Computer Networks Assignment-1

1.02.2024

Birudugadda Srivibhav (22110050)

Srivathsa Vamsi Chaturvedula (22110260)

Computer Science and Engineering

IIT Gandhinagar

Introduction

We have created a packet sniffer using the Python `socket` library, `dpkt`, and other utilities. The sniffer captures network packets, analyzes traffic, and provides insights into network activity. Github_link for sniffer code: https://github.com/Sparky1743/CN-Assignments

Part-1

In this section, we analyze the data captured during the replay of a PCAP file using `tcpreplay` and a custom Python sniffer. We executed the experiment on two different Ubuntu Linux versions (20.04 and 24.04) and ensured that both Wi-Fi and Bluetooth were disabled during the data transfer.

Experimental Setup

- **Sender (Ubuntu 20.04)**: Used `tcpreplay` to send the PCAP file (`2.pcap`).
- **Receiver (Ubuntu 24.04)**: Used our Python sniffer program to capture the traffic.
- **Network Interface**: Ethernet (`eth0`).

Commands Used:

- **Sender**:

```
birud_ubuntu_20.04@chinnu:~$ sudo tcpreplay --pps=10000 -i eth0 2.pcap
```

- **Receiver**:

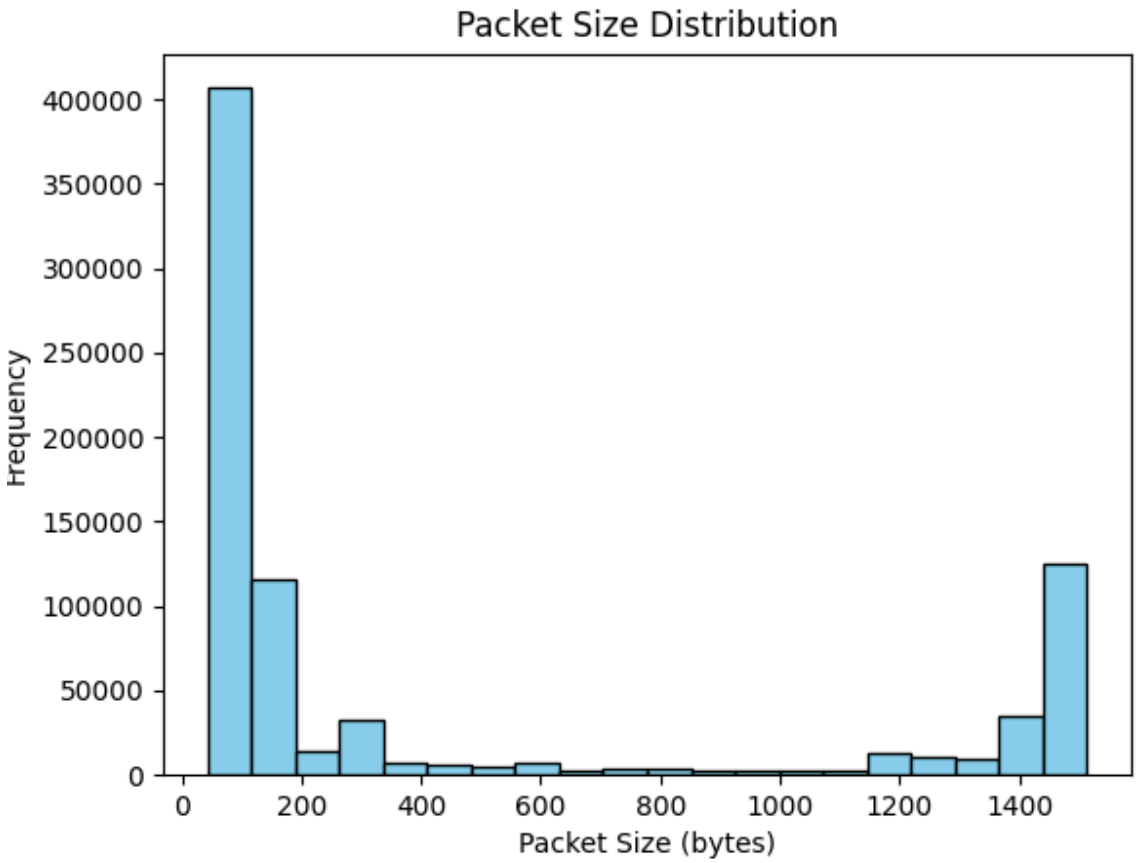
```
(base) birud_ubuntu_24.04@chinnu:~/CN/Computer-Networks/Packet-Sniffer$ sudo /home/birud_ubuntu/miniconda3/envs/cn/bin/python3 python_sniffer.py -i eth0 -d 10000 -p capture.pcap
```

1) Original 2.pcap analysis

In this section, we analyze the statistics in the original `2.pcap` file and then compare these statistics with the results obtained using our sniffer code.

```
Total Data: 364641996 bytes
Total Packets: 805997
Min Packet Size: 42 bytes
Max Packet Size: 1514 bytes
Average Packet Size: 452.41 bytes
```

Histogram of packet size distribution



```
Unique Source-Destination Pairs: 41899
('96.43.146.22:443', '172.16.133.29:60614')
('172.16.133.116:53504', '172.16.139.250:5440')
('172.16.133.84:58955', '172.16.139.250:5440')
('96.43.146.22:443', '172.16.133.63:54135')
('8.8.8.8:53', '172.16.133.6:63245')
('172.16.133.67:55957', '172.16.139.250:5440')
('172.16.128.169:4449', '172.16.133.248:161')
('172.16.128.169:4400', '172.16.133.242:161')
('172.16.133.41:52978', '172.16.139.250:5440')
('66.220.149.32:80', '192.168.3.131:56048')
```

The complete data of unique source-destination pairs for the original 2.pcap file can be found in `Original_2.pcap_statistics/part1_step2.txt` in the GitHub repository.

```
Destination IP -> Total Flows:
192.168.3.131: 6184
10.0.2.15: 808
207.46.0.109: 29
65.54.95.68: 664
172.16.255.1: 1219
204.14.234.85: 740
65.54.95.140: 495
172.16.0.1: 10
147.31.122.1: 90
66.235.139.121: 39
109.227.83.224: 36
```

```
Source IP -> Total Flows:
65.54.95.68: 1275
65.54.95.75: 766
65.54.95.140: 658
204.14.234.85: 1036
65.54.186.19: 66
192.168.3.131: 4294
72.14.213.105: 6
65.54.189.173: 126
207.46.0.109: 63
184.85.226.161: 222
```

The complete data of the total flows can be found in `Original_2.pcap_statistics/part1_step3.txt` in the GitHub repository.

```
Source-Destination Pair Transferring the Most Data:
('172.16.133.95:49358', '157.56.240.102:443'): 17342229 bytes
```

2) Captured data analysis

In our approach, we analyze the captured network traffic in real time while simultaneously storing the captured packets in `capture.pcap` for further analysis in Part 2. This enables us to monitor and process network statistics dynamically as the program is running, ensuring immediate insights into data transfer patterns.

Once the user decides to stop sniffing, they need to press `Ctrl+C`. At this point, the program prints the final statistics and transitions into analysis mode, providing the following options:

- Press **'a'** to analyze packets.
- Press **'h'** to generate a histogram.
- Press **'u'** to display unique source-destination pairs.
- Press **'m'** to analyze flows.
- Press **'q'** to quit.

