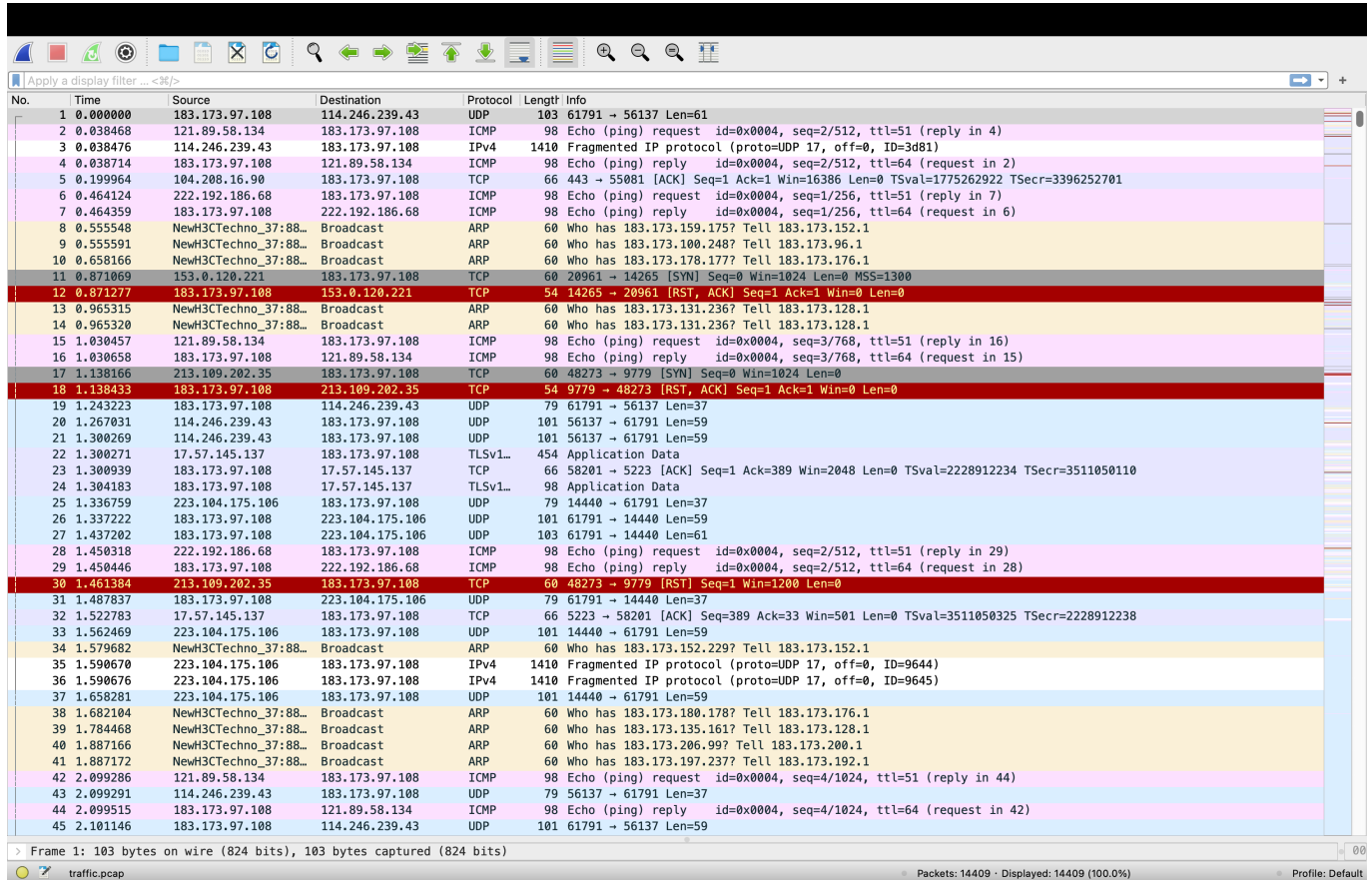


# Network Data Capture and Analysis

Name: 朱志杭

Student ID: 2022012069

## One screenshot of the captured web data



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	183.173.97.108	114.246.239.43	UDP	103	61791 → 56137 Len=61
2	0.038468	121.89.58.134	183.173.97.108	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=51 (reply in 4)
3	0.038476	114.246.239.43	183.173.97.108	IPv4	1410	Fragmented IP protocol (proto=UDP 17, off=0, ID=3d81)
4	0.038714	183.173.97.108	121.89.58.134	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512, ttl=64 (request in 2)
5	0.199964	104.208.16.90	183.173.97.108	TCP	66	443 → 55081 [ACK] Seq=1 Ack=1 Win=16386 Len=0 TSval=1775262922 TSecr=3396252701
6	0.464124	222.192.186.68	183.173.97.108	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=51 (reply in 7)
7	0.464359	183.173.97.108	222.192.186.68	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256, ttl=64 (request in 6)
8	0.555548	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.159.175? Tell 183.173.152.1
9	0.555591	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.100.248? Tell 183.173.96.1
10	0.658166	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.178.177? Tell 183.173.176.1
11	0.871069	153.0.120.221	183.173.97.108	TCP	60	20961 → 14265 [SYN] Seq=0 Win=1024 Len=0 MSS=1300
12	0.871277	183.173.97.108	153.0.120.221	TCP	54	14265 → 20961 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	0.965315	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.131.236? Tell 183.173.128.1
14	0.965320	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.131.236? Tell 183.173.128.1
15	1.030457	121.89.58.134	183.173.97.108	ICMP	98	Echo (ping) request id=0x0004, seq=3/768, ttl=51 (reply in 16)
16	1.030658	183.173.97.108	121.89.58.134	ICMP	98	Echo (ping) reply id=0x0004, seq=3/768, ttl=64 (request in 15)
17	1.138166	213.109.202.35	183.173.97.108	TCP	60	48273 → 9779 [SYN] Seq=0 Win=1024 Len=0
18	1.138433	183.173.97.108	213.109.202.35	TCP	54	9779 → 48273 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	1.243223	183.173.97.108	114.246.239.43	UDP	79	61791 → 56137 Len=37
20	1.267031	114.246.239.43	183.173.97.108	UDP	101	56137 → 61791 Len=59
21	1.300269	114.246.239.43	183.173.97.108	UDP	101	56137 → 61791 Len=59
22	1.300271	17.57.145.137	183.173.97.108	TLSv1...	454	Application Data
23	1.300939	183.173.97.108	17.57.145.137	TCP	66	58201 → 5223 [ACK] Seq=1 Ack=389 Win=2048 Len=0 TSval=2228912234 TSecr=3511050110
24	1.304183	183.173.97.108	17.57.145.137	TLSv1...	98	Application Data
25	1.336759	223.104.175.106	183.173.97.108	UDP	79	14440 → 61791 Len=37
26	1.337222	183.173.97.108	223.104.175.106	UDP	101	61791 → 14440 Len=59
27	1.437202	183.173.97.108	223.104.175.106	UDP	103	61791 → 14440 Len=61
28	1.450318	222.192.186.68	183.173.97.108	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=51 (reply in 29)
29	1.450446	183.173.97.108	222.192.186.68	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512, ttl=64 (request in 28)
30	1.461384	213.109.202.35	183.173.97.108	TCP	60	48273 → 9779 [RST] Seq=1 Win=1200 Len=0
31	1.487837	183.173.97.108	223.104.175.106	UDP	79	61791 → 14440 Len=37
32	1.522783	17.57.145.137	183.173.97.108	TCP	66	5223 → 58201 [ACK] Seq=389 Ack=33 Win=501 Len=0 TSval=3511050325 TSecr=2228912238
33	1.562469	223.104.175.106	183.173.97.108	UDP	101	14440 → 61791 Len=59
34	1.579682	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.152.229? Tell 183.173.152.1
35	1.590670	223.104.175.106	183.173.97.108	IPv4	1410	Fragmented IP protocol (proto=UDP 17, off=0, ID=9644)
36	1.590676	223.104.175.106	183.173.97.108	IPv4	1410	Fragmented IP protocol (proto=UDP 17, off=0, ID=9645)
37	1.658281	223.104.175.106	183.173.97.108	UDP	101	14440 → 61791 Len=59
38	1.682104	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.180.178? Tell 183.173.176.1
39	1.784468	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.135.161? Tell 183.173.128.1
40	1.887166	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.206.99? Tell 183.173.200.1
41	1.887172	NewH3CTechno_37:88...	Broadcast	ARP	60	Who has 183.173.197.237? Tell 183.173.192.1
42	2.099286	121.89.58.134	183.173.97.108	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=51 (reply in 44)
43	2.099291	114.246.239.43	183.173.97.108	UDP	79	56137 → 61791 Len=37
44	2.099515	183.173.97.108	121.89.58.134	ICMP	98	Echo (ping) reply id=0x0004, seq=4/1024, ttl=64 (request in 42)
45	2.101146	183.173.97.108	114.246.239.43	UDP	101	61791 → 56137 Len=59

Use tcpdump to capture the traffic on a physical network connected to my computer and save it to a file.

I use the order

```
tcpdump -G 300 -w traffic-%Y%m%d%H%M%S.pcap
```

to capture the traffic on my computer(Mac). This will make a file named with the start time in every five minutes, all the packages captured in this five minutes will be saved in this file. When I want to analysis the packages, I will delete the time in its name.

## Description of my data processing methods and dataprocessing program.

I use python package scapy to process the packages I captured. Here I will describe how do I answer questions in procedure 2.

1. In order to give the load proportion of different transport layer protocols carried by IP packets, I use following code to process. I find that in most case the following four protocols are used.

```
# 1) Load proportion of different transport layer protocols
transport_protocols = {'TCP': 0, 'UDP': 0, 'ICMP':0, 'ARP':0 }
for packet in packets:
    if IP in packet:
        if TCP in packet:
            transport_protocols['TCP'] += 1
        elif UDP in packet:
            transport_protocols['UDP'] += 1
        elif ICMP in packet:
            transport_protocols['ICMP'] += 1
        elif NTP in packet:
            transport_protocols['ARP'] += 1

import matplotlib.pyplot as plt
plt.pie(transport_protocols.values(),
labels=transport_protocols.keys(), autopct='%1.1f%%')
plt.title('Proportion of Different Transport Layer Protocols')
plt.show()
```

This will check all the IP packets in the capture file and count the number of packages using TCP protocol and packages using UDP protocol. After that, I use matplotlib to draw a pie chart to show.

2. In the program, I check all the IP packages captured. If flags is 1 or frag is not 0 in the IP word in a package, this package is fragmented. I use following code to achieve this.

```
# 2) Identify fragmented IP packets
fragmented_packets = []

# Iterate through all packets
for packet in packets:
    # Check if IP header exists
    if IP in packet:
        # Check if packet is fragmented
        if packet[IP].flags & 0x1: # Check if the 'more fragments'
flag is set
            fragmented_packets.append(packet)
        elif packet[IP].frag != 0: # Check if fragment offset is
non-zero, indicating fragmented packet
            fragmented_packets.append(packet)

print("Number of fragmented IP packets:", len(fragmented_packets))
```

After checking all packages, I will print the number of fragmented packages.

3. In the program, I use the folowing code to give the cumulative distribution curve of IP packet length. I first get all the packages using TCP(UDP) protocol and then use numpy to help draw the cumulative

probability.

```
# 3) Cumulative distribution curve of IP packet length
tcp_lengths = [len(packet) for packet in packets if IP in packet and
TCP in packet]
udp_lengths = [len(packet) for packet in packets if IP in packet and
UDP in packet]

# Plot cumulative distribution curve
import numpy as np

plt.hist(tcp_lengths, bins=np.arange(0, max(tcp_lengths), 100),
cumulative=True, density=True, histtype='step', label='TCP')
plt.hist(udp_lengths, bins=np.arange(0, max(udp_lengths), 100),
cumulative=True, density=True, histtype='step', label='UDP')
plt.xlabel('IP Packet Length')
plt.ylabel('Cumulative Probability')
plt.title('Cumulative Distribution of IP Packet Length')
plt.legend()
plt.show()
```

After drawing the cumulative probability, I compare the difference under different loads. (TCP or UDP)

4. I find the broadcast packets using following codes. The destination of broadcast\_packets is `ff:ff:ff:ff:ff:ff` and I checked all packages to find them.

```
# 4) Find broadcast packets
broadcast_packets = [packet for packet in packets if Ether in packet
and packet[Ether].dst == 'ff:ff:ff:ff:ff:ff']

# Print number of broadcast packets
print("Number of broadcast packets:", len(broadcast_packets))
```

5. I use the order

```
tcpdump -w ipv6_traffic.pcap 'ip6' -G 300 -W 1
```

to capture ipv6 traffic.

## Process the data I've collected and try to answer questions.

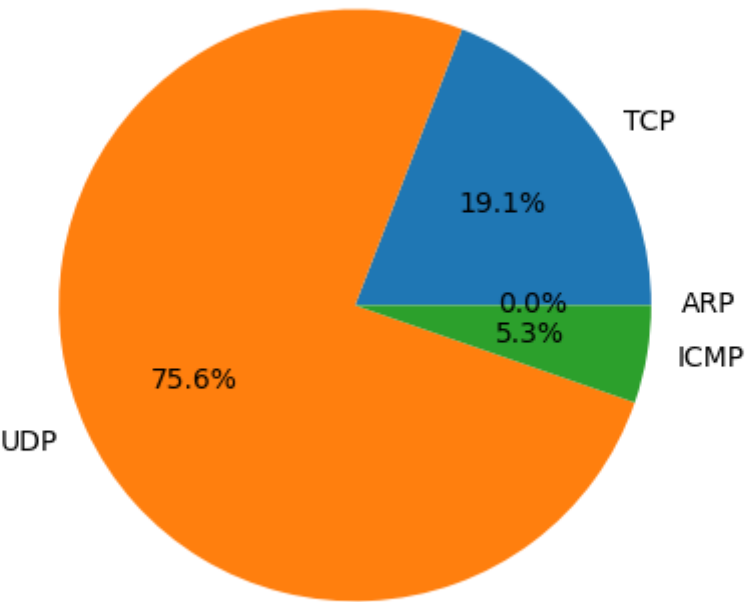
I choose a file created by the order and change its name to traffic.pcap. Then run the order

```
python analysis.py
```

Here comes the result and my answer.

- 1. A pie chart to show the load proportion of different transport layer protocols carried by IP packets

Proportion of Different Transport Layer Protocols

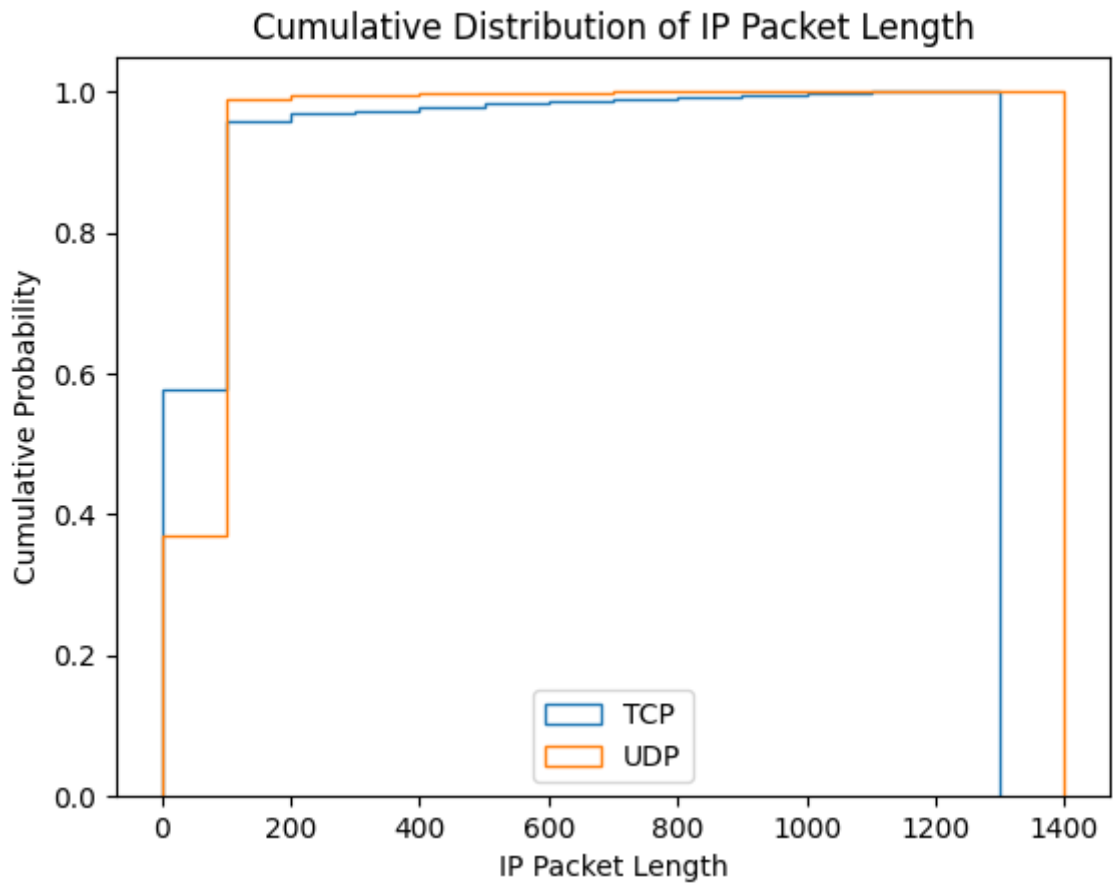


- 2. If flags is 1 or frag is not 0 in the IP word in a package, this package is fragmented.

Number of fragmented IP packets: 1939

The result is that I captured 1939 fragmented packages.

3. The cumulative distribution curve of IP packet length.



4. The result is that I find 1406 broadcast packets.

Number of broadcast packets: 1406

According to the cumulative distribution function, the maximum length of IP packets transmitted by TCP is smaller than the maximum length of IP packets transmitted by UDP. At the same time, there are significantly more packets with lengths less than 200 transmitted by TCP.

5. According to the file `ipv6_traffic.pcap`, the tcpdump filters successfully capture ipv6 traffic.  
[ipv6\\_traffic.pcap](#)