Computer Networks CS 331
Assignment-1

Gaurav Joshi (21110065) Husain Malwat (21110117)

GitHub Repository: https://github.com/Gaurav17Joshi/CN_assignment1

Part-1

In this part, we have made a sniffer called fast_sniffer.py, and it can be run and used.

- Protocols of 2.pcap:-

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s
Frame	100.0	805997	100.0	364642055	6
Ethernet	100.0	805997	3.5	12800341	0
Internet Protocol Version 6	0.1	436	0.0	17712	0
User Datagram Protocol	0.0	402	0.0	3216	0
Internet Control Message Protocol v6	0.0	34	0.0	2068	0
Internet Protocol Version 4	99.9	805543	4.4	16112460	0
User Datagram Protocol	19.0	153234	0.3	1225872	0
Transmission Control Protocol	80.4	647723	3.9	14239696	0
Internet Group Management Protocol	0.0	286	0.0	6884	0
Internet Control Message Protocol	0.5	4300	0.1	217975	0
Address Resolution Protocol	0.0	18	0.0	504	0

The transport layer protocols in the 2.pcap are:-

1. UDP
2. TCP
3. ICMP
4. IGMP

The information we need from each packet is:

1. Packet size (q1)
2. Source ip (q2, q3)
3. Source port (q2, q3)
4. Destination ip (q2, q3)
5. Destination port (q2, q3)

Based on the problem, and the 2.pcap file, these were the different transport layers we had and the information we needed, so to make our sniffer as fast as possible, we only sniffed and stored this information. Next, we run different python scripts to use the sniffed data and get the various statistical results.

We have given the instructions to run the code in the github repo also.

To run the sniffer with tcpreplay:-

Steps:-

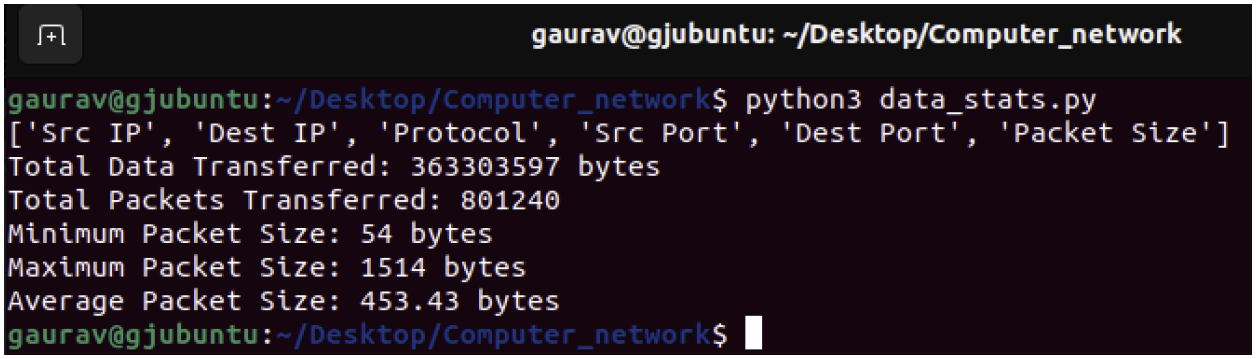
1. In python install the required libraries; pip install socket struct csv
2. Run the fast_sniffer.py: sudo python3 fast_sniffer.py
3. Run the tcpreplay of the 2.pcap in another terminal: sudo tcpreplay -i enp0s1 -M 10 2.pcap. (here enp0s1 can be replace by other ports “eth”)

Q1) Packet Capture and Analysis

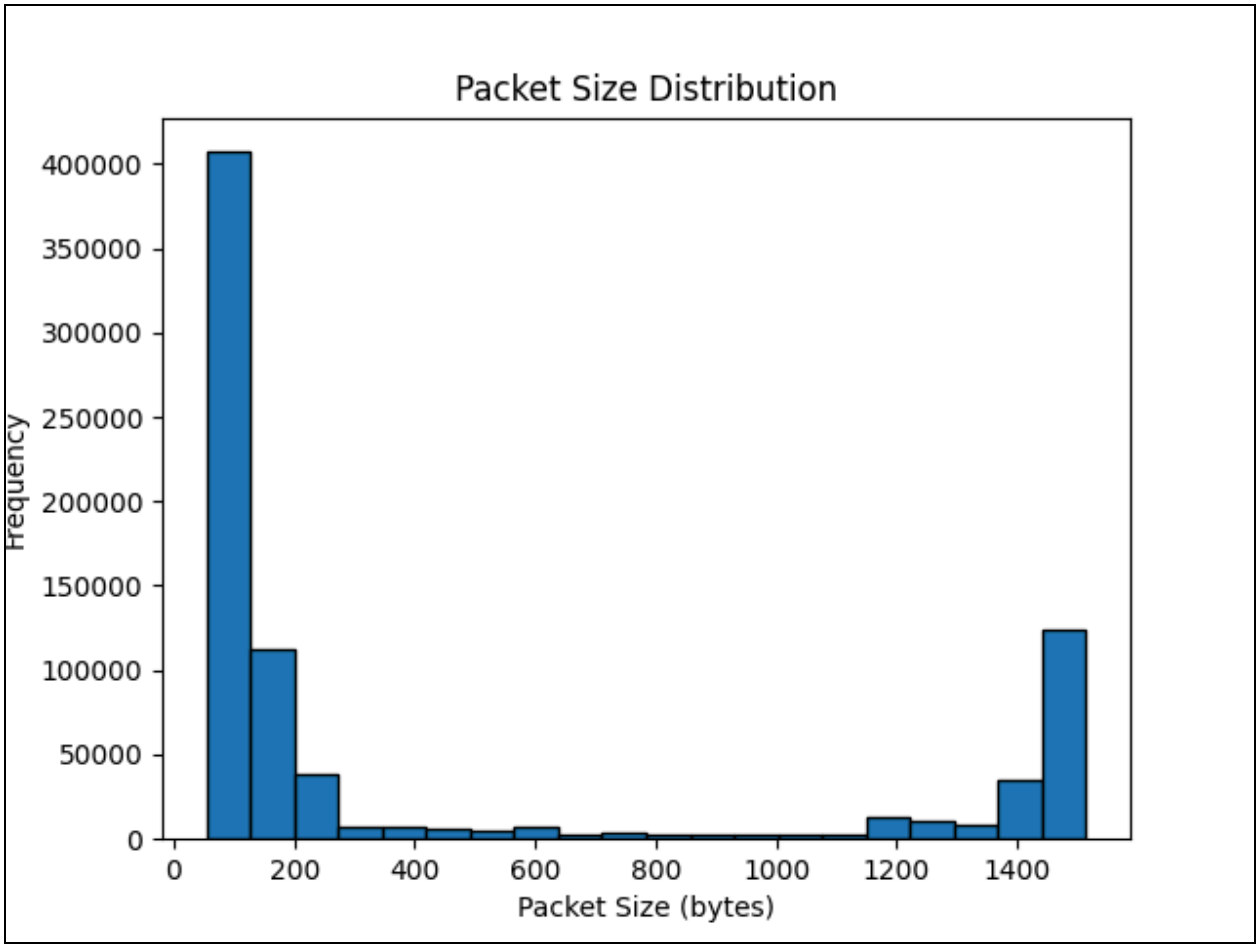
This program captures network packets using a raw socket and logs relevant details (source/destination IPs, protocol, ports, and packet sizes) into a CSV file. The captured data is then analyzed to determine total data transferred, packet count, and size distribution. A histogram visualizes the packet size distribution.

Steps to Run:

1. Run the packet capture script with superuser privileges (sudo python3 fast_sniffer.py).
2. Stop capture using Ctrl+C, which saves packet data to fast_captured_packets_info.csv.
3. Run the analysis script (python3 data_stats.py) to generate statistics and a packet size histogram (Packet_Size_dist.png).



Here, are the statistics of the data, including a histogram:-



x axis = packet size, and the y-axis = number of packets.

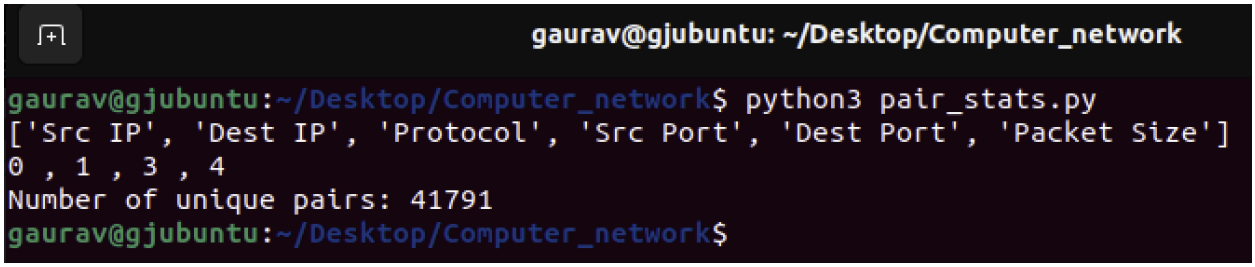
Q2.)Unique Source-Destination Pairs Extraction

This script (pair_stats.py) extracts and identifies unique source-destination pairs (IP:port combinations) from captured packet data stored in captured_packets_info.csv. The program reads the CSV file, locates the necessary columns, and stores unique (source IP, source port, destination IP, destination port) pairs in a set to eliminate duplicates.

Steps to Run:

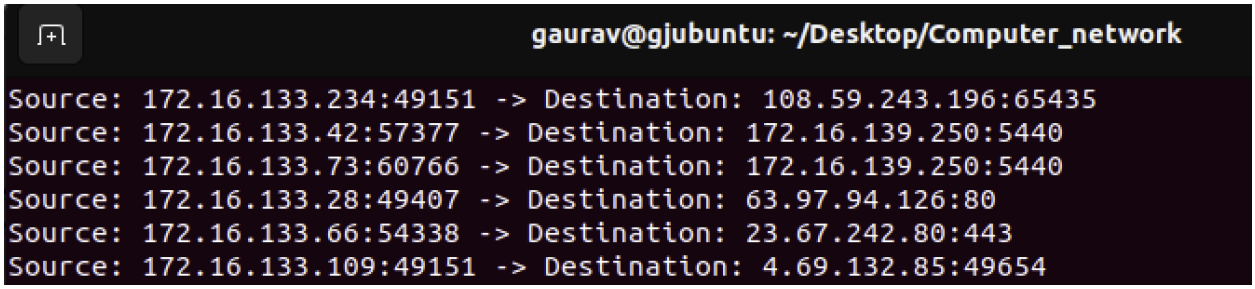
1. Ensure captured_packets_info.csv exists with proper packet data.
2. Run the script:
   “python3 pair_stats.py”

3. The script outputs the total number of unique pairs.



   - Here, we have 41791 unique Source destination pairs.

4. Set print_all = True to list them. We can also print out all of them:-



Q3. Flow Calculation and Data Transfer Analysis

This script calculates flow statistics for each source and destination IP, as well as identifies the source-destination pair with the most data transferred. It reads the captured packet data from fast_captured_packets_info.csv and processes it to populate dictionaries that track:

1. Source flows: Number of flows initiated by each source IP.
2. Destination flows: Number of flows received by each destination IP.
3. Data transferred: Total data transferred for each unique source-destination IP:port pair.

The script also prints the total number of source and destination flows and identifies the source-destination pair with the maximum data transfer.

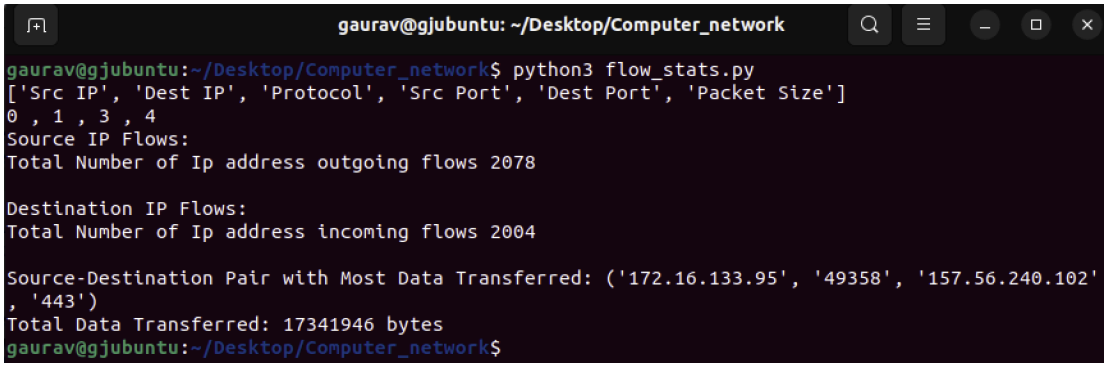
Steps to Run:

1. Ensure the file fast_captured_packets_info.csv contains the relevant packet data.
2. Run the script:

   “python3 flow_calculation.py”

3. The script outputs:
    - Number of source and destination flows.
    - A dictionary of source flows.
    - A dictionary of destination flows.
    - The source-destination pair with the most data transferred.

The total number of unique outgoing and incoming flows, and the pair with most data transferred.