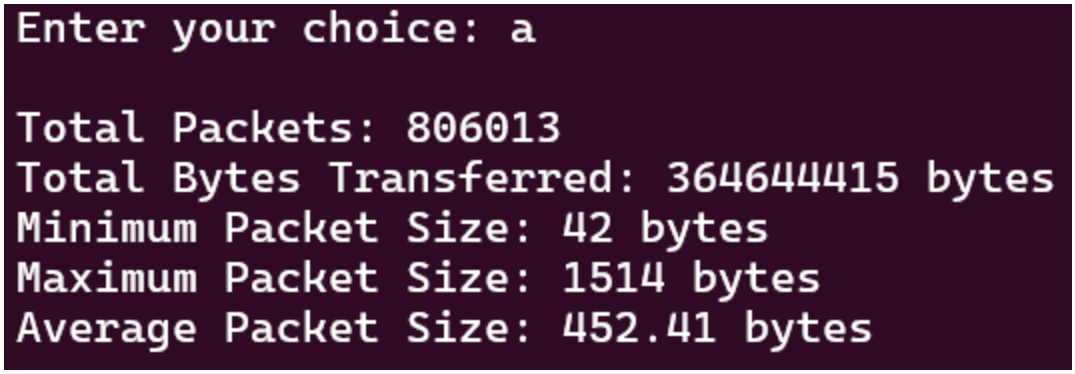
This interactive approach ensures that users can further investigate captured data efficiently while maintaining real-time monitoring capabilities.

```
birud_ubuntu_20.04@chinnu:~$ sudo tcpreplay --pps=10000 -i eth0 2.pcap
Actual: 805997 packets (364642055 bytes) sent in 80.59 seconds
Rated: 4524116.1 Bps, 36.19 Mbps, 10000.00 pps
Statistics for network device: eth0
        Successful packets:        805997
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

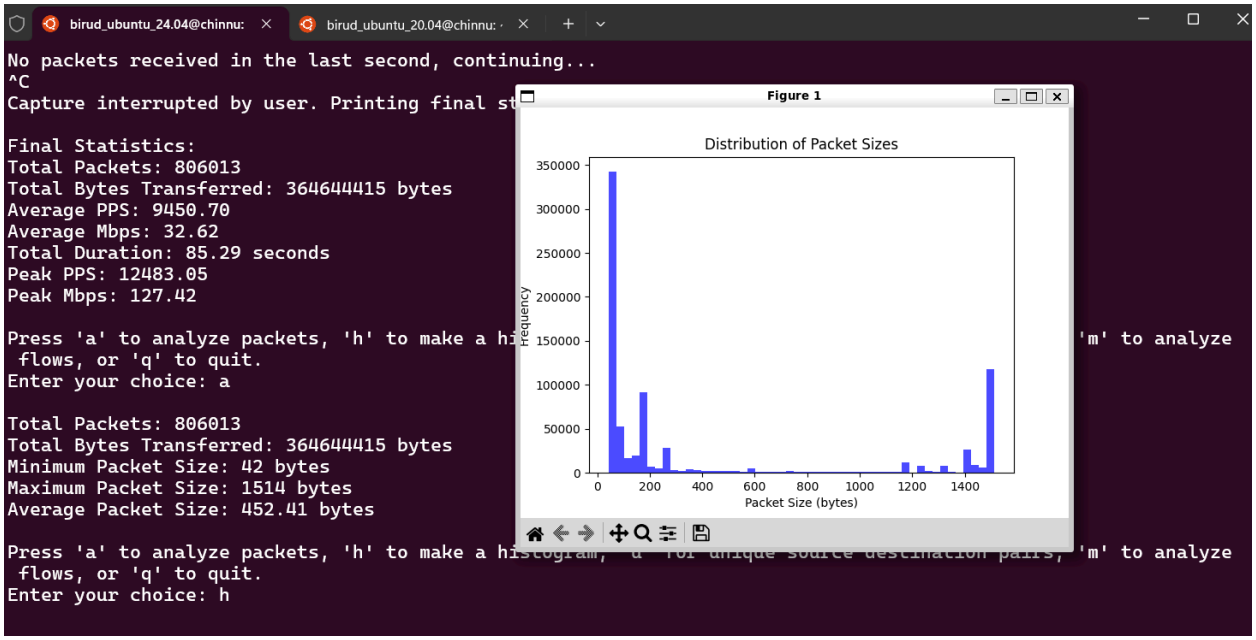
```
Current PPS: 10000.07, Mbps: 34.10
Current PPS: 10000.07, Mbps: 34.05
Current PPS: 10000.07, Mbps: 34.25
Current PPS: 10000.22, Mbps: 34.40
No packets received in the last second, continuing...
No packets received in the last second, continuing...
No packets received in the last second, continuing...
No packets received in the last second, continuing...
^C
Capture interrupted by user. Printing final statistics...

Final Statistics:
Total Packets: 806013
Total Bytes Transferred: 364644415 bytes
Average PPS: 9450.70
Average Mbps: 32.62
Total Duration: 85.29 seconds
Peak PPS: 12483.05
Peak Mbps: 127.42

Press 'a' to analyze packets, 'h' to make a histogram, 'u' for unique source-destination pairs, 'm' to analyze
 flows, or 'q' to quit.
Enter your choice:
```

In the above image, the sender has completed sending the packets in 80.59 seconds, so there are no more packets to receive, so the user has pressed `Ctrl+C` to analyze further the captured packets.

- **Analyzing captured packets**



- **Plotting Histogram**



- **Unique source-destination pairs**

The unique source-destination pairs will be written to a text file and saved to disk, which can be found in the `Packet-Sniffer/unique_pairs.txt` path of the GitHub repository.

```
Unique Source-Destination Pairs: 41900
('8.8.4.4:53', '172.16.133.6:63104')
('172.16.133.55:57701', '172.16.139.250:5440')
('96.43.146.48:443', '172.16.133.37:60769')
('173.194.43.38:80', '172.16.133.115:53733')
('172.16.133.28:65345', '172.16.139.250:5440')
('172.16.133.43:57817', '172.16.139.250:5440')
('172.16.133.26:53121', '96.43.146.176:443')
('180.153.31.250:443', '172.16.133.40:50296')
('172.16.133.116:53802', '172.16.139.250:5440')
('172.16.133.20:54795', '172.16.128.202:53')
('172.16.133.78:59066', '172.16.139.250:5440')
('172.16.133.92:57645', '172.16.139.250:5440')
```

- **Analyze flows**

The complete total flows data will also be written to disk as a text file, which can be found in the `Packet-Sniffer/total_flows.txt` path of the GitHub repository.

```
Destination IP -> Total Flows:
192.168.3.131:56368: 124
192.168.3.131:56427: 90
192.168.3.131:56132: 29
192.168.3.131:56093: 23
192.168.3.131:56233: 28
192.168.3.131:57244: 242
10.0.2.15:2526: 66
207.46.0.109:80: 29
192.168.3.131:57721: 4
65.54.95.68:80: 664
```

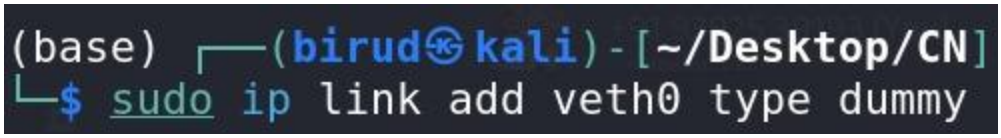
```
Source IP -> Total Flows:
65.54.95.68:80: 1275
65.54.95.75:80: 766
65.54.95.140:80: 658
204.14.234.85:8443: 516
65.54.186.19:5443: 33
192.168.3.131:56320: 15
72.14.213.105:443: 4
192.168.3.131:56511: 56
65.54.189.173:61863: 63
207.46.0.109:80: 63
184.85.226.161:443: 222
204.14.234.85:443: 516
```

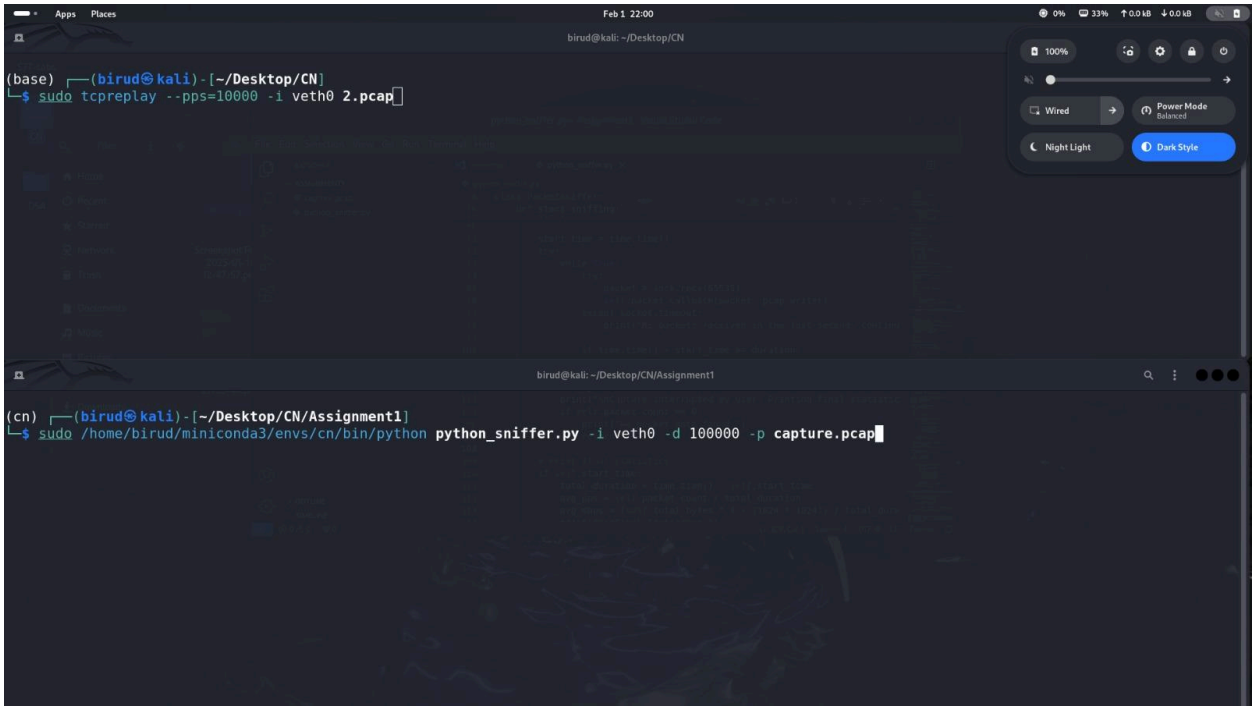
```
Source-Destination Pair Transferring the Most Data:
('172.16.133.95:49358', '157.56.240.102:443'): 17342229 bytes
```

- **Running both tcpreplay and our program on the same machine**

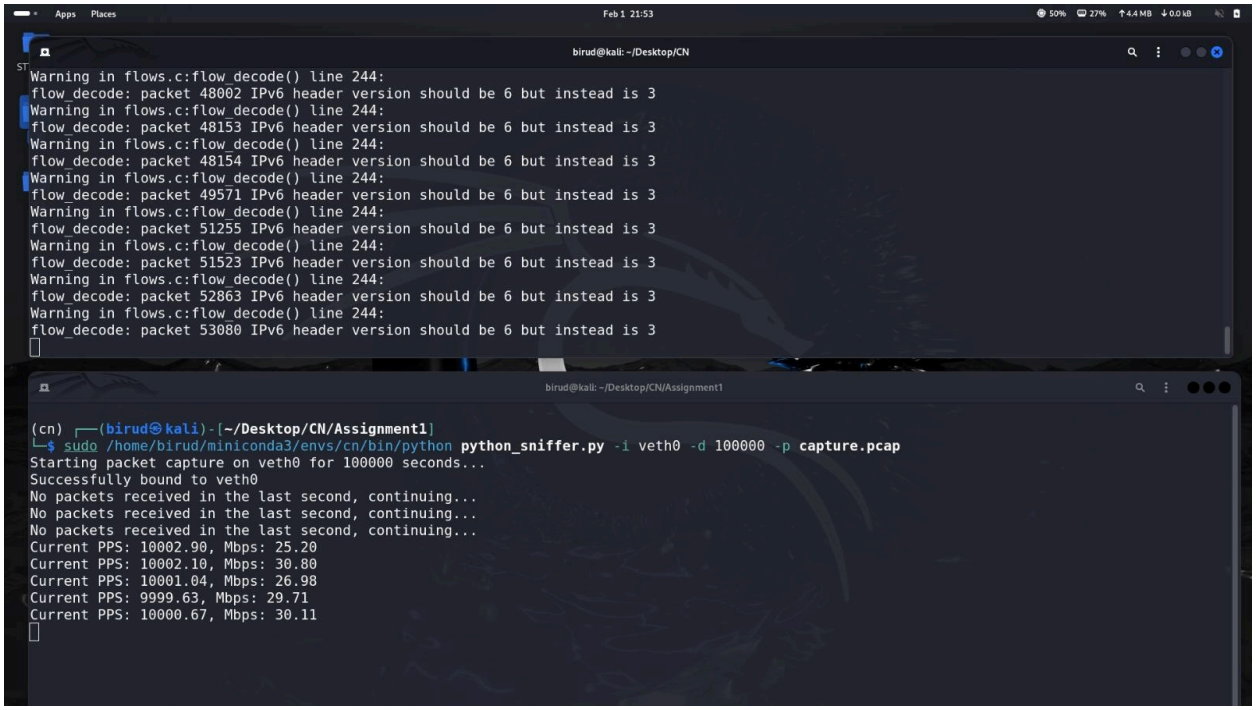
For this process, we are using a Kali Linux virtual machine (VM). To ensure an isolated testing environment, we have disabled all wired connections and created a virtual network interface that is not connected to the internet. This setup allows us to replay network traffic using `tcpreplay` and simultaneously capture and analyze the flows using our program, all within the same machine without external interference.

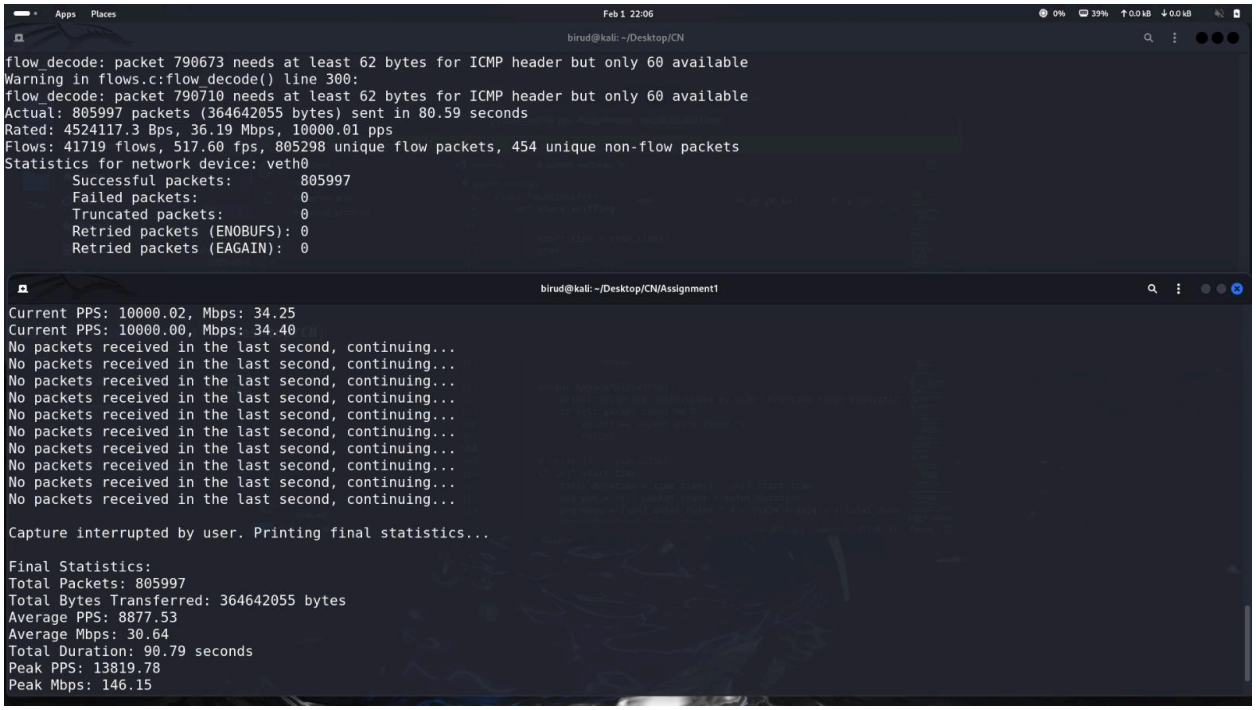
This was done on linux using the following commands





There are two terminals here: the upper one is the sender, and the second one is the receiver.





Clearly, we can see that the number of packets sent is equal to the number of packets received, demonstrating high accuracy even in a VM.

The peak values are unusually high, likely due to calculating the instantaneous rate, where consecutive small packets resulted in a high packets-per-second (pps) value.

- **Running tcpreplay and our program on different machines**

We executed the experiment on two different Ubuntu Linux versions (20.04 and 24.04) and ensured that both Wi-Fi and Bluetooth were disabled during the data transfer.

Experimental Setup

- **Sender (Ubuntu 20.04)**: Used `tcpreplay` to send the PCAP file (`2.pcap`).
- **Receiver (Ubuntu 24.04)**: Used our Python sniffer program to capture the traffic.
- **Network Interface**: Ethernet (`eth0`).

Commands Used:

- Sender:

```
birud_ubuntu_20.04@chinnu:~$ sudo tcpreplay --pps=10000 -i eth0 2.pcap
```

- Receiver:

```
(base) birud_ubuntu_24.04@chinnu:~/CN/Computer-Networks/Packet-Sniffer$ sudo /home/birud_ubuntu/miniconda3/envs/cn/bin/python3 python_sniffer.py -i eth0 -d 10000 -p capture.pcap
```

Results:

- Sender:

```
birud_ubuntu_20.04@chinnu:~$ sudo tcpreplay --pps=10000 -i eth0 2.pcap
Actual: 805997 packets (364642055 bytes) sent in 80.59 seconds
Rated: 4524116.1 Bps, 36.19 Mbps, 10000.00 pps
Statistics for network device: eth0
        Successful packets:        805997
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

- Receiver:

```
Current PPS: 10000.07, Mbps: 34.10
Current PPS: 10000.07, Mbps: 34.05
Current PPS: 10000.07, Mbps: 34.25
Current PPS: 10000.22, Mbps: 34.40
No packets received in the last second, continuing...
No packets received in the last second, continuing...
No packets received in the last second, continuing...
No packets received in the last second, continuing...
^C
Capture interrupted by user. Printing final statistics...

Final Statistics:
Total Packets: 806013
Total Bytes Transferred: 364644415 bytes
Average PPS: 9450.70
Average Mbps: 32.62
Total Duration: 85.29 seconds
Peak PPS: 12483.05
Peak Mbps: 127.42

Press 'a' to analyze packets, 'h' to make a histogram, 'u' for unique source-destination pairs, 'm' to analyze
 flows, or 'q' to quit.
Enter your choice:
```

We have observed that the code is able to receive packets with negligible loss when TCP replay is sending up to 10,000 packets. However, beyond this limit, the code is unable to properly receive packets, resulting in some packet loss.

Part-2

The following are the answers to the CTF questions. We saved the packets received by our sniffer in a new pcap file named ***capture.pcap*** and then used our program to solve the CTF challenges. Additionally, we also cross-checked whether our answers were correct using Linux commands.

```
Q1. My IP address: 10.1.2.200
Q2. Number of packets with IP 10.1.2.200: 80
Q3a. laptop = lenovo
Q3b. TCP checksum of that packet: 2657
Q4. Number of packets with 'Order successful': 40
```

Verification with linux Commands:

1) The following command was used to find “my ip address”

```
(base) ┌──(birud㉿kali)-[~/Desktop/CN/Assignment1]
└─$ strings capture.pcap | grep "10.1.2.200"

My ip address = <10.1.2.200>?M
```

2) Now to find the number of packets with this ip address:

```
(base) ┌──(birud㉿kali)-[~/Desktop/CN/Assignment1]
└─$ tshark -r capture.pcap -Y "ip.addr == 10.1.2.200" | wc -l
80
```

3) Finding the name of the laptop

```
(base) ┌──(birud㉿kali)-[~/Desktop/CN/Assignment1]
└─$ strings capture.pcap | grep "laptop"
ve had great local technical support from HP Korea, and we have seen big improve
ments since using this new solution. With most employees using Wi-Fi phones/lapt
ops/smart phones, we can access patient data much faster and diagnose t
GET /t5/image/serverpage/avatar-name/icon_laptop2/avatar-theme/classic/avatar-co
llection/computers/avatar-display-size/message HTTP/1.1
ve had great local technical support from HP Korea, and we have seen big improve
ments since using this new solution. With most employees using Wi-Fi phones/lapt
ops/smart phones, we can access patient data much faster and diagnose t
GET /resources/images/pages/hero-device-screenshots-laptop.png HTTP/1.1
                        <div class="content-text marginBottom15"><p>The HP BYOD
solution delivers a robust, simple and secure way for your enterprise to allow u
sers to access your network as well as applications from their own laptop, table
t or smartphone.</p></div>
 PC, laptop, smartphone, etc.</p>
desktops, laptops, and servers protecting system state, file data and applicatio
n data like MS SQL, Exchange and
desktops, laptops, and servers protecting system state, file data and applicatio
n data like MS SQL, Exchange and
The name of laptop = lenovo?M
```

```
(base) ┌──(birud㉿kali)-[~/Desktop/CN/Assignment1]
└─$ tshark -r capture.pcap -Y 'tcp contains "lenovo"' -T fields -e tcp.checksum
0x0a61
```

The laptop name is **lenovo**

The checksum of the packet is 0x0a61 (or 2657)

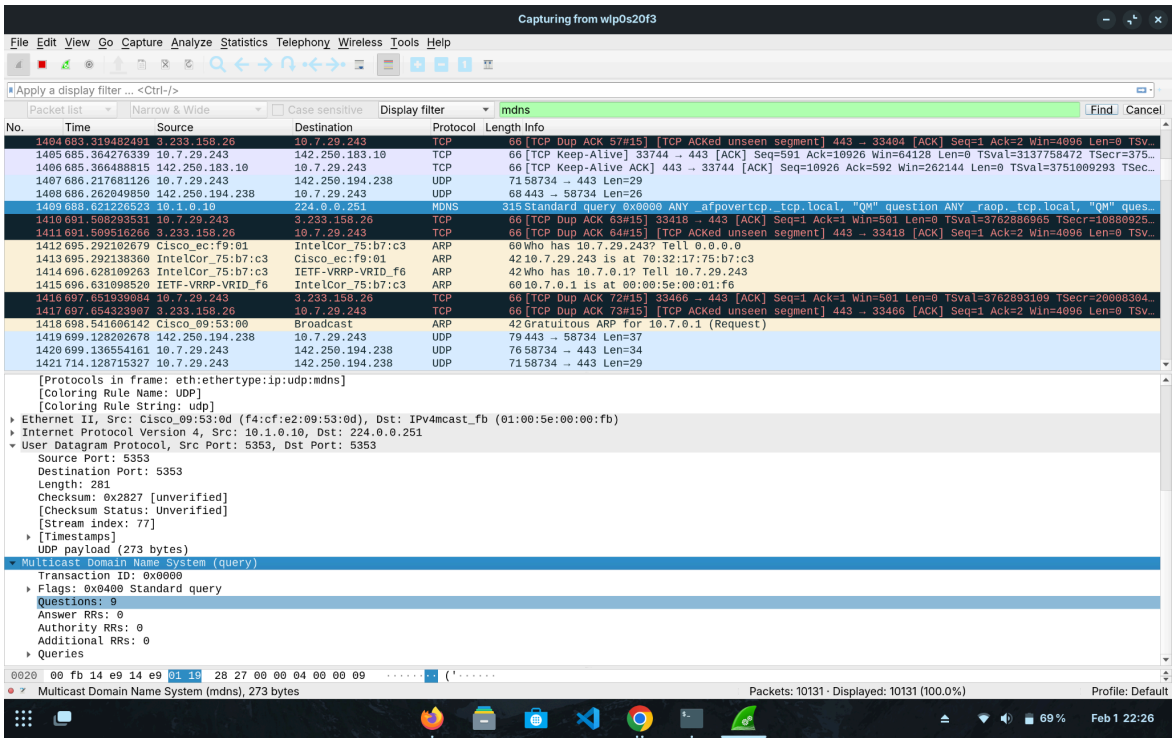
4) Find the number of packets which contain the message “Order successful”.

```
(base) ┌──(birud㉿kali)-[~/Desktop/CN/Assignment1]
└─$ strings capture.pcap | grep -c "Order Successful"
40
```

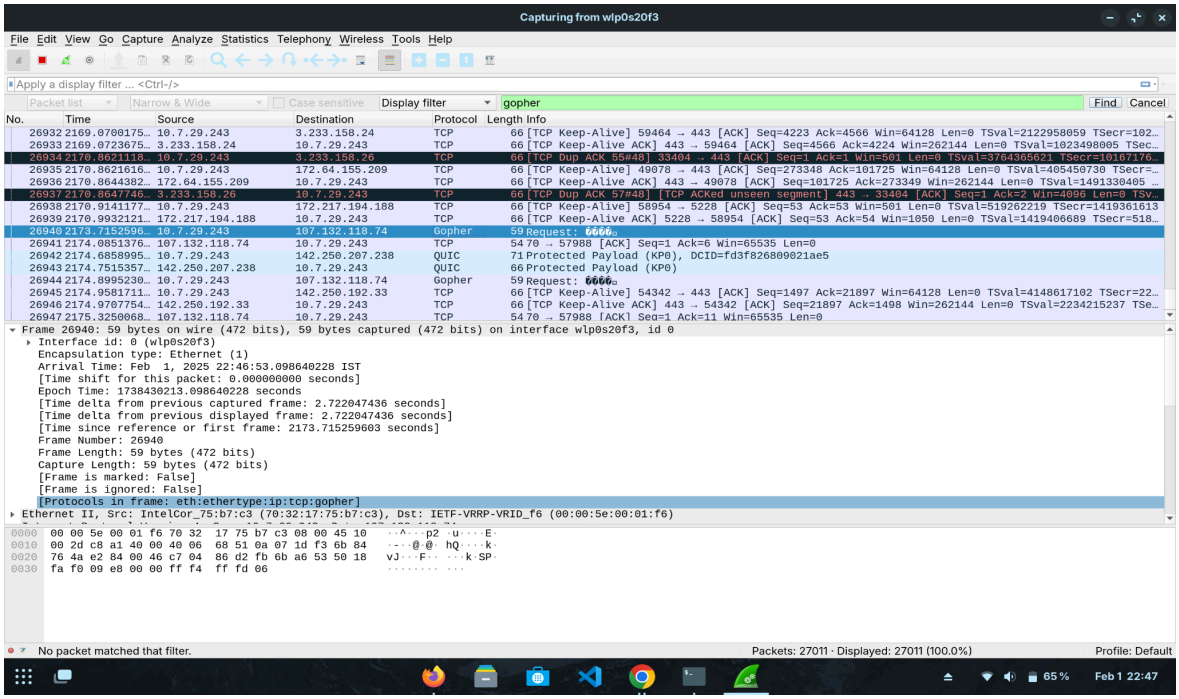
Part 3

Question 1:

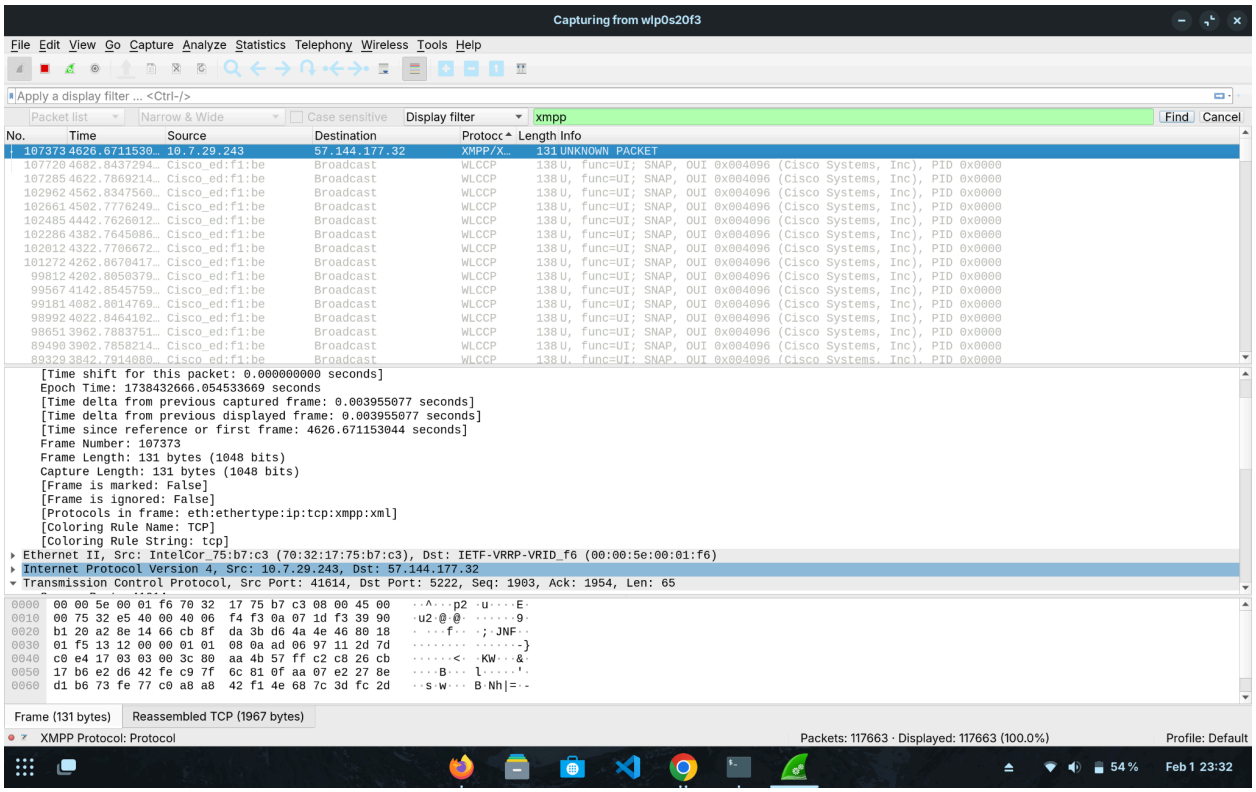
MDNS:



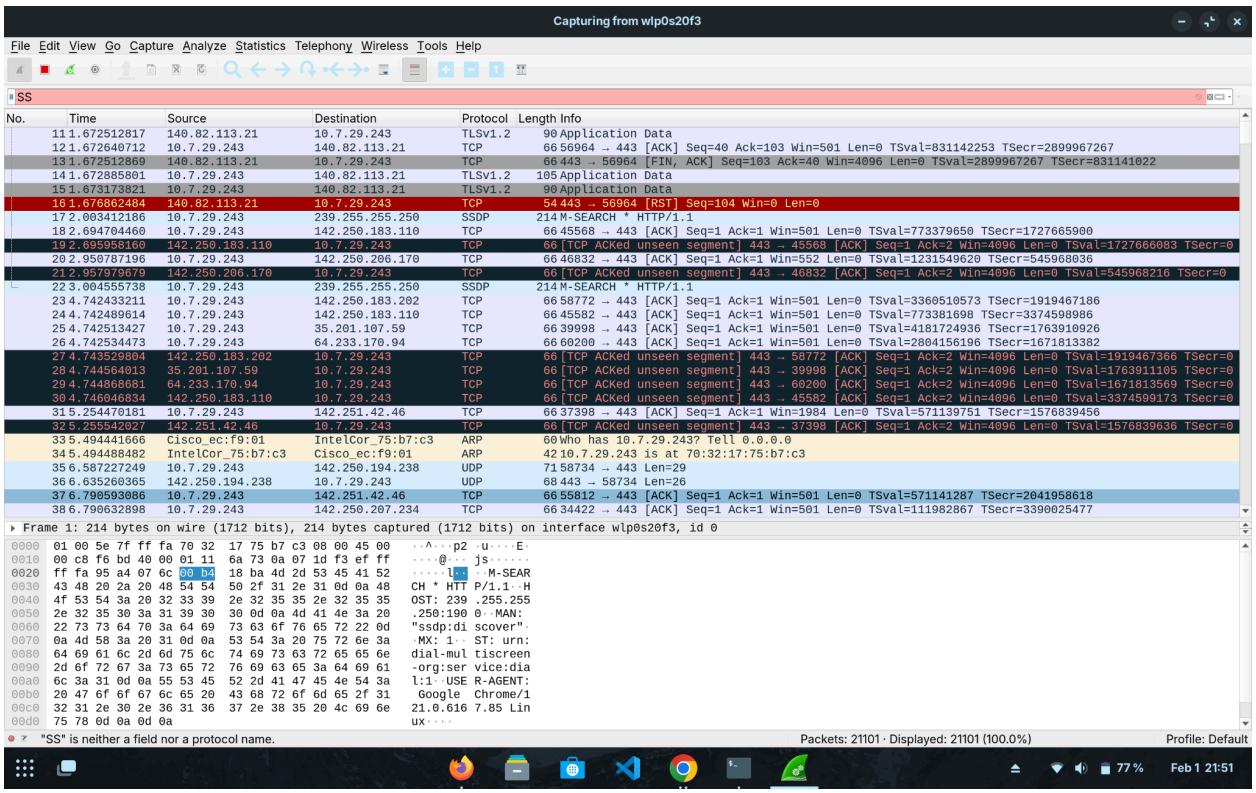
Gopher:



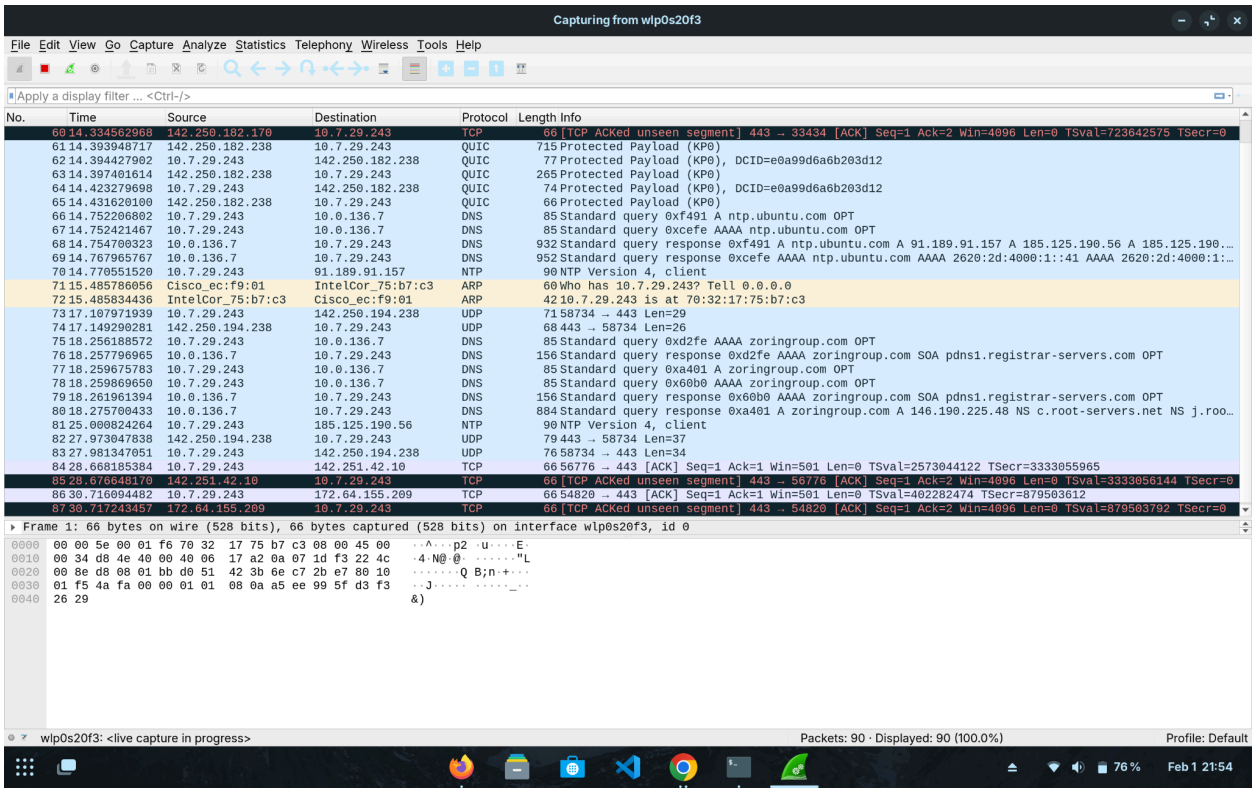
XMPP:


Capturing from wlp0s20f3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Packet list
Narrow & Wide
Case sensitive
Display filter
xmpp
Find
Cancel
No. Time Source Destination Protocol Length Info
107373 4626.6711530… 10.7.29.243 57.144.177.32 XMPP/X… 131 UNKNOWN PACKET
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1738432666.054533669 seconds
[Time delta from previous captured frame: 0.003955077 seconds]
[Time delta from previous displayed frame: 0.003955077 seconds]
[Time since reference or first frame: 4626.671153044 seconds]
Frame Number: 107373
Frame Length: 131 bytes (1048 bits)
Capture Length: 131 bytes (1048 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp:xmpp:xml]
[Coloring Rule Name: TCP]
[Coloring Rule String: tcp]
Ethernet II, Src: IntelCor_75:b7:c3 (70:32:17:75:b7:c3), Dst: IETF-VRRP-VRID_f6 (00:00:5e:00:01:f6)
Internet Protocol Version 4, Src: 10.7.29.243, Dst: 57.144.177.32
Transmission Control Protocol, Src Port: 41614, Dst Port: 5222, Seq: 1903, Ack: 1954, Len: 65
Frame (131 bytes)
Reassembled TCP (1967 bytes)
XMPP Protocol: Protocol
Packets: 117663 · Displayed: 117663 (100.0%)
Profile: Default
54 %
Feb 1 23:32

SSDP:


Capturing from wlp0s20f3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
SS
No. Time Source Destination Protocol Length Info
11 1.672512817 140.82.113.21 10.7.29.243 TLSv1.2 90 Application Data
12 1.672640712 10.7.29.243 140.82.113.21 TCP 66 56964 → 443 [ACK] Seq=40 Ack=103 Win=501 Len=0 TSval=831142253 TSecr=2899967267
13 1.672512869 140.82.113.21 10.7.29.243 TCP 66 443 → 56964 [FIN, ACK] Seq=103 Ack=40 Win=4096 Len=0 TSval=2899967267 TSecr=831141022
14 1.672885801 10.7.29.243 140.82.113.21 TLSv1.2 105 Application Data
15 1.673173821 10.7.29.243 140.82.113.21 TLSv1.2 90 Application Data
16 1.676862484 140.82.113.21 10.7.29.243 TCP 54 443 → 56964 [RST] Seq=104 Win=0 Len=0
17 2.003412186 10.7.29.243 239.255.255.250 SSDP 214 M-SEARCH * HTTP/1.1
18 2.694704460 10.7.29.243 142.250.183.110 TCP 66 45568 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=773379650 TSecr=1727665900
19 2.695958160 142.250.183.110 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 45568 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1727666083 TSecr=0
20 2.950787196 10.7.29.243 142.250.206.170 TCP 66 46832 → 443 [ACK] Seq=1 Ack=1 Win=552 Len=0 TSval=1231549620 TSecr=545968036
21 2.957979679 142.250.206.170 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 46832 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=545968216 TSecr=0
22 3.004555738 10.7.29.243 239.255.255.250 SSDP 214 M-SEARCH * HTTP/1.1
23 4.742433211 10.7.29.243 142.250.183.202 TCP 66 58772 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=3360510573 TSecr=1919467186
24 4.742489614 10.7.29.243 142.250.183.110 TCP 66 45582 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=773381698 TSecr=3374598986
25 4.742513427 10.7.29.243 35.201.107.59 TCP 66 39998 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=4181724936 TSecr=1763910926
26 4.742534473 10.7.29.243 64.233.170.94 TCP 66 60200 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=2804156196 TSecr=1671813382
27 4.743529804 142.250.183.202 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 58772 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1919467366 TSecr=0
28 4.744564013 35.201.107.59 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 39998 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1763911105 TSecr=0
29 4.744868681 64.233.170.94 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 60200 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1671813569 TSecr=0
30 4.746046834 142.250.183.110 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 45582 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=3374599173 TSecr=0
31 5.254470181 10.7.29.243 142.251.42.46 TCP 66 37398 → 443 [ACK] Seq=1 Ack=1 Win=1984 Len=0 TSval=571139751 TSecr=1576839456
32 5.255542027 142.251.42.46 10.7.29.243 TCP 66 [TCP ACKed unseen segment] 443 → 37398 [ACK] Seq=1 Ack=2 Win=4096 Len=0 TSval=1576839636 TSecr=0
33 5.494441666 Cisco_ec:f9:01 IntelCor_75:b7:c3 ARP 60 Who has 10.7.29.243? Tell 0.0.0.0
34 5.494488482 IntelCor_75:b7:c3 Cisco_ec:f9:01 ARP 42 10.7.29.243 is at 70:32:17:75:b7:c3
35 6.587227249 10.7.29.243 142.250.194.238 UDP 71 58734 → 443 Len=29
36 6.635260365 142.250.194.238 10.7.29.243 UDP 68 443 → 58734 Len=26
37 6.790593086 10.7.29.243 142.251.42.46 TCP 66 55812 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=571141287 TSecr=2041958618
38 6.790632898 10.7.29.243 142.250.207.234 TCP 66 34422 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=111982867 TSecr=3390025477
Frame 1: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits) on interface wlp0s20f3, id 0
"SS" is neither a field nor a protocol name.
Packets: 21101 · Displayed: 21101 (100.0%)
Profile: Default
77 %
Feb 1 21:51

NTP:



mDNS (Multicast DNS) is a protocol that allows devices on a local network to resolve hostnames to IP addresses without needing a central DNS server. It is commonly used in zero-configuration networking (Zeroconf) for service discovery, such as in Apple's Bonjour and IoT devices. RFC: 6762

Gopher – A text-based protocol from the early internet (RFC 1436) designed for retrieving hierarchical documents, similar to HTTP but with a menu-driven interface. It was largely replaced by the web but still has niche usage.

XMPP (Extensible Messaging and Presence Protocol) is an open standard communication protocol used for real-time messaging, presence notification, and contact list management. It is widely used for instant messaging and chat applications due to its extensibility, scalability, and support for federated communication.RFC 6120

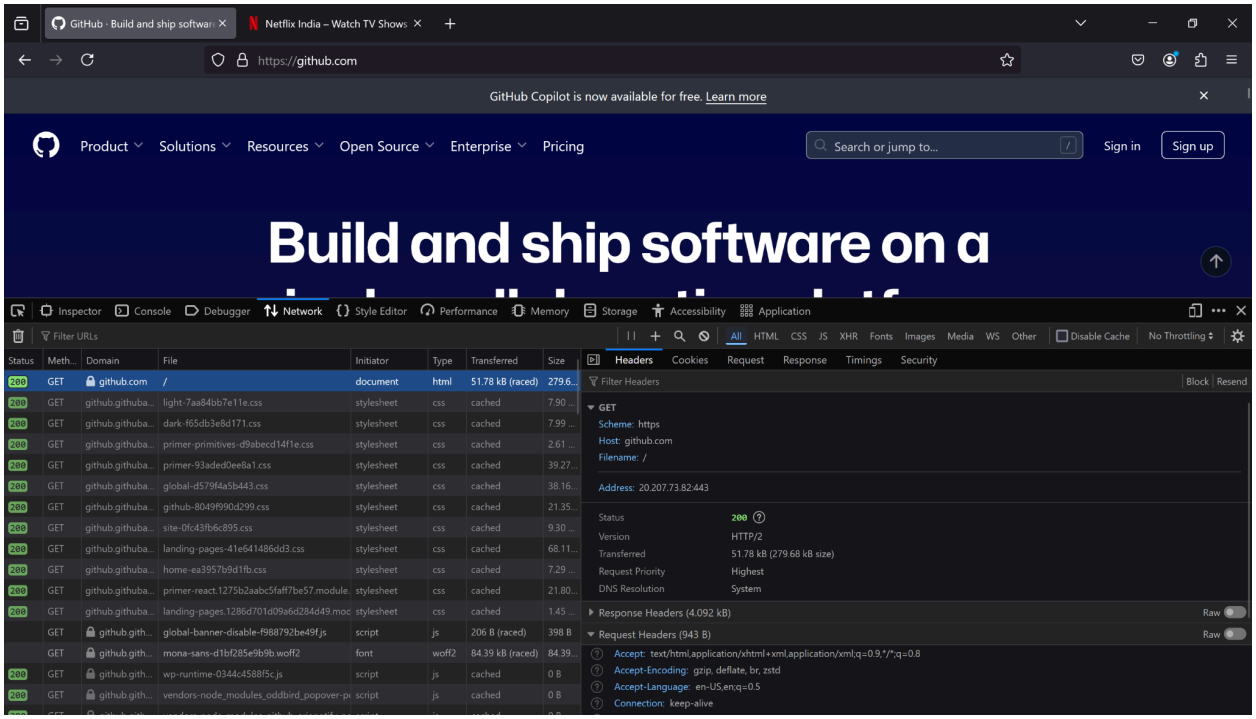
SSDP (Simple Service Discovery Protocol): A network protocol used in UPnP (Universal Plug and Play) to discover devices and services on a local network, often used in smart home and IoT applications. RFC: 2326

NTP (Network Time Protocol): A protocol designed to synchronize clocks across devices on a network with high precision, ensuring accurate timekeeping for distributed systems and security mechanisms. RFC: 5905

Question 2:

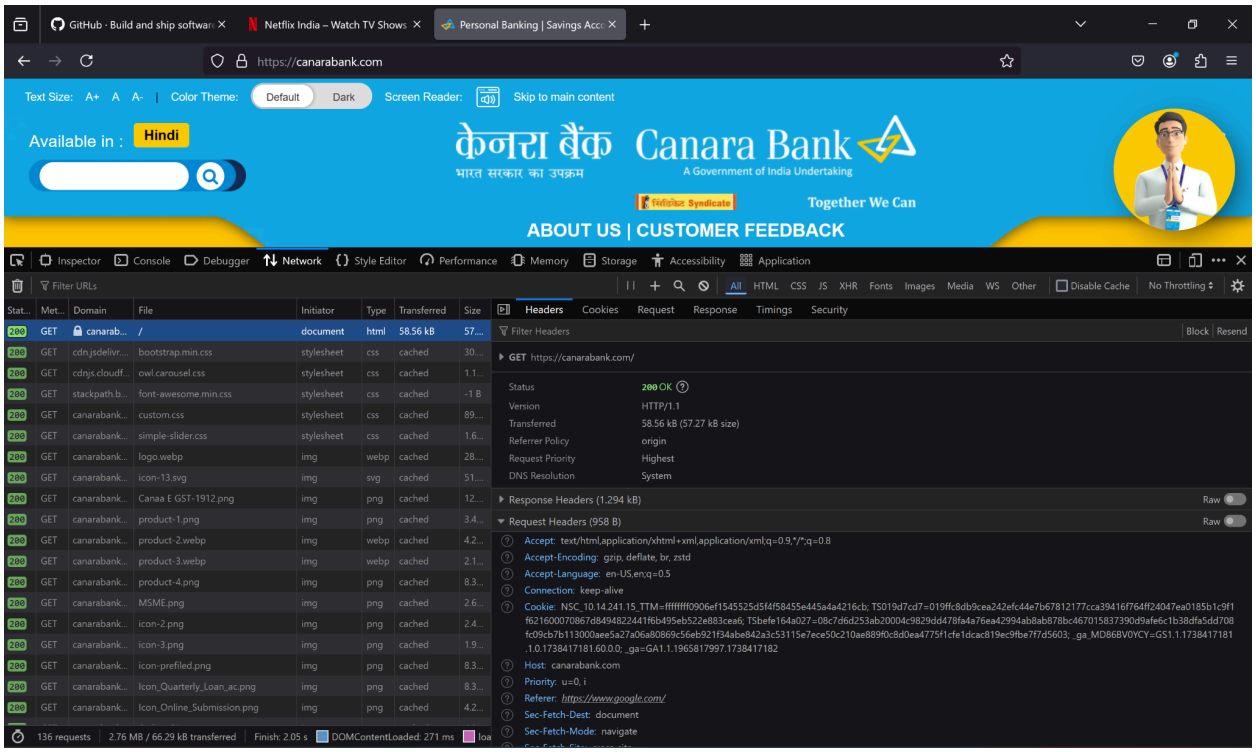
Part a:

1)github.com

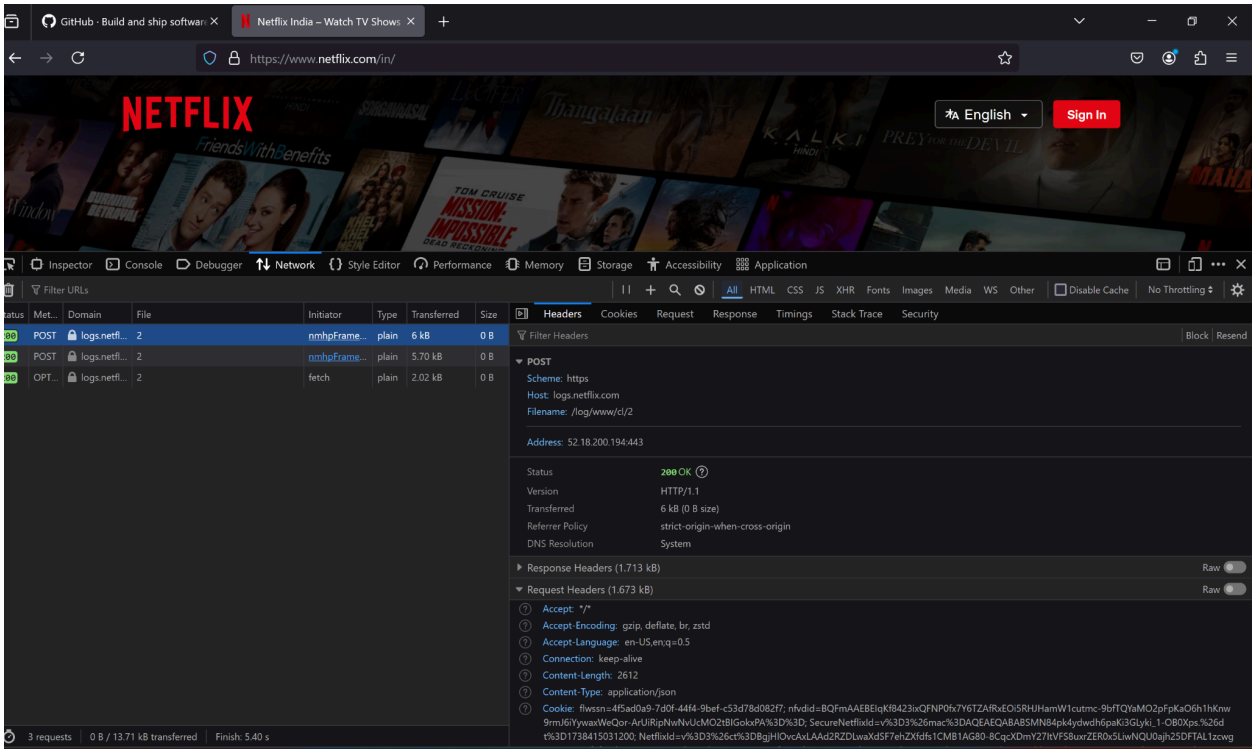


Keep-alive connection implies it is persistent. IP address, protocol version are visible in the screenshot.
Request Line: GET / HTTP/2
IP Address: 20.207.73.82.443
Connection Type: persistent

2)canarabank.com

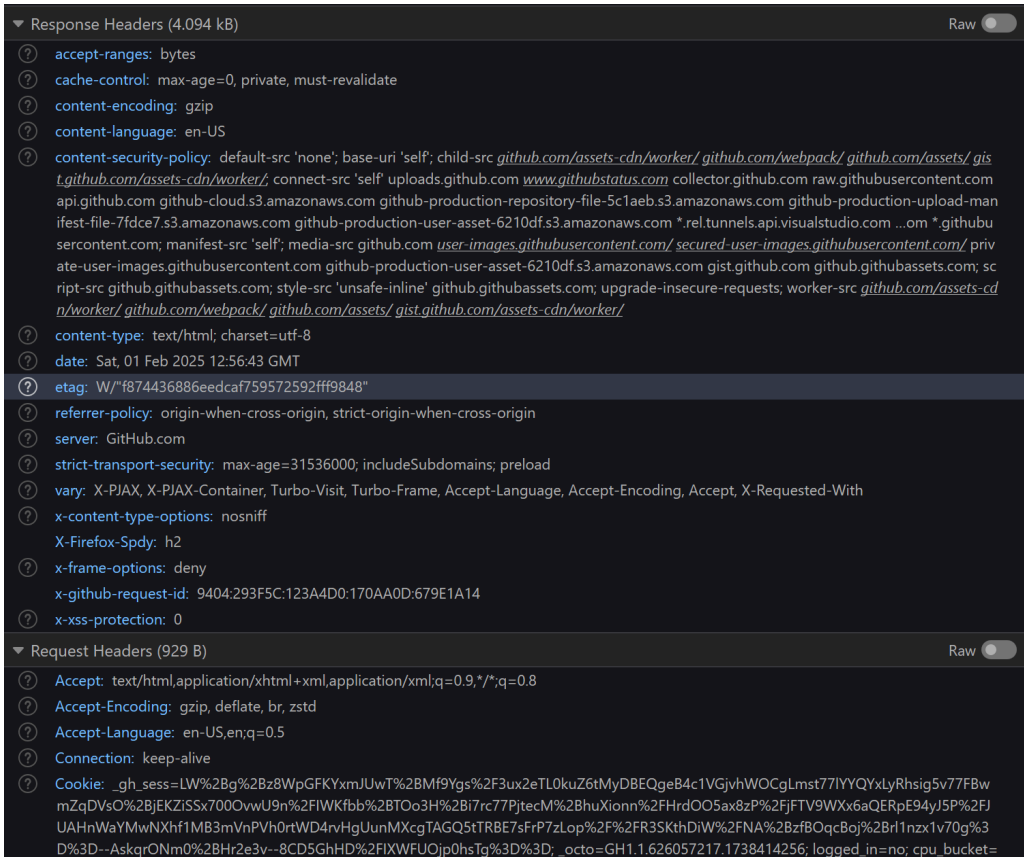


3)netflix.com

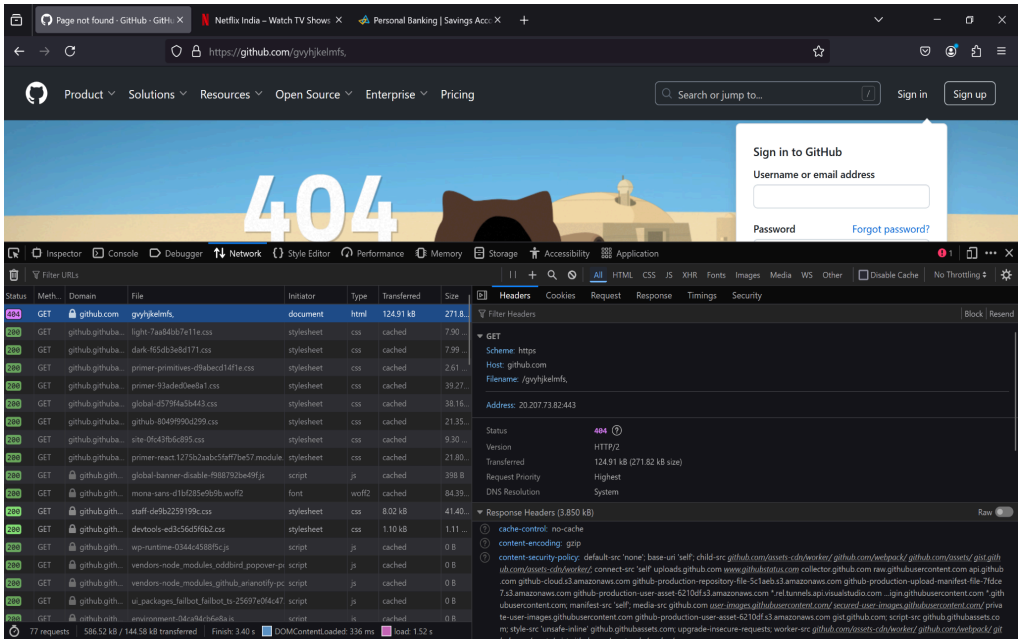


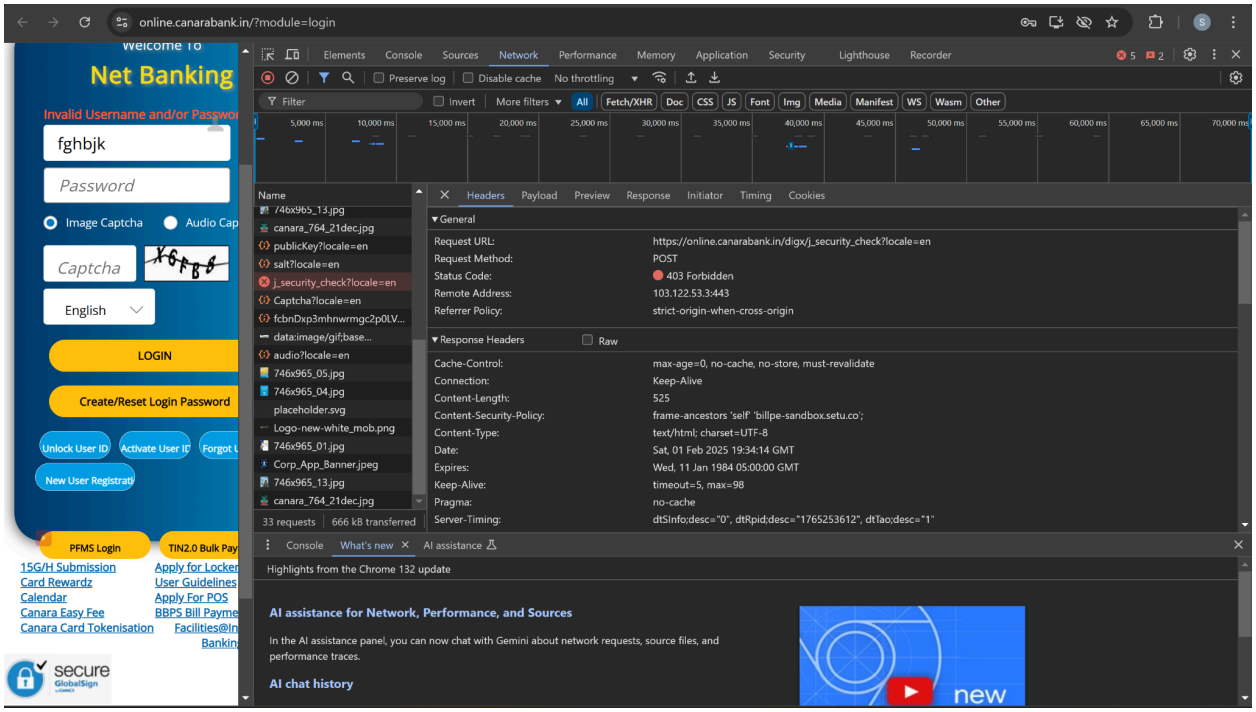
Part b(for github.com):

Request and Response Headers (any 3 fields can be chosen):



HTTP Error Codes:





404 Not Found – The requested resource or webpage could not be found on the server, often due to a broken or incorrect URL.

400 Bad Request – The server cannot process the request due to malformed syntax, invalid parameters, or a client error.

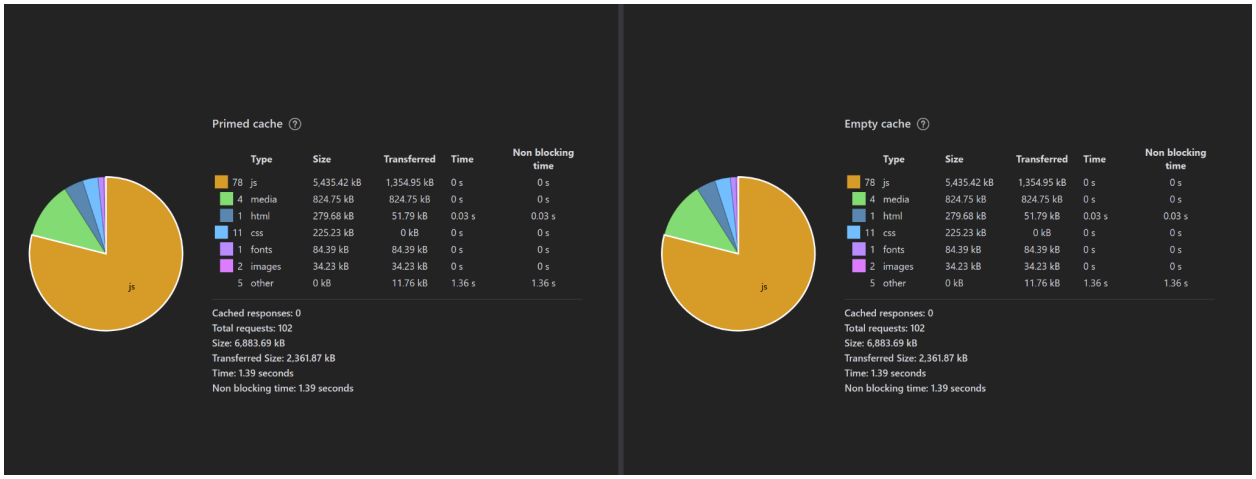
403 Forbidden – The server understands the request but refuses to authorize access, usually due to insufficient permissions.

429 Too Many Requests – This error occurs when the user has sent too many requests in a given amount of time, often due to rate limiting. It indicates that the client should slow down and try again later.

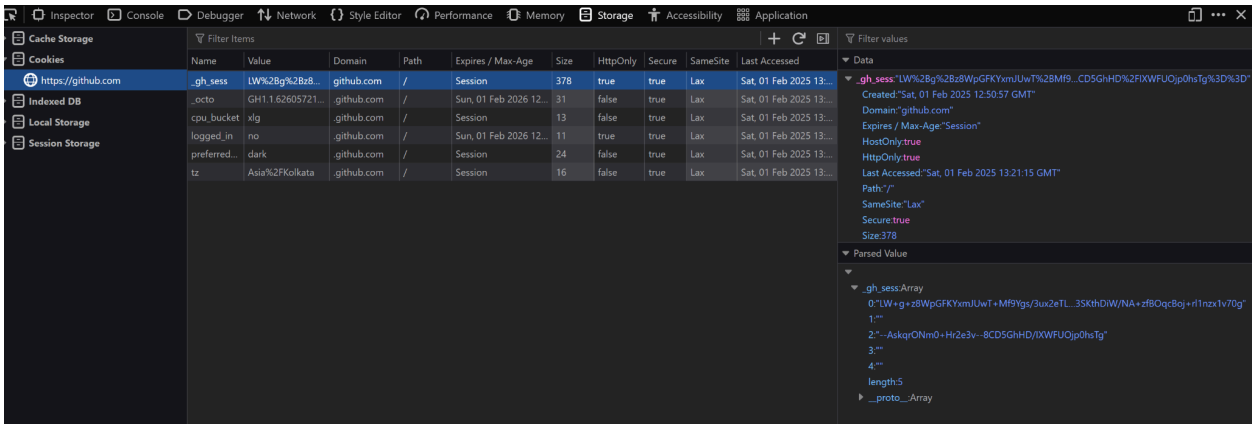
Part c(for github.com):

Browser Name: Firefox

Performance Metrics



List of Cookies



Associated flags in response and request headers

```
strict-transport-security: max-age=31536000; includeSubdomains; preload
vary: X-Requested-With, X-PJAX-Container, Turbo-Frame, Turbