmd('tcpdump -w ../out/TCP_h3.pcap &')
    h3.cmd('iperf -s -i 1 -t 5 -p 7778 > ../out/TCP_s_h3_2.txt &')
    h1.cmd('iperf -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7778 > ../out/TCI

    h4.cmd('tcpdump -w ../out/TCP_h4.pcap &')
    h4.cmd('iperf -s -i 1 -t 5 -p 7779 > ../out/TCP_s_h4.txt &')
    h2.cmd('iperf -c ' + str(h4.IP()) + ' -i 1 -t 5 -p 7779 > ../out/TCI
```

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.

```
    h3.cmd('tcpdump -w ../out/UDP_h3.pcap &')
    h3.cmd('iperf -u -s -i 1 -t 5 -p 7787 > ../out/UDP_s_h3_1.txt &')
    h1.cmd('iperf -u -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7787 > ../out/
    
    h3.cmd('iperf -u -s -i 1 -t 5 -p 7788 > ../out/UDP_s_h3_2.txt &')
    h1.cmd('iperf -u -c ' + str(h3.IP()) + ' -i 1 -t 5 -p 7788 > ../out/
    
    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/
```

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].

```
print ("***Get data of this packet***")
print (f"     src IP: {packets[TCP][0][1].src}")     # in IP layer
print (f"     dst IP: {packets[TCP][0][1].dst}")     # in IP layer
print (f"   src port: {packets[TCP][0][2].sport}") # in TCP layer
print (f"   dst port: {packets[TCP][0][2].dport}") # in TCP layer
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).

```
# read the file
INPUTPATH = '/home/cn2023-lab1/Desktop/out/TCP_h3.pcap'

# Read the pcap file
packets = rdpcap(INPUTPATH)

# Initialize the variable to store total transmit bits for port 7777
total_transmit_bits = 0

# Iterate through all TCP packets
for packet in packets[TCP]:
    # Get the source and destination ports
    src_port = packet[2].sport
    dst_port = packet[2].dport

    # Check if the source port is 7777
    if dst_port == 7777:
        # Calculate the transmit bits for the current packet
        transmit_bits = len(packet)

        # Accumulate the transmit bits to the total
        total_transmit_bits += transmit_bits

# Output the total transmit bits for port 7777
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.t
```

- 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't 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

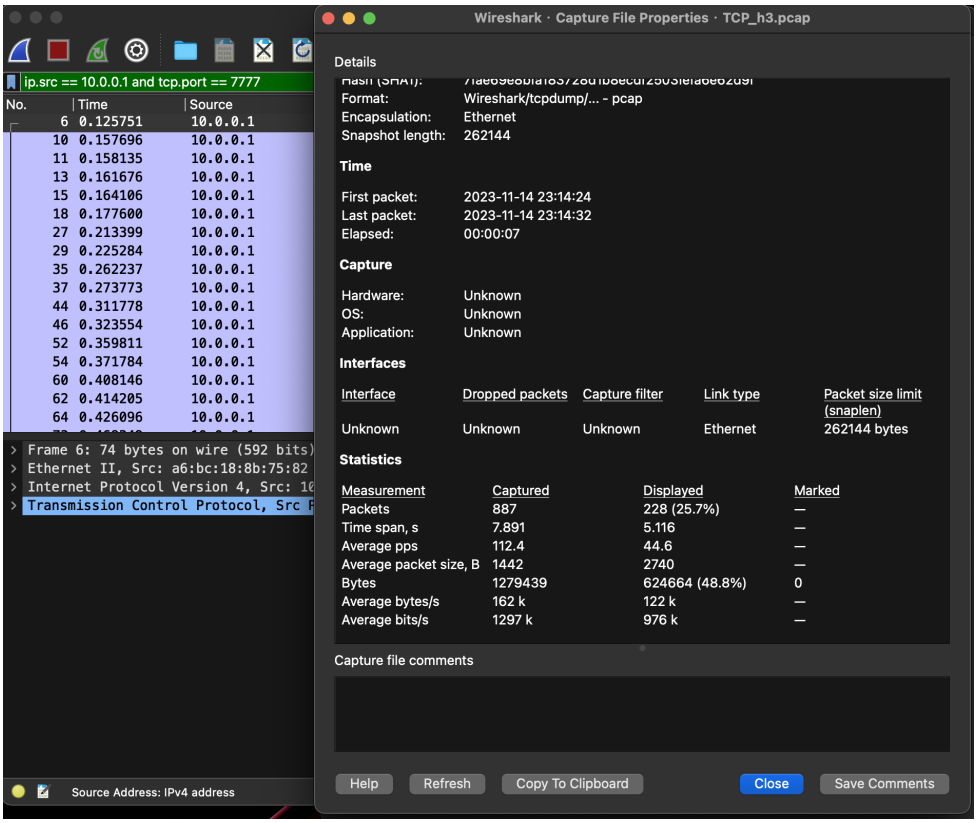
Show the results of computeRate.py (http://computeRate.py) and statistics of Wireshark

computeRateResult

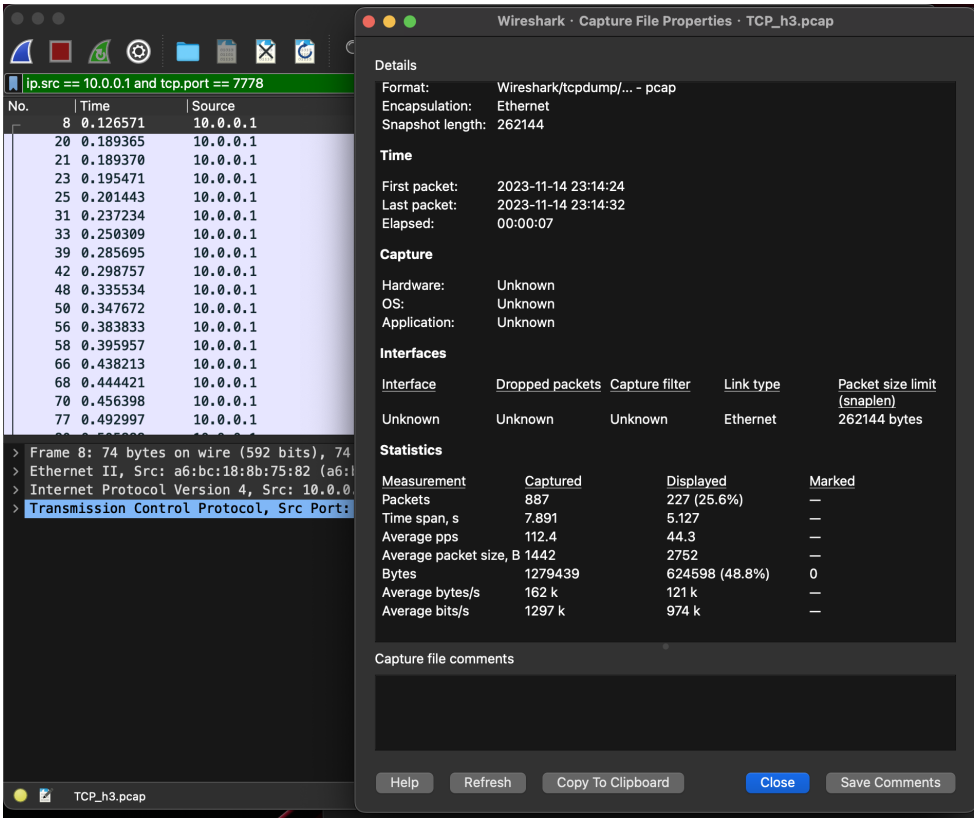
```
cn2023-lab1@cn2023lab1-VirtualBox:~/Desktop/Lab1-Liao-Hsiu-I/src$ python3 computeRate.py
--- TCP ---
Flow1(h1->h3): 0.9994624 Mbps
Flow2(h1->h3): 0.9993568 Mbps
Flow3(h2->h4): 1.9754208 Mbps

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

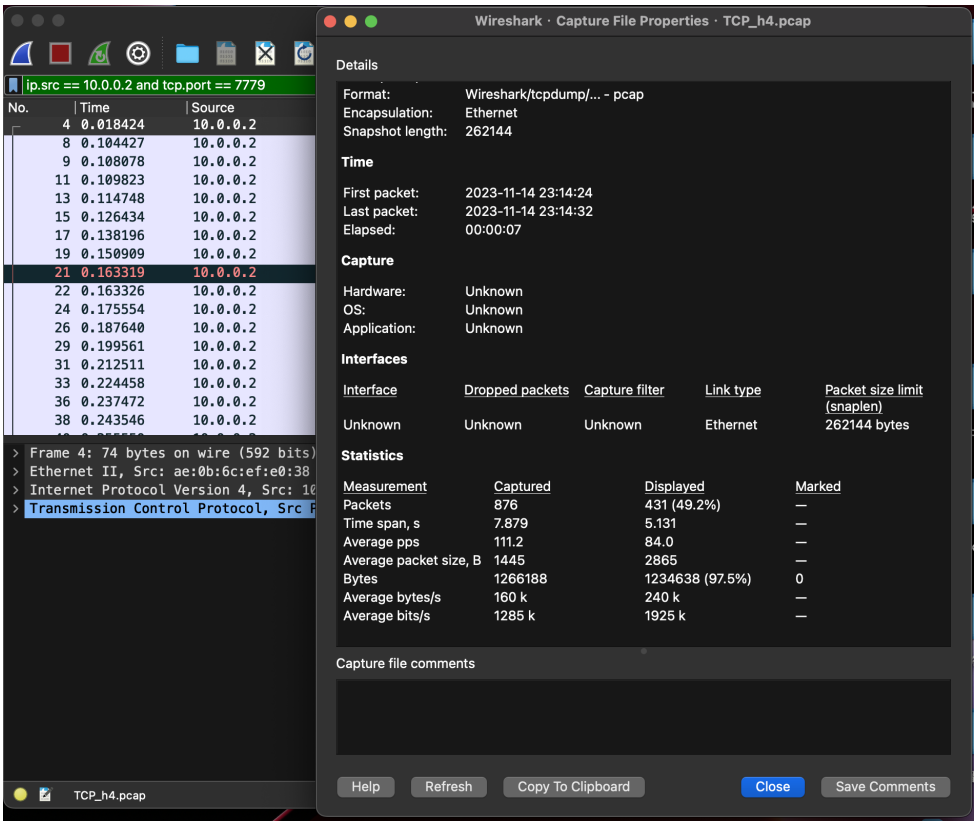
TCP flow1(h1->h3)



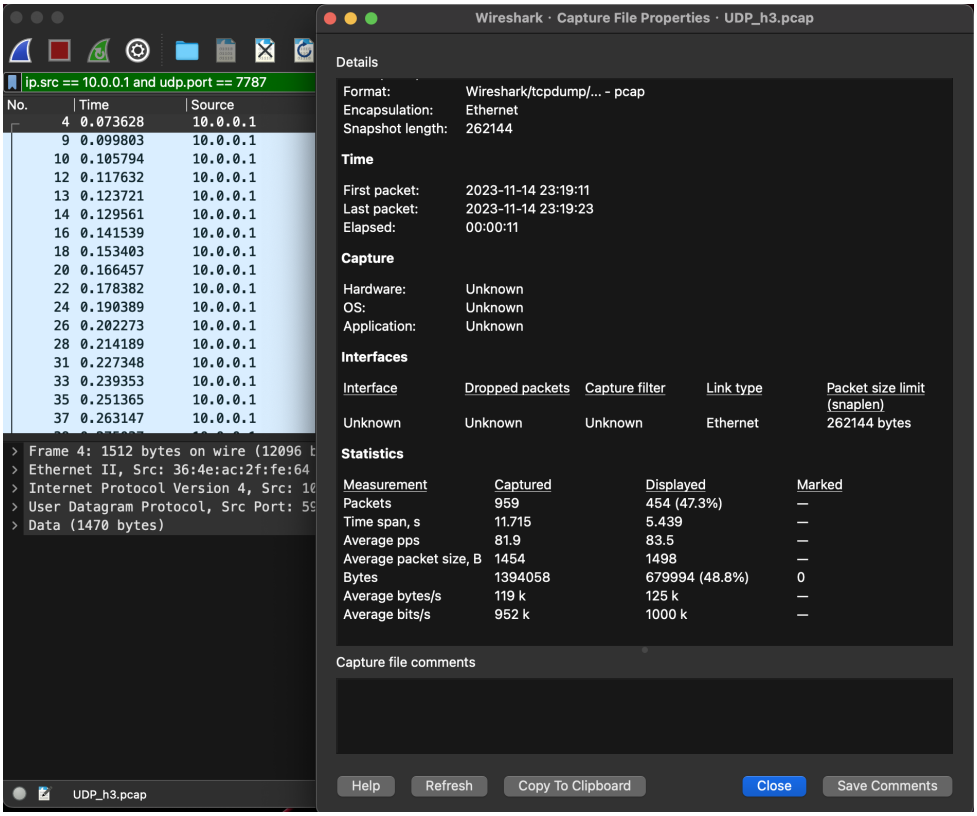
TCP flow2(h1->h3)



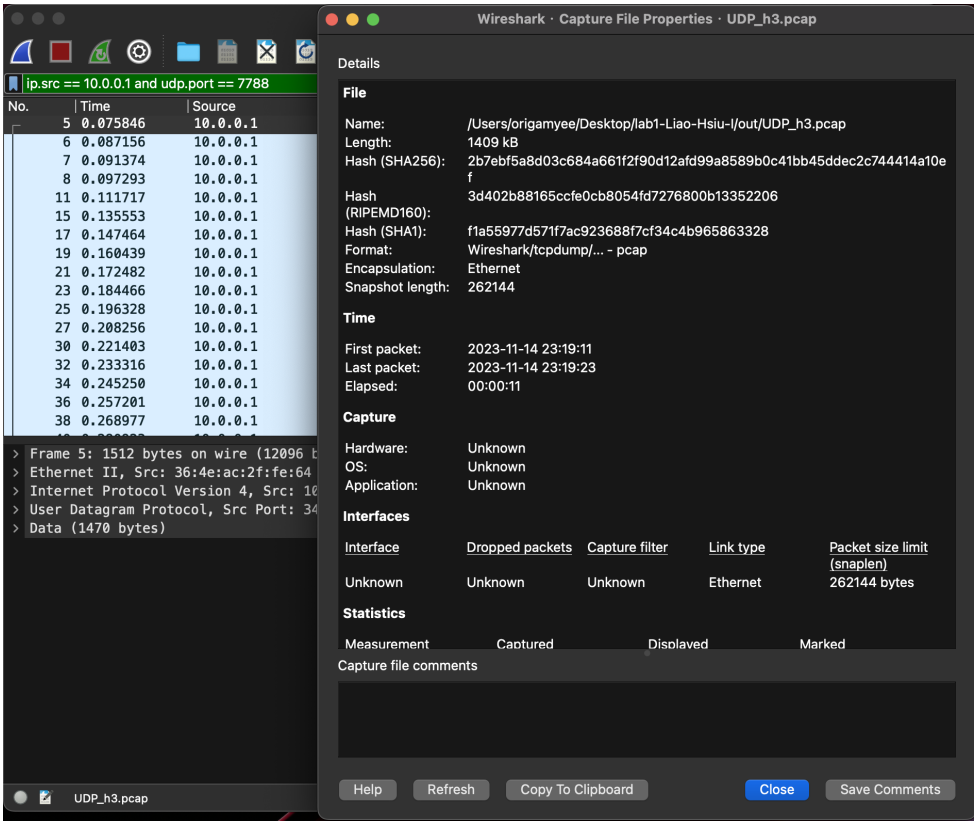
TCP flow3(h2->h4)



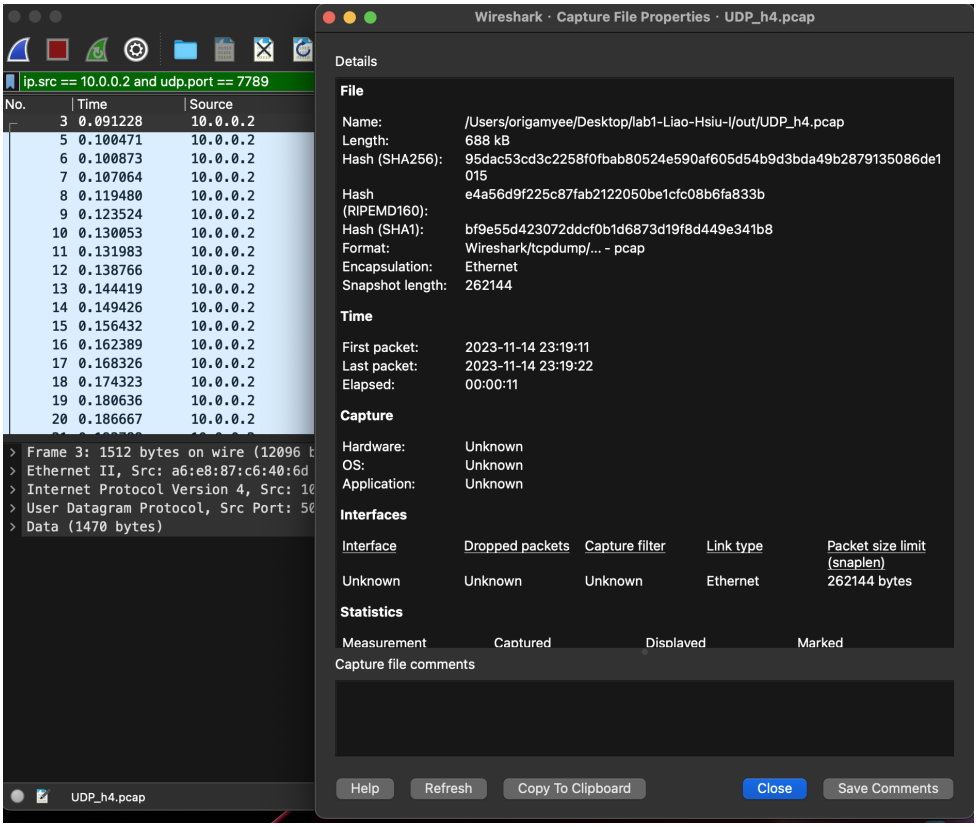
UDP flow1(h1->h3)



UDP flow2(h1->h3)



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-13fa3fc4c3db)