4. Setting the `print_all` flag to `True` will display all the source and destination flows.

Source Flows:-

```
gaurav@gjubuntu:~/Desktop/Computer_network$ python3 flow_stats.py
['Src IP', 'Dest IP', 'Protocol', 'Src Port', 'Dest Port', 'Packet Size']
0 , 1 , 3 , 4
Source IP Flows:
Total Number of Ip address source flows 2078
192.168.64.1: 4 flows
65.54.95.68: 1275 flows
65.54.95.75: 766 flows
65.54.95.140: 658 flows
204.14.234.85: 1036 flows
65.54.186.19: 66 flows
192.168.3.131: 4294 flows
```

Destination Flows:-

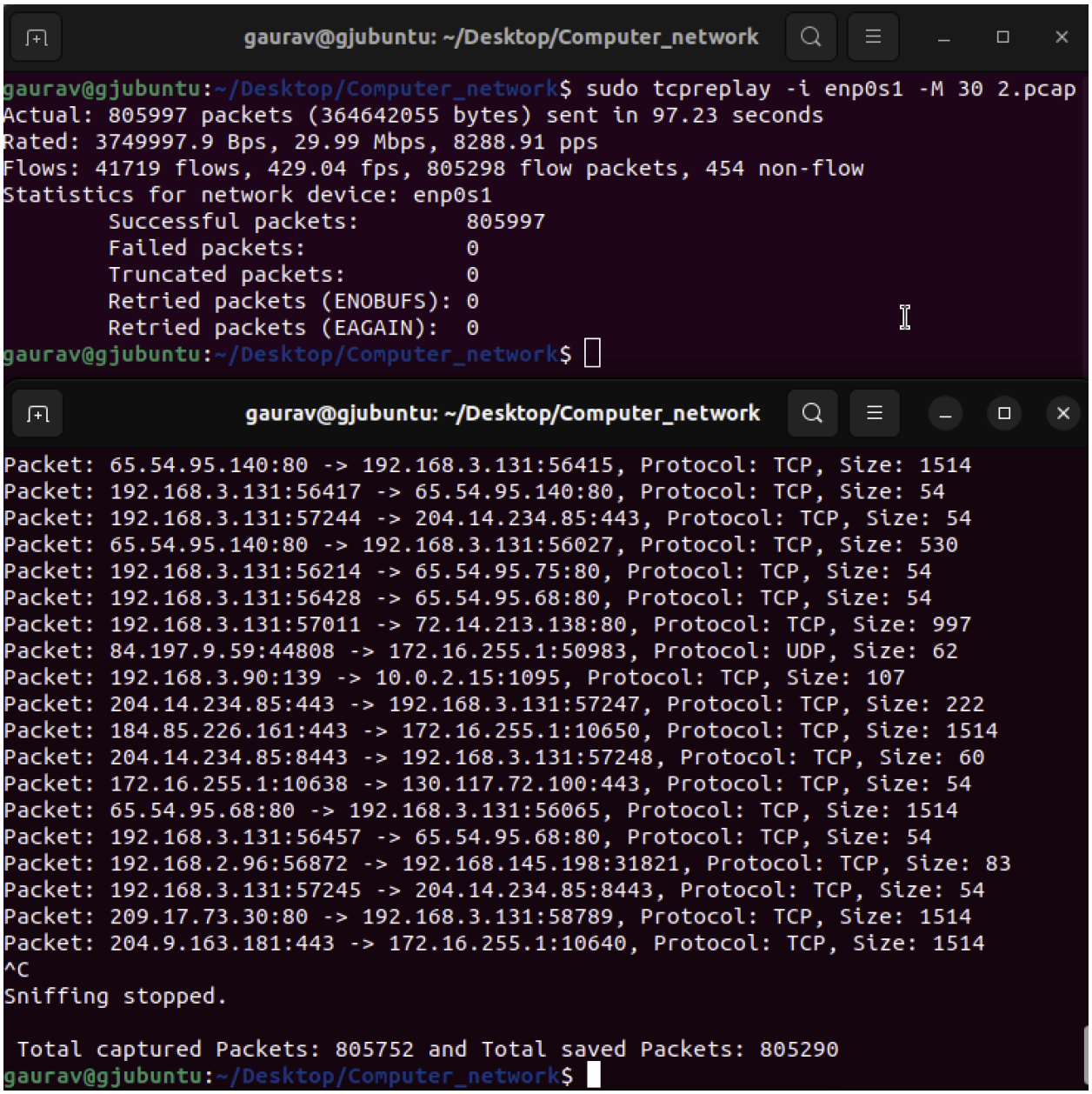
```
Destination IP Flows:
Total Number of Ip address destination flows 2004
224.0.0.251: 29 flows
192.168.3.131: 6184 flows
10.0.2.15: 808 flows
207.46.0.109: 29 flows
65.54.95.68: 664 flows
172.16.255.1: 1219 flows
204.14.234.85: 740 flows
65.54.95.140: 495 flows
172.16.0.1: 10 flows
147.31.122.1: 90 flows
```

Q4. Sniffer Timing

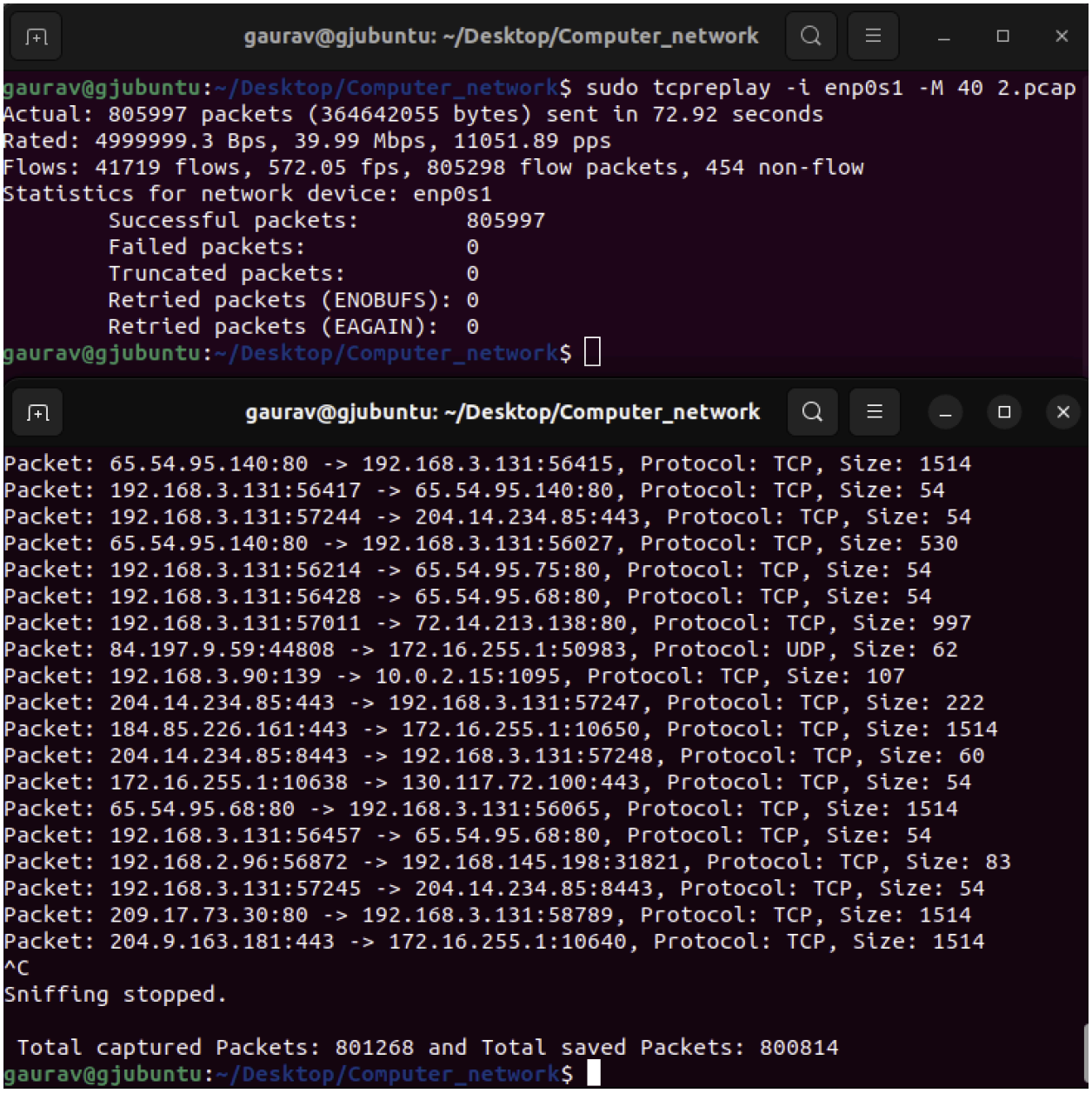
PART1: Self
We will run tcpreplay with a speed cap for ‘mbps’ and ‘pps’

Tcpreplay commands:-
Run :
sudo tcpreplay -i enp0s1 -M 10 2.pca (for 10 mbps)

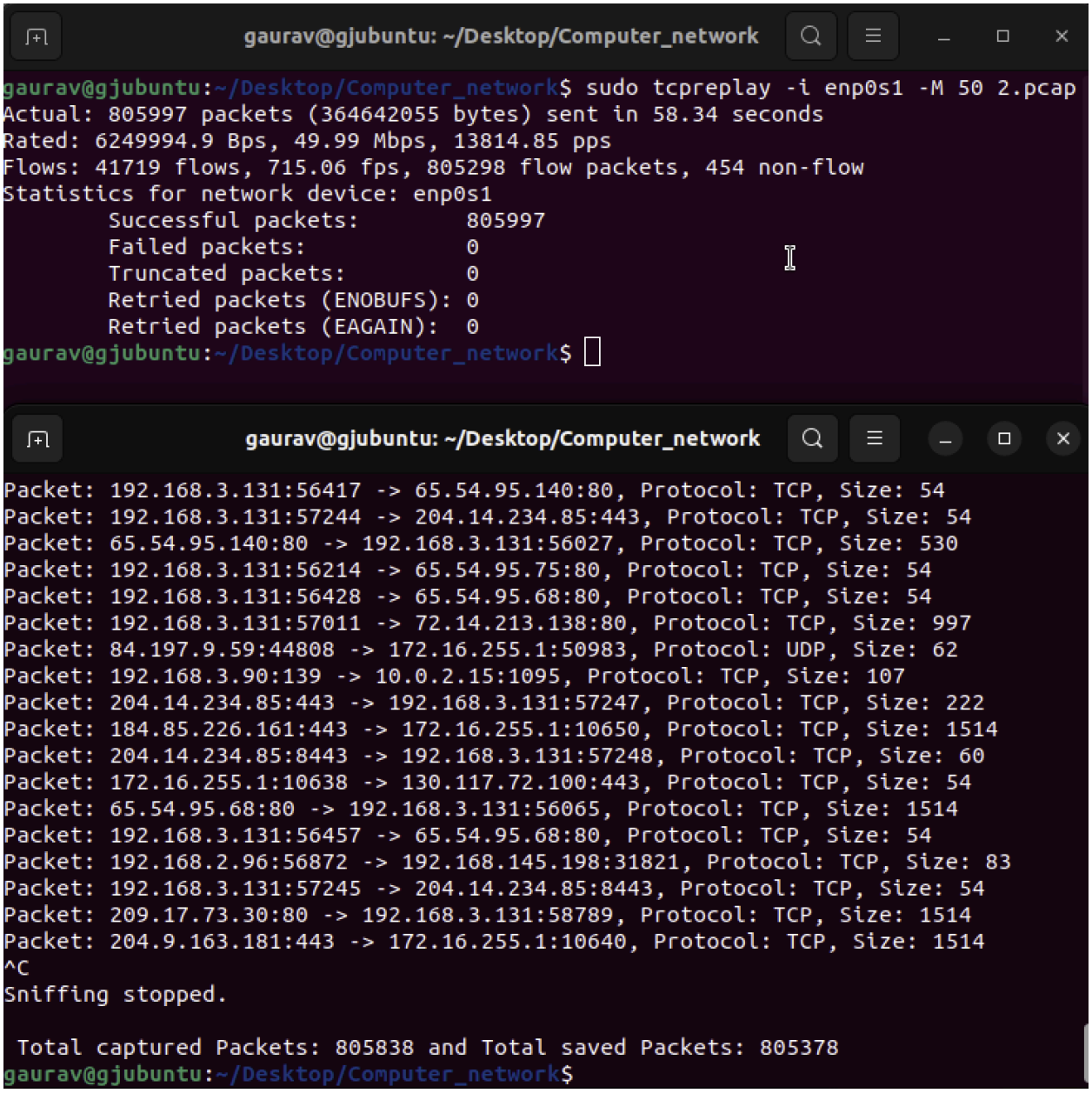
30 Mbps



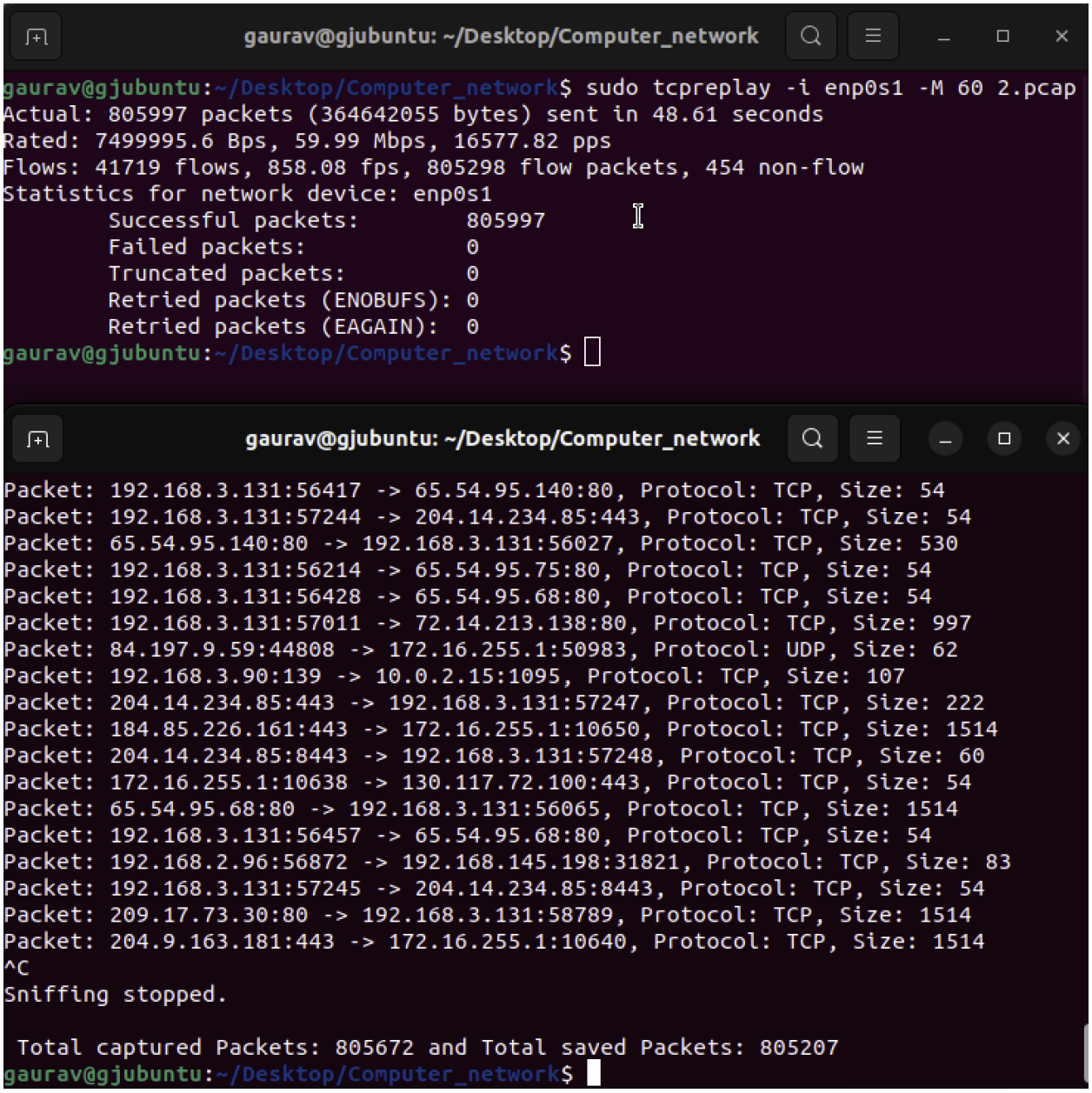
40 Mbps


gaurav@gjubuntu: ~/Desktop/Computer_network
gaurav@gjubuntu:~/Desktop/Computer_network$ sudo tcpreplay -i enp0s1 -M 40 2.pcap
Actual: 805997 packets (364642055 bytes) sent in 72.92 seconds
Rated: 4999999.3 Bps, 39.99 Mbps, 11051.89 pps
Flows: 41719 flows, 572.05 fps, 805298 flow packets, 454 non-flow
Statistics for network device: enp0s1
Successful packets: 805997
Failed packets: 0
Truncated packets: 0
Retried packets (ENOBUFS): 0
Retried packets (EAGAIN): 0
gaurav@gjubuntu:~/Desktop/Computer_network$
gaurav@gjubuntu: ~/Desktop/Computer_network
Packet: 65.54.95.140:80 -> 192.168.3.131:56415, Protocol: TCP, Size: 1514
Packet: 192.168.3.131:56417 -> 65.54.95.140:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:57244 -> 204.14.234.85:443, Protocol: TCP, Size: 54
Packet: 65.54.95.140:80 -> 192.168.3.131:56027, Protocol: TCP, Size: 530
Packet: 192.168.3.131:56214 -> 65.54.95.75:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:56428 -> 65.54.95.68:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:57011 -> 72.14.213.138:80, Protocol: TCP, Size: 997
Packet: 84.197.9.59:44808 -> 172.16.255.1:50983, Protocol: UDP, Size: 62
Packet: 192.168.3.90:139 -> 10.0.2.15:1095, Protocol: TCP, Size: 107
Packet: 204.14.234.85:443 -> 192.168.3.131:57247, Protocol: TCP, Size: 222
Packet: 184.85.226.161:443 -> 172.16.255.1:10650, Protocol: TCP, Size: 1514
Packet: 204.14.234.85:8443 -> 192.168.3.131:57248, Protocol: TCP, Size: 60
Packet: 172.16.255.1:10638 -> 130.117.72.100:443, Protocol: TCP, Size: 54
Packet: 65.54.95.68:80 -> 192.168.3.131:56065, Protocol: TCP, Size: 1514
Packet: 192.168.3.131:56457 -> 65.54.95.68:80, Protocol: TCP, Size: 54
Packet: 192.168.2.96:56872 -> 192.168.145.198:31821, Protocol: TCP, Size: 83
Packet: 192.168.3.131:57245 -> 204.14.234.85:8443, Protocol: TCP, Size: 54
Packet: 209.17.73.30:80 -> 192.168.3.131:58789, Protocol: TCP, Size: 1514
Packet: 204.9.163.181:443 -> 172.16.255.1:10640, Protocol: TCP, Size: 1514
^C
Sniffing stopped.
Total captured Packets: 801268 and Total saved Packets: 800814
gaurav@gjubuntu:~/Desktop/Computer_network$

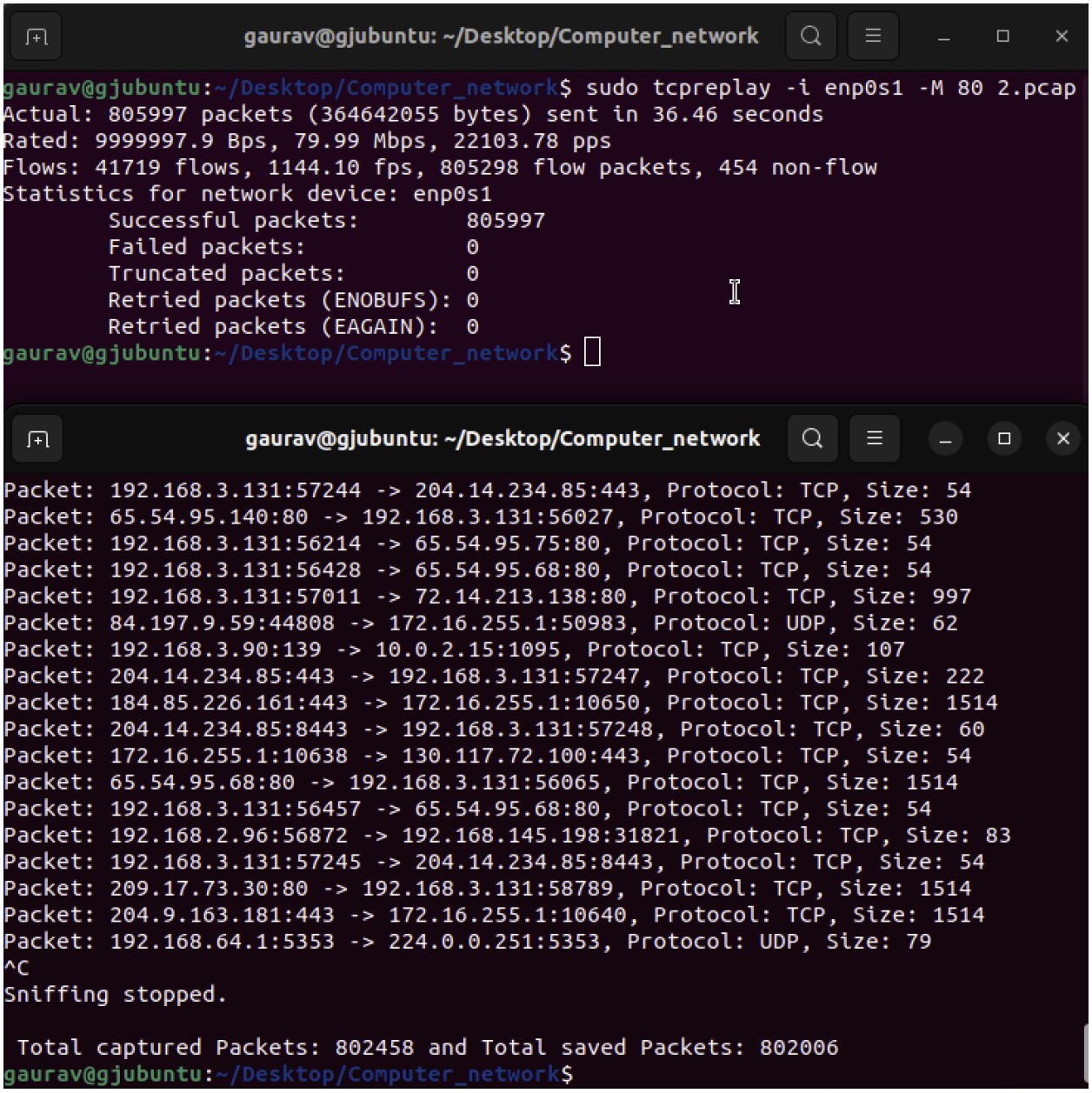
50 Mbps


gaurav@gjubuntu: ~/Desktop/Computer_network
gaurav@gjubuntu:~/Desktop/Computer_network$ sudo tcpreplay -i enp0s1 -M 50 2.pcap
Actual: 805997 packets (364642055 bytes) sent in 58.34 seconds
Rated: 6249994.9 Bps, 49.99 Mbps, 13814.85 pps
Flows: 41719 flows, 715.06 fps, 805298 flow packets, 454 non-flow
Statistics for network device: enp0s1
Successful packets: 805997
Failed packets: 0
Truncated packets: 0
Retried packets (ENOBUFS): 0
Retried packets (EAGAIN): 0
gaurav@gjubuntu:~/Desktop/Computer_network$
gaurav@gjubuntu: ~/Desktop/Computer_network
Packet: 192.168.3.131:56417 -> 65.54.95.140:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:57244 -> 204.14.234.85:443, Protocol: TCP, Size: 54
Packet: 65.54.95.140:80 -> 192.168.3.131:56027, Protocol: TCP, Size: 530
Packet: 192.168.3.131:56214 -> 65.54.95.75:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:56428 -> 65.54.95.68:80, Protocol: TCP, Size: 54
Packet: 192.168.3.131:57011 -> 72.14.213.138:80, Protocol: TCP, Size: 997
Packet: 84.197.9.59:44808 -> 172.16.255.1:50983, Protocol: UDP, Size: 62
Packet: 192.168.3.90:139 -> 10.0.2.15:1095, Protocol: TCP, Size: 107
Packet: 204.14.234.85:443 -> 192.168.3.131:57247, Protocol: TCP, Size: 222
Packet: 184.85.226.161:443 -> 172.16.255.1:10650, Protocol: TCP, Size: 1514
Packet: 204.14.234.85:8443 -> 192.168.3.131:57248, Protocol: TCP, Size: 60
Packet: 172.16.255.1:10638 -> 130.117.72.100:443, Protocol: TCP, Size: 54
Packet: 65.54.95.68:80 -> 192.168.3.131:56065, Protocol: TCP, Size: 1514
Packet: 192.168.3.131:56457 -> 65.54.95.68:80, Protocol: TCP, Size: 54
Packet: 192.168.2.96:56872 -> 192.168.145.198:31821, Protocol: TCP, Size: 83
Packet: 192.168.3.131:57245 -> 204.14.234.85:8443, Protocol: TCP, Size: 54
Packet: 209.17.73.30:80 -> 192.168.3.131:58789, Protocol: TCP, Size: 1514
Packet: 204.9.163.181:443 -> 172.16.255.1:10640, Protocol: TCP, Size: 1514
^C
Sniffing stopped.
Total captured Packets: 805838 and Total saved Packets: 805378
gaurav@gjubuntu:~/Desktop/Computer_network$

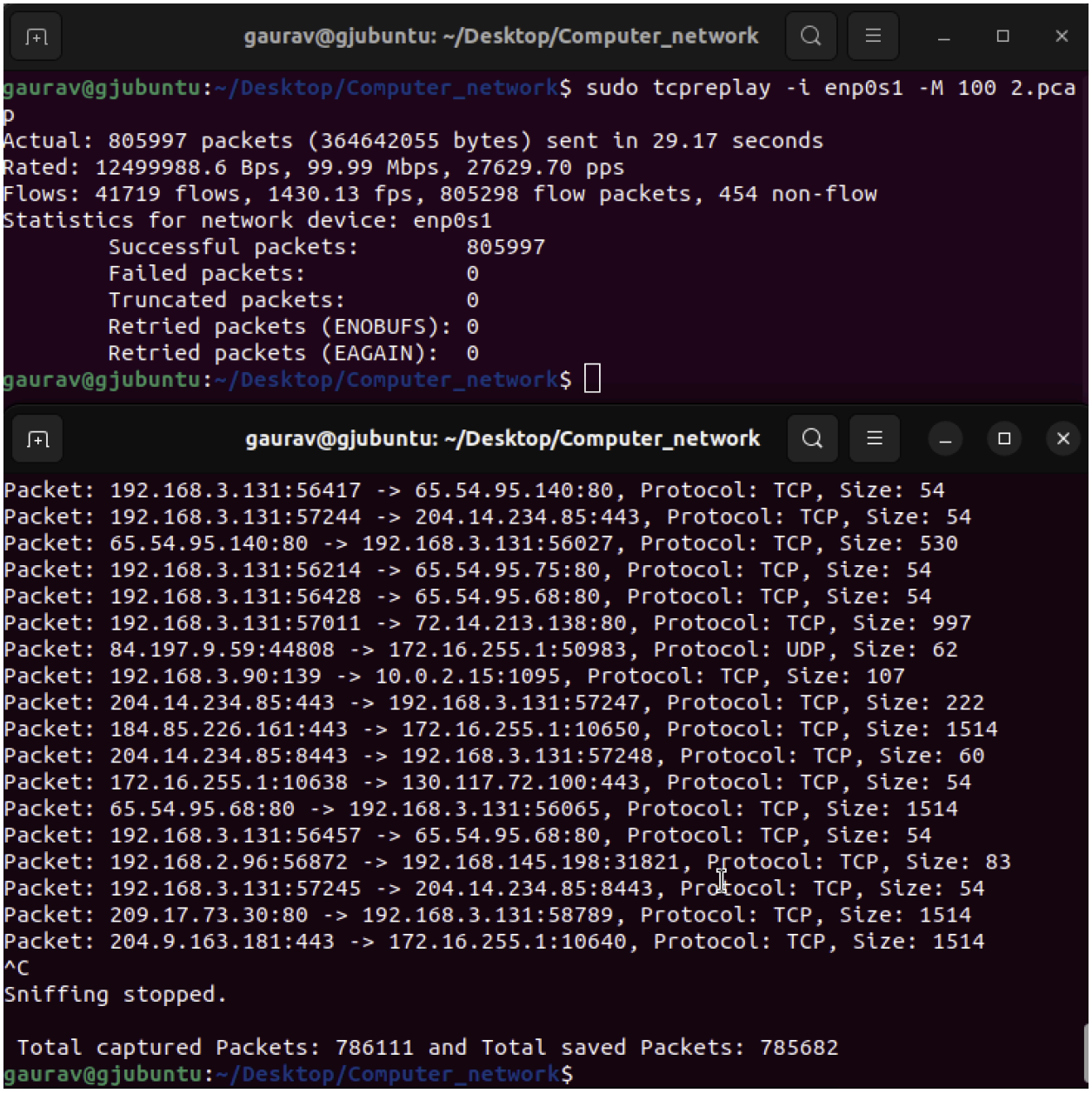
60 Mbps



80 Mbps (It now captures almost all packets)



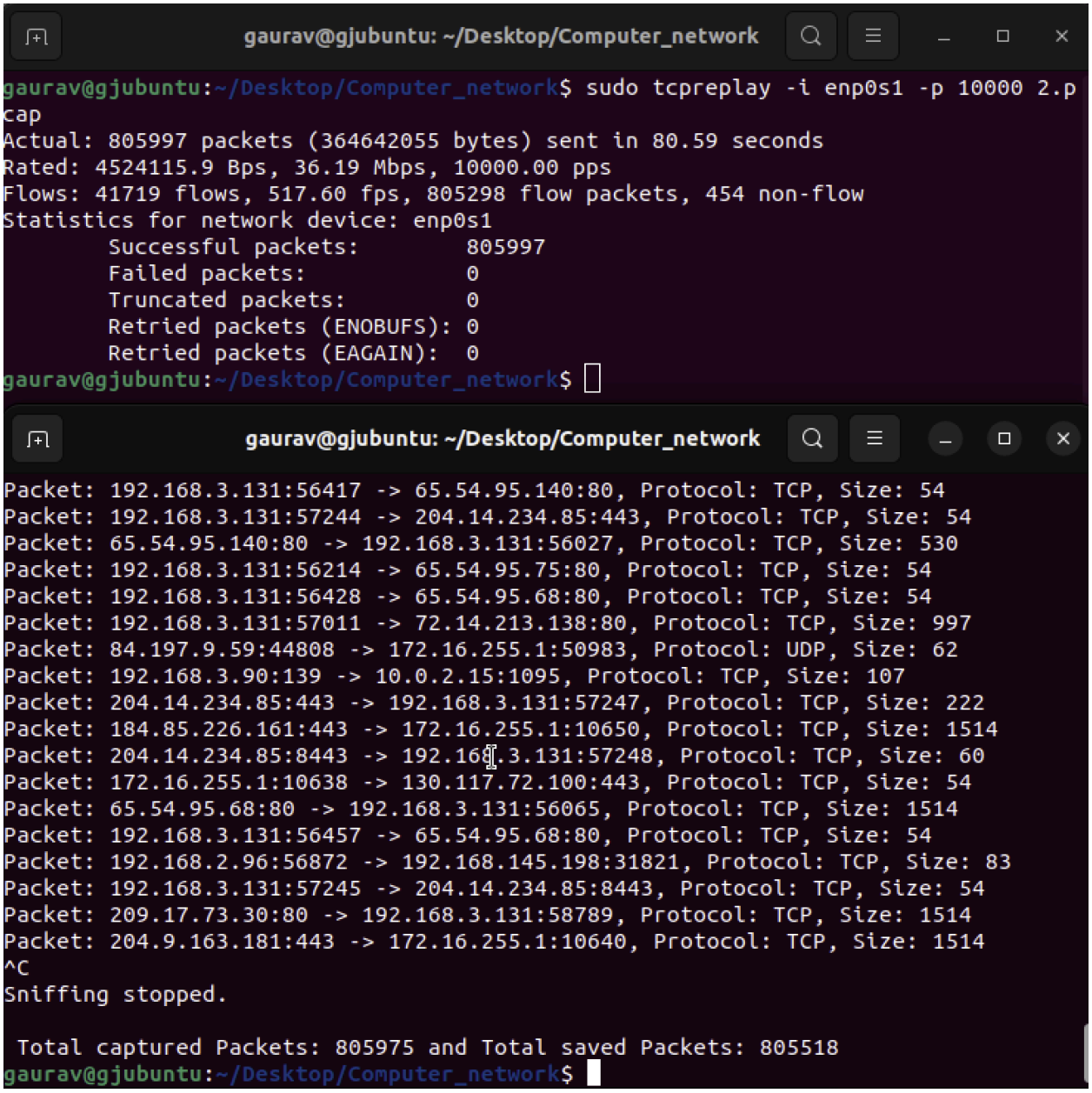
100 Mbps (This has a significant loss)



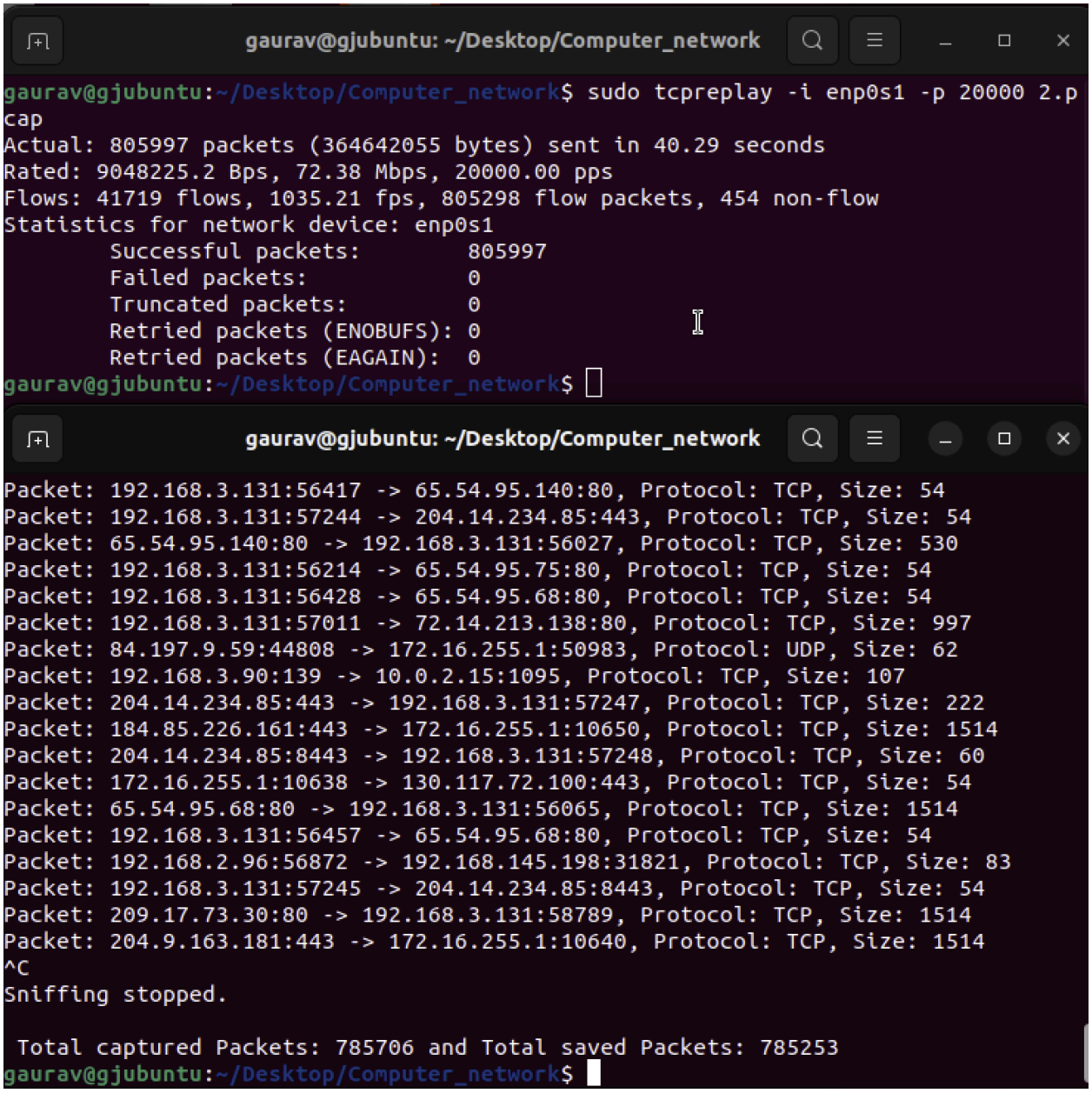
Hence, the TOP SPEED = 60 Mbps (over that, there is some loss)

Using packet rate:-
sudo tcpreplay -i enp0s1 -p 1000 2.pca (for 1000 packets per second)

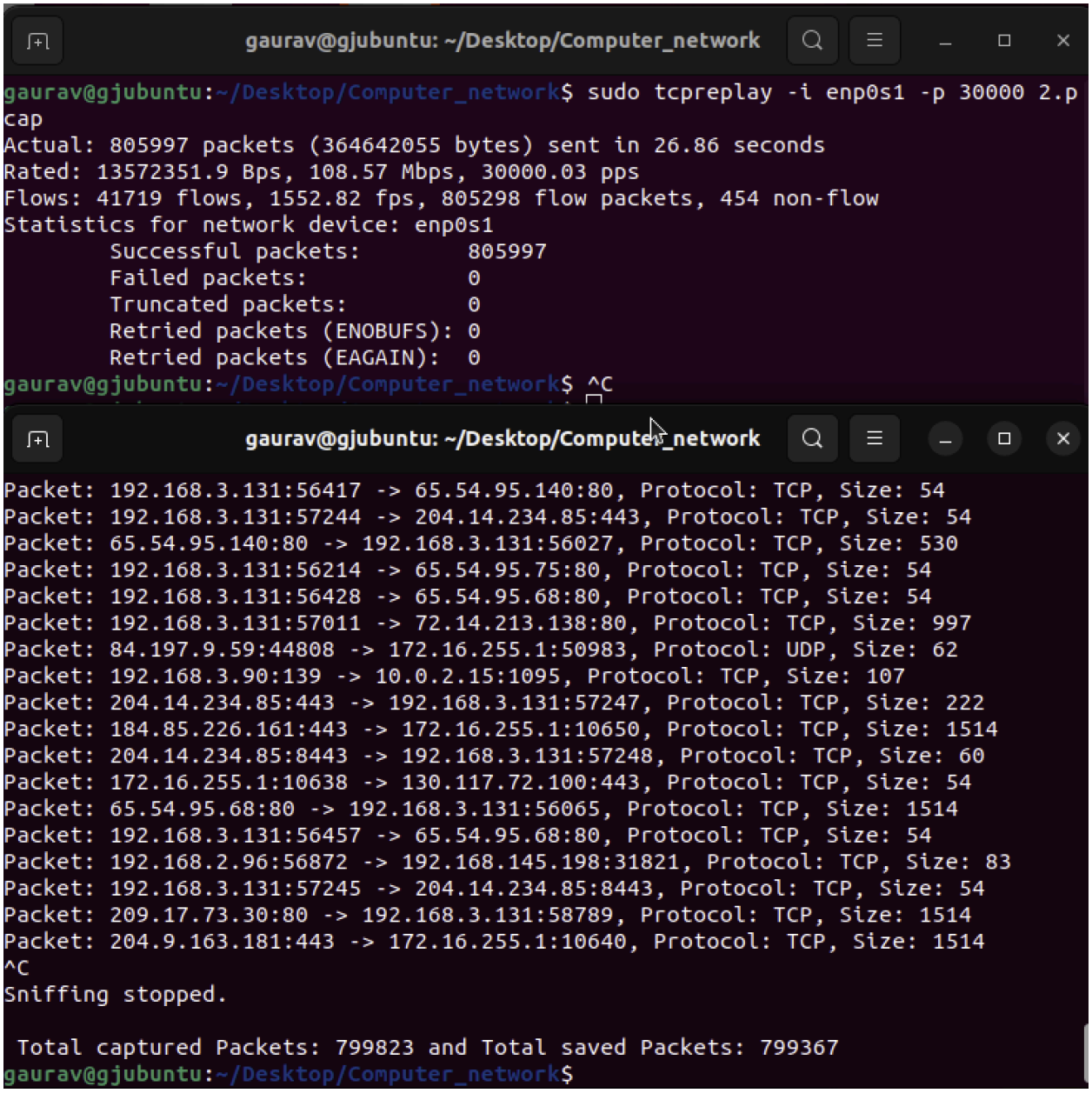
10000 Packets per second (Collects all packets)



20000 Packets per second (Collects all packets, but has some losses)



30000 Packets per second (Collects all packets, but has some losses)



Hence, the TOP SPEED = 10000 Pps (over that, there is some loss)

ii)

Part-2

Sniffing Specific Packet Information

Our script captures packets using a raw socket and processes them to answer specific questions regarding a network traffic capture. It looks for specific information in TCP packets, including an IP address, laptop name, checksum, and specific messages. The script answers the following questions:

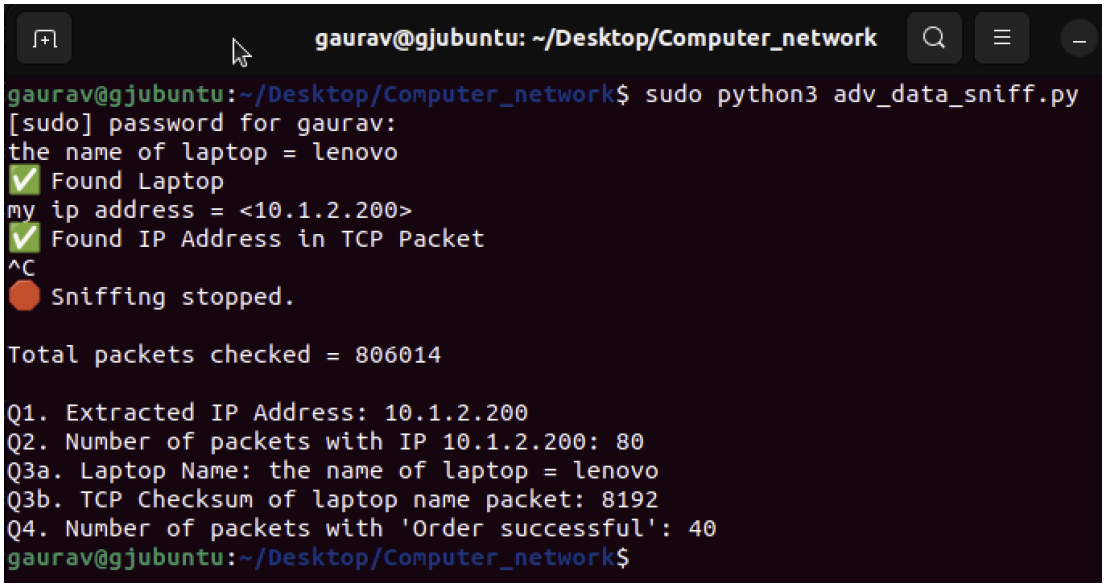
1. Find the IP Address: Searches for a message containing "my ip address = " in the payload of TCP packets.
2. Count Packets with IP: Counts the number of packets where the captured IP address appears either as a source or destination.
3. Laptop Name and TCP Checksum: Extracts the laptop name and its corresponding TCP checksum from packets.
4. Order Successful Messages: Counts number packets contain the phrase "Order successful".

Steps to Run:

1. Set the variable found_ip to the IP address you are looking for and run the script with the following command:
   “sudo python3 sniff_packets.py”.
   - Note: Here, we first ran the code to get the ip, and then saved it. We had to do this as the ip address was being used before the its location.
2. The script listens for packets and checks for the specified conditions.
3. Once you stop the script (with Ctrl+C), it will print the results for all the questions:
   - The extracted IP address.
   - The number of packets with the given IP.
   - The laptop name and TCP checksum.
   - The count of packets containing "Order successful".

The answers for the questions are:-

1. 10.1.2.200
2. 80
3. a. lenovo , b, 8192
4. 40


gaurav@gjubuntu: ~/Desktop/Computer_network
gaurav@gjubuntu:~/Desktop/Computer_network$ sudo python3 adv_data_sniff.py
[sudo] password for gaurav:
the name of laptop = lenovo
✅ Found Laptop
my ip address = <10.1.2.200>
✅ Found IP Address in TCP Packet
^C
🛑 Sniffing stopped.
Total packets checked = 806014
Q1. Extracted IP Address: 10.1.2.200
Q2. Number of packets with IP 10.1.2.200: 80
Q3a. Laptop Name: the name of laptop = lenovo
Q3b. TCP Checksum of laptop name packet: 8192
Q4. Number of packets with 'Order successful': 40
gaurav@gjubuntu:~/Desktop/Computer_network$

Part-3

“”

Part 3: Capture the packets (20 points)

1) Run the Wireshark tool and capture the trace of the network packets on your host device. We expect you would be connected to the Internet and perform regular network activities.
    a. List at-least 5 different application layer protocols that we have not discussed so far in the classroom and describe in 1-2 sentences the operation/usage of protocol and its layer of operation and indicate the associated RFC number if any.

1) Analyze the following details by visiting the following websites in your favourite browser.
    i) canarabank.in
    ii) github.com
    iii) netflix.com
    a. Identify request line with the version of the application layer protocol and the IP address. Also, identify whether the connection(s) is/are persistent or not.
    b. For any one of the websites, list any three header field names and corresponding values in the request and response message. Any three HTTP error codes obtained while loading one of the pages with a brief description.
    c. Capture the Performance metrics that your browser records when a page is loaded and also report the list the cookies used and the associated flags in the request and response headers. Please report the browser name and screenshot of the performance metrics reported for any one of the page loads.

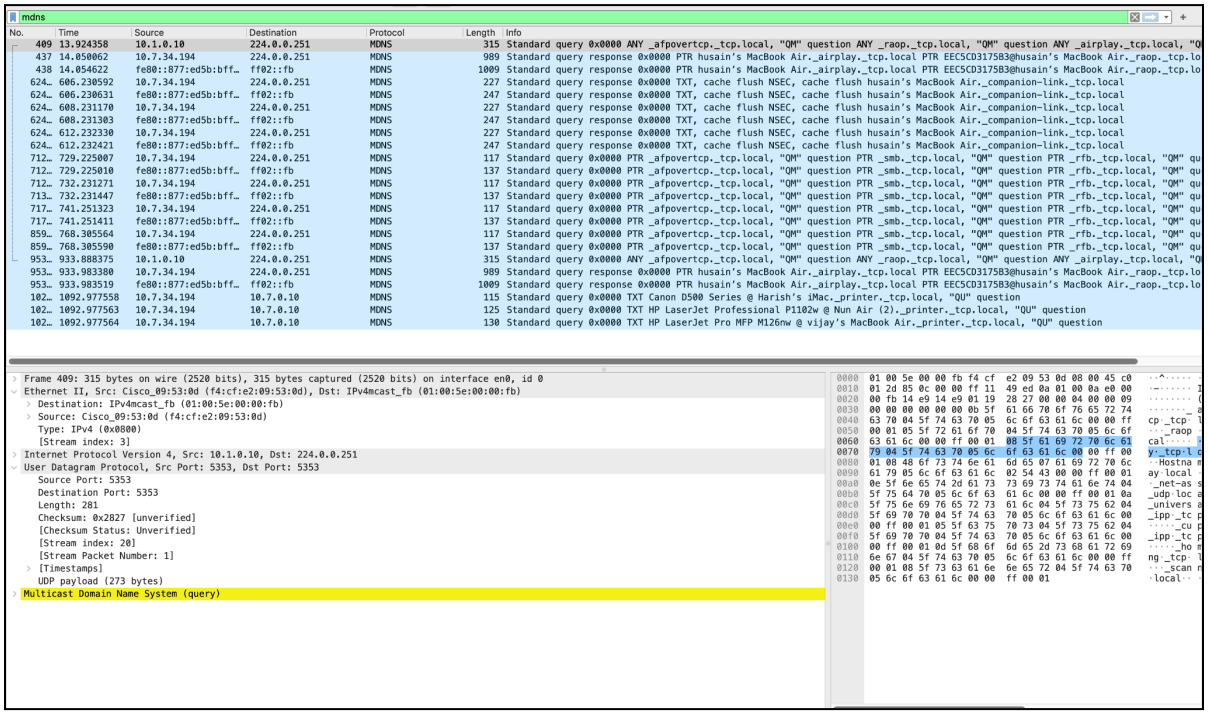
“”

Wireshark Experiments

Q 1.a) Application Layer Protocols Captured Using Wireshark:

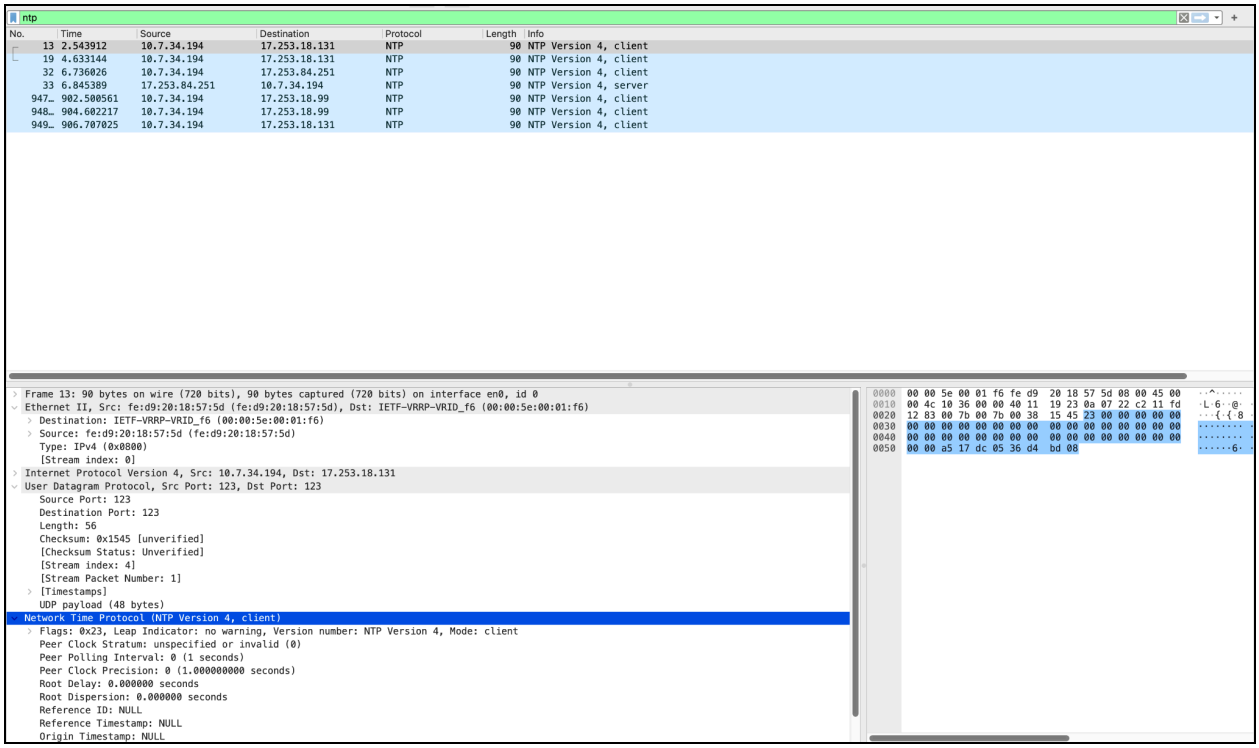
1. **Multicast Domain Name System (mDNS)**
    - mDNS is a zero-configuration networking protocol that enables devices to resolve hostnames to IP addresses without relying on a central DNS server.
    - It is used for device discovery in local networks and is commonly found in applications such as Apple AirPlay, Google Chromecast, and various smart home and IoT devices.
    - mDNS operates at the Application Layer and uses UDP (port 5353), meaning data is sent without connection setup, making it fast but unreliable.

RFC: https://datatracker.ietf.org/doc/html/rfc6762



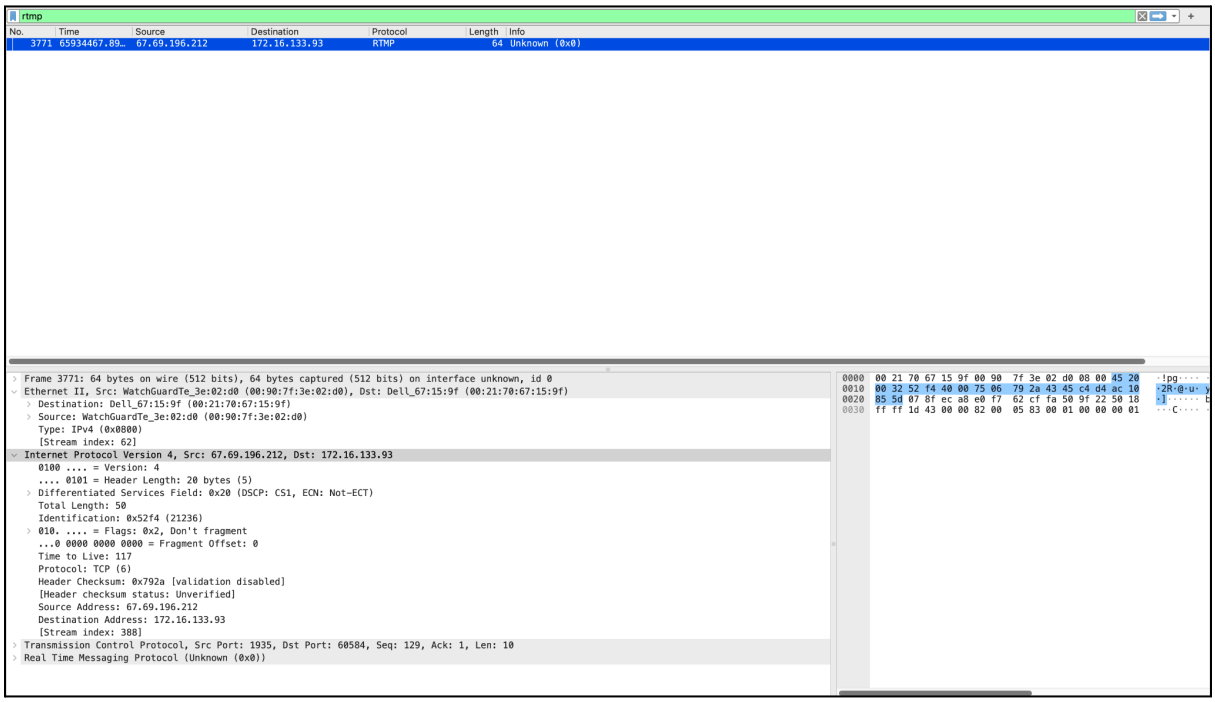
2. **Network Time Protocol (NTP)**
    - NTP protocol is used to synchronize the clocks of computers and network devices over a network.
    - It operates using UDP (port 123) for its communication.

RFC: https://datatracker.ietf.org/doc/html/rfc5905

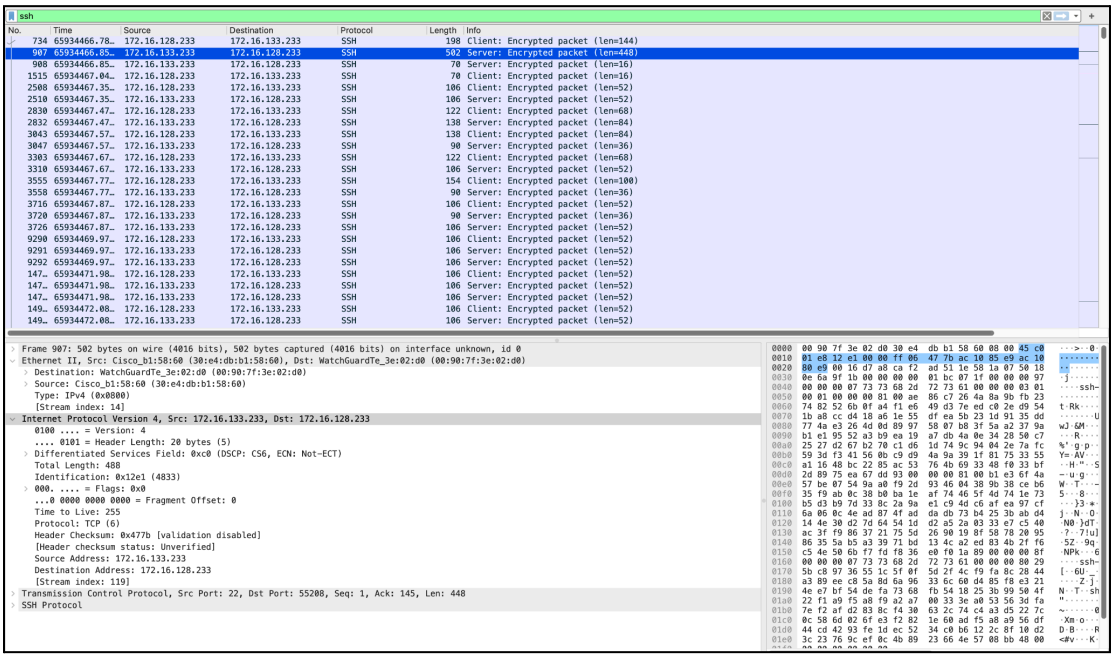


3. Real-Time Messaging Protocol (RTMP)

- RTMP is used for delivering streaming audio, video, and data to a Flash player, commonly for live streaming media applications. https://datatracker.ietf.org/doc/html/rfc7826

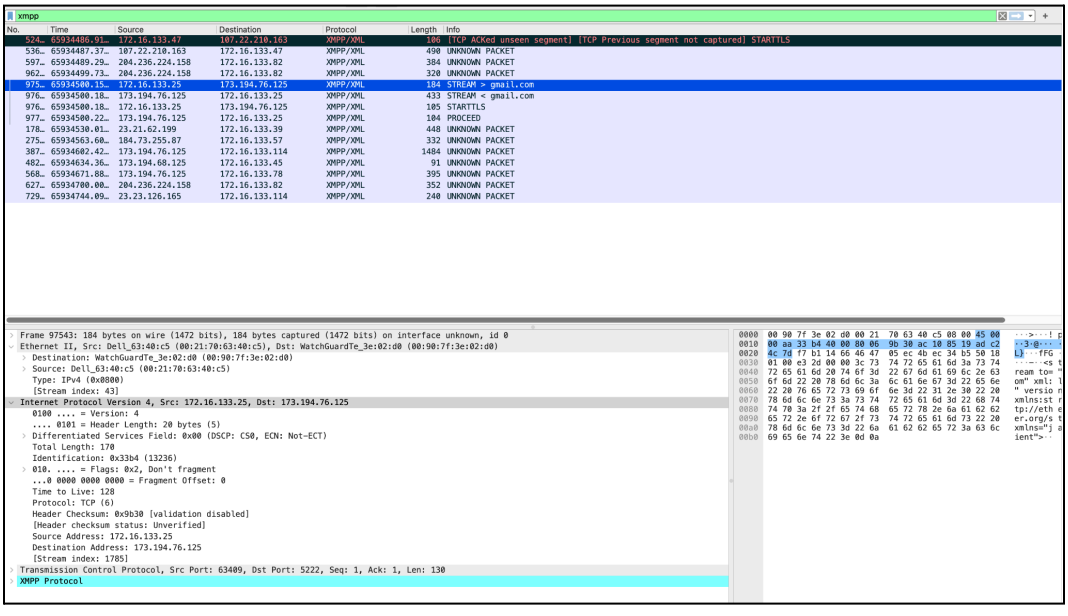


4. **SSH Protocol (Secure Shell)**
    - SSH is a cryptographic network protocol that allows secure remote login and command execution over an unsecured network, providing encryption and authentication. https://datatracker.ietf.org/doc/html/rfc4253

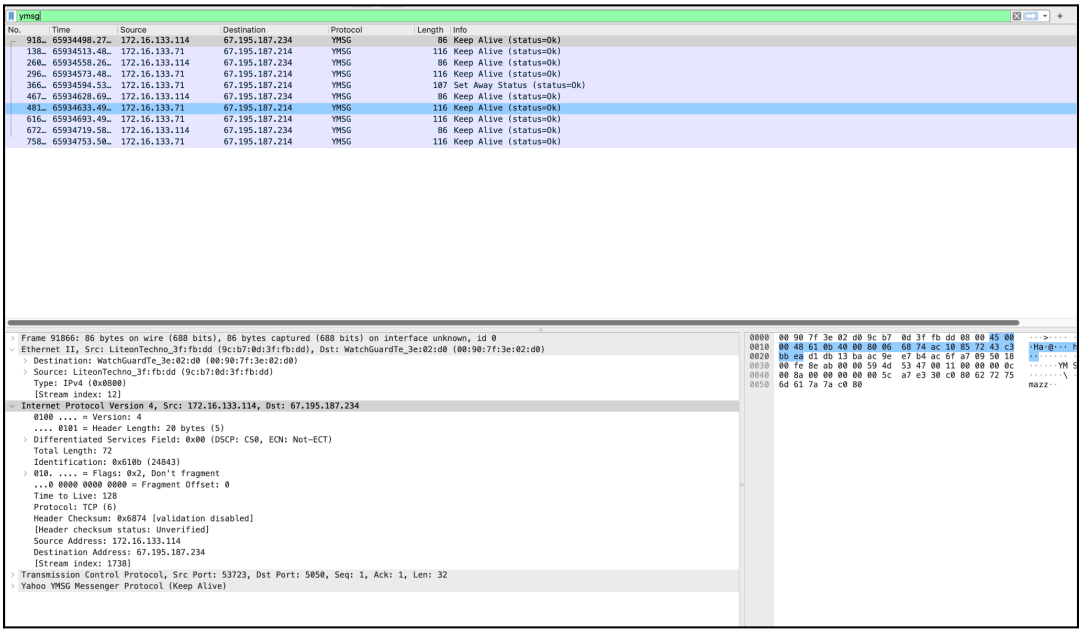


5. **XMPP Protocol (Extensible Messaging and Presence Protocol)**
    - XMPP is a communication protocol for instant messaging, presence information, and multi-party chat, designed to be extensible and decentralized.

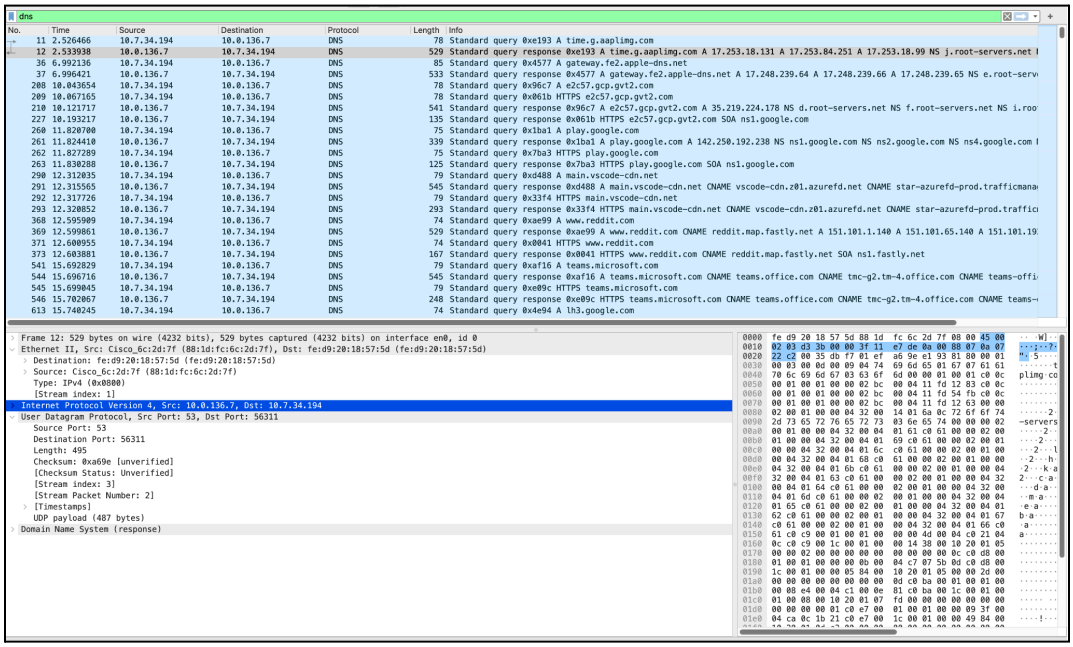
RFC: https://datatracker.ietf.org/doc/html/rfc6120



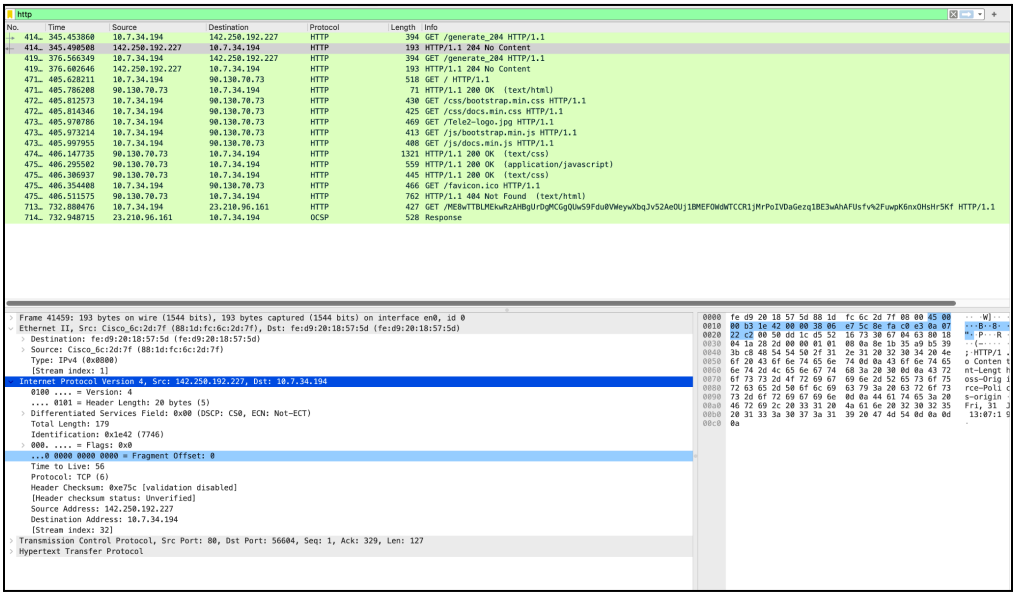
6. **Yahoo YMSG Messenger Protocol**
    - YMSG is the messaging protocol used by Yahoo Messenger for sending instant messages and supporting features like file transfers, presence, and notifications.



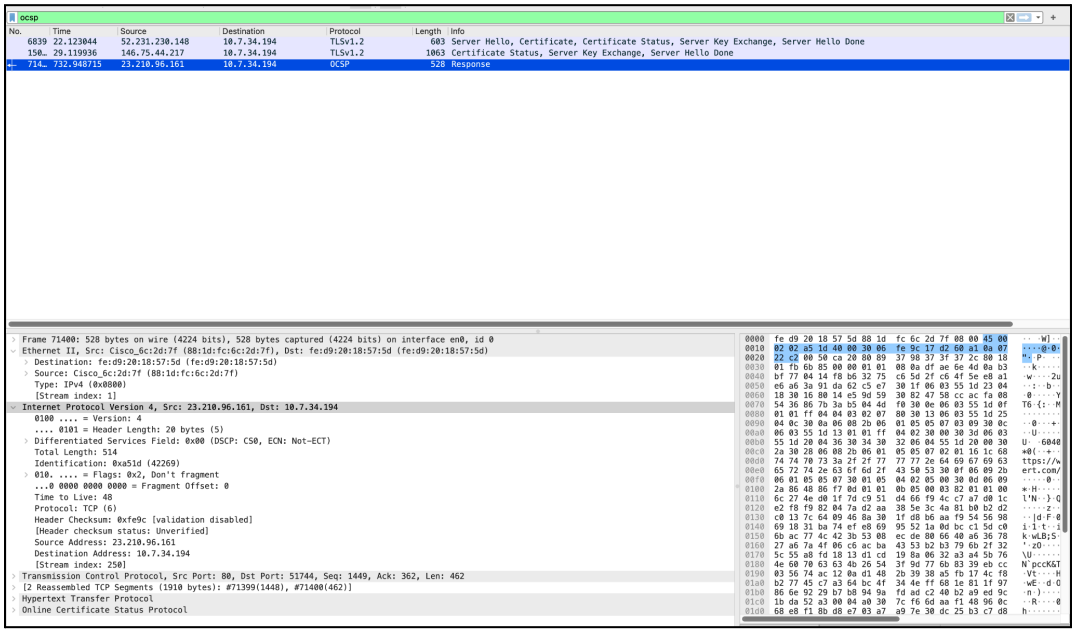
7. **Domain Name System (DNS)**
    DNS is a protocol that translates human-readable domain names into IP addresses, allowing browsers to locate web servers. It operates over TCP/UDP (port 53) and is essential for the functioning of the internet.
    RFC: https://www.ietf.org/rfc/rfc1035.txt



8. **Hypertext Transfer Protocol (HTTP)**
   HTTP is the foundational protocol for transferring data across the web. It defines how messages are formatted and transmitted, operating over TCP (port 80) and often using encryption (HTTPS) over port 443.
   RFC: <from slides>



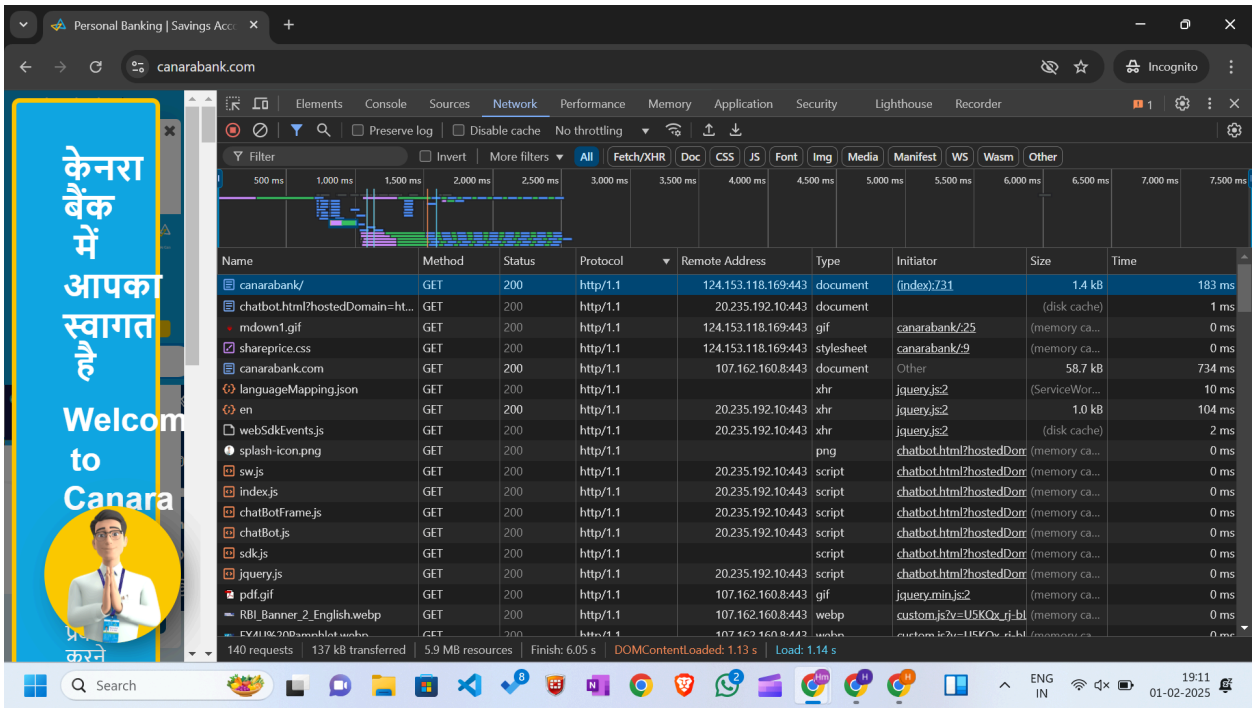
9. **Online Certificate Status Protocol (OCSP)**
   OCSP is a protocol used to check the revocation status of an X.509 digital certificate, providing real-time verification to enhance the security of communications. It operates over HTTP (port 80).
   RFC: 6960



Q 2)

i. canarabank.in

a. Request Line and Connection Information:



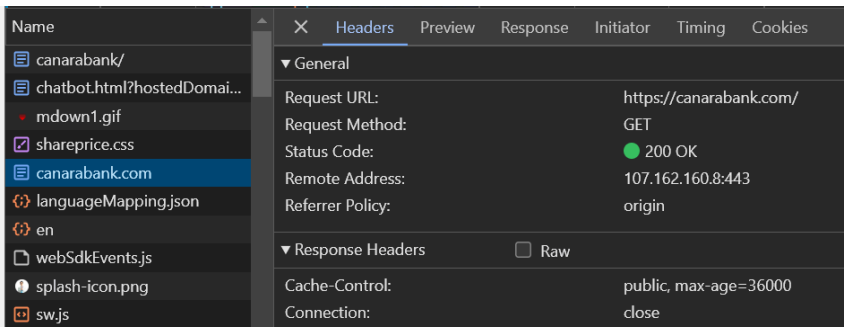
1. **Request Line:**
   - GET / HTTP/1.1
   - **Version of Application Layer Protocol**: HTTP/1.1
   - **IP Address**: 107.162.160.8

```
PS C:\Users\3malw> ping canarabank.com

Pinging canarabank.com [107.162.160.8] with 32 bytes of data:
Reply from 107.162.160.8: bytes=32 time=72ms TTL=241
Reply from 107.162.160.8: bytes=32 time=74ms TTL=241
Reply from 107.162.160.8: bytes=32 time=71ms TTL=241
Reply from 107.162.160.8: bytes=32 time=72ms TTL=241

Ping statistics for 107.162.160.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 71ms, Maximum = 74ms, Average = 72ms
PS C:\Users\3malw>
```

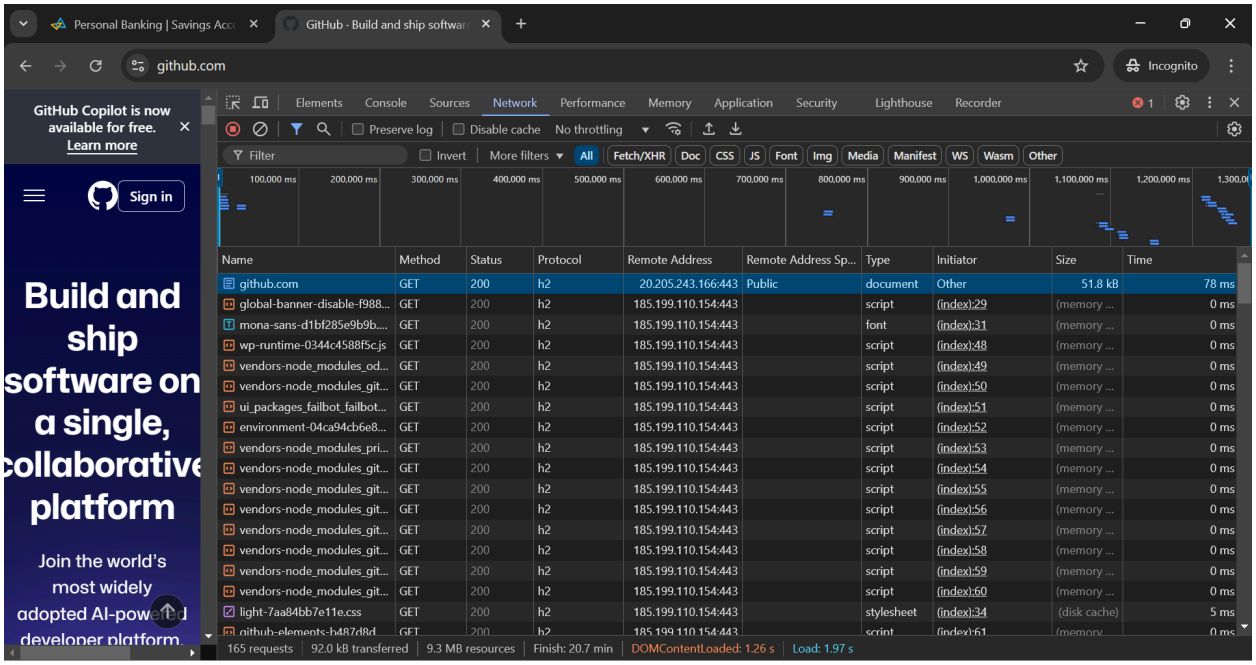
2. **Connection Type:**
    - **Persistent Connection**: NO



[**Connection: keep-alive** indicates a persistent connection, while **Connection: close** indicates a non-persistent connection]

ii. github.com

a. Request Line and Connection Information:



1. **Request Line:**
    - GET / HTTP/2
    - **Version of Application Layer Protocol**: HTTP/2
    - **IP Address**: 20.207.73.82

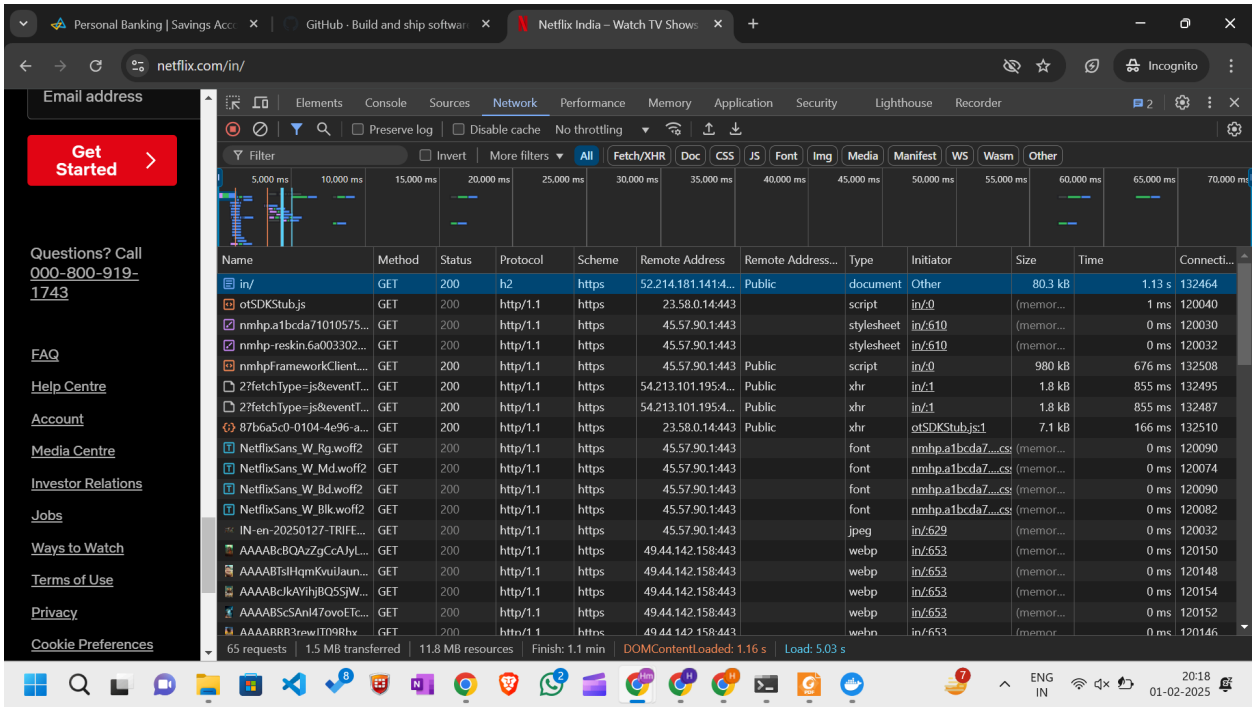
```
Pinging github.com [20.207.73.82] with 32 bytes of data:
Reply from 20.207.73.82: bytes=32 time=16ms TTL=112
Reply from 20.207.73.82: bytes=32 time=17ms TTL=112
Reply from 20.207.73.82: bytes=32 time=39ms TTL=112
Reply from 20.207.73.82: bytes=32 time=15ms TTL=112

Ping statistics for 20.207.73.82:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 15ms, Maximum = 39ms, Average = 21ms
PS C:\Users\3malw>
```

2. **Connection Type:**
    - **Persistent Connection**: NOT SPECIFIED

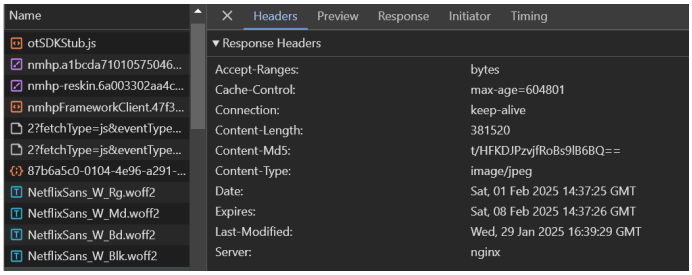
iii. netflix.com

a. Request Line and Connection Information:



1. **Request Line:**
    - GET / HTTP/2
    - **Version of Application Layer Protocol**: HTTP/1.1 & HTTP/2 & HTTP/3
    - **IP Address**: 52.214.181.141

2. **Connection Type:**
    - **Persistent Connection**: Yes



---

b. Request and Response Headers, and HTTP Error Codes:

Request Header Fields
accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 **accept-encoding:** gzip, deflate, br, zstd **accept-language:** en-US,en;q=0.9 **connection:** keep-alive **host:** canarabank.com

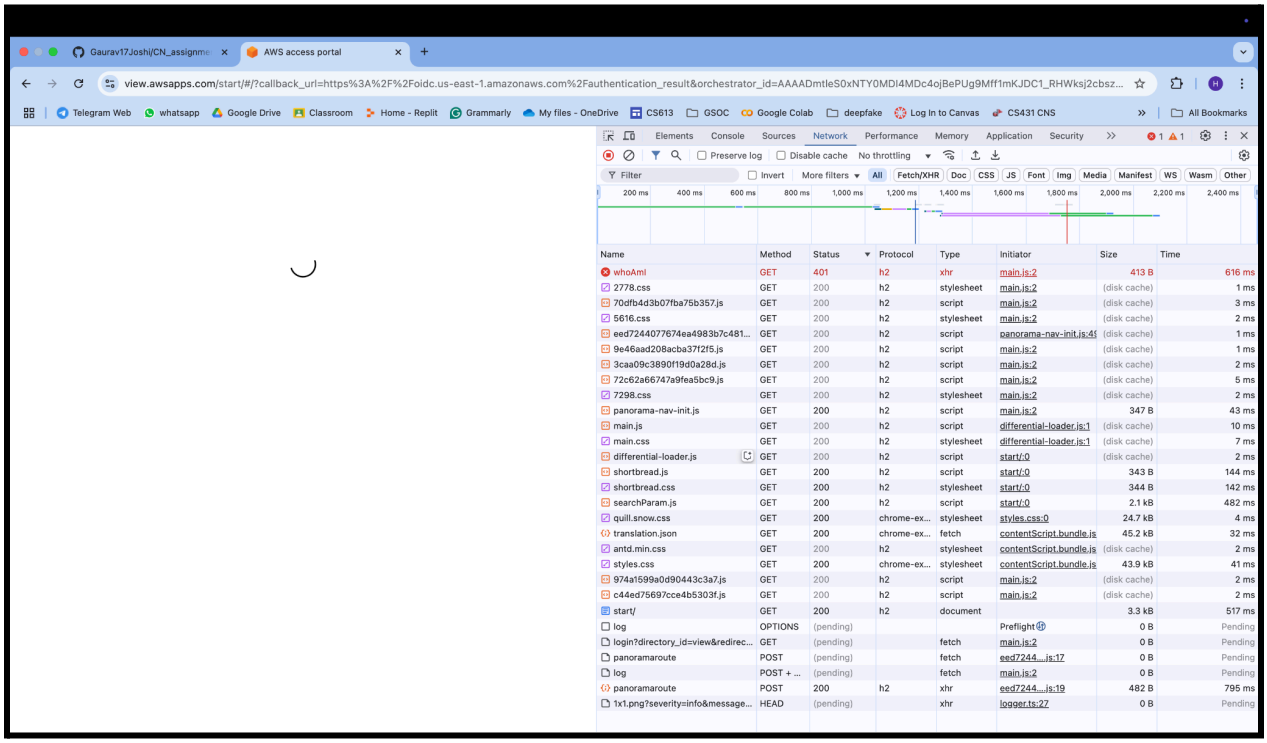
Response Header Fields
cache-control: public, max-age=36000 connection: close

content-type: text/html; charset=utf-8

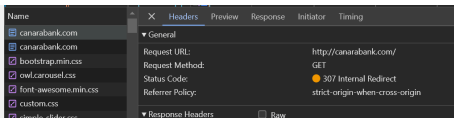
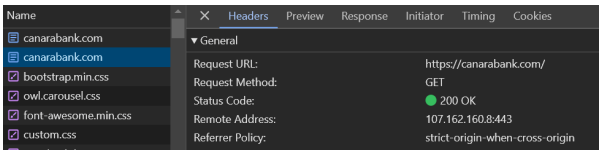
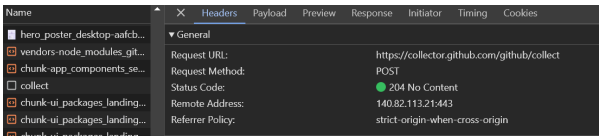
date: Sat, 01 Feb 2025 15:06:44 GMT

referrer-policy: no-referrer-when-downgrade

Error codes:

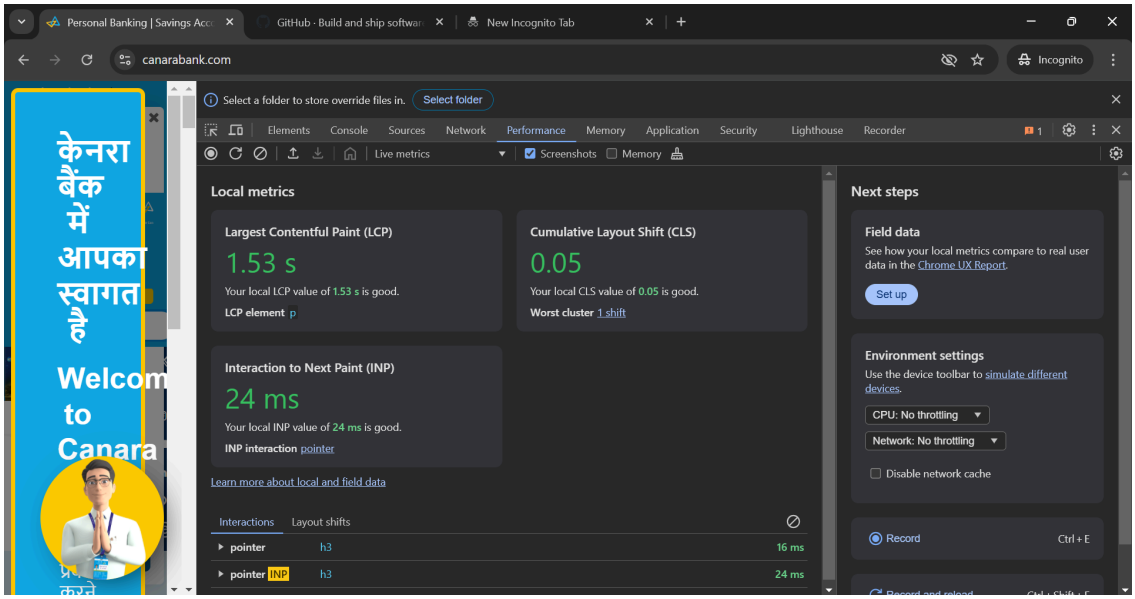


HTTP Error Codes	Description
401 Unauthorized	It occurs when authentication is required but has either failed or has not been provided. This typically happens when accessing protected resources without valid credentials.

307 	Temporary Redirect in HTTP status codes means that the requested resource has been temporarily moved to a different URL.
200 OK 	The request has been fulfilled, and the server has returned the requested data.
204 NO CONTENT 	No Content response in HTTP status codes indicates that the server has successfully processed the request, but there is no content to return in the response body.

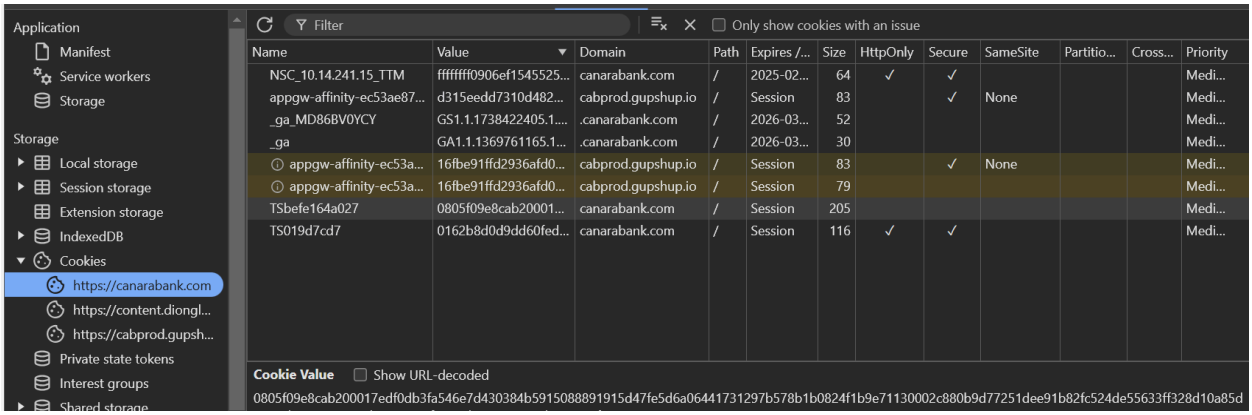
c. Performance Metrics and Cookies:

1. **Performance Metrics**:



- **Browser Name**: Google Chrome
- **Website:**
- **Metrics Captured**:
  - **Largest Contentful Paint (LCP)**: 1.53 s
  - **Cumulative Layout Shift (CLS)**: 0.05
  - **Interaction to Next Paint (INP)**: 24ms

2. **Cookies Used**:
   - **Website:** canarabank.com



Cookie Name	Value	flag
TS019d7cd7	0162b8d0d9dd60fed8b8046afbedbd7757d8125dcbe2ea466c47dcf9e418b46f58b799f744cf4dbb65896a877cce75396b22679df1	Secure, HTTPOnly
TSbefe164a027	0805f09e8cab200017edf0db3fa546e7d430384b5915088891915d47fe5d6a06441731297b578b1b0824f1b9e71130002c880b9d77251dee91b82fc524de55633ff328d10a85dc50e3b8cc17a8679d63791328f073cebe78178463d4c305a4f4	
NSC_10.14.241.15_TTM	ffffffff0906ef1545525d5f4f58455e445a4a4216cb	Secure, HTTPOnly
_ga_MD86BV0YCY	GS1.1.1738422405.1.1.1738423752.53.0.0	
_ga	GA1.1.1369761165.1738422405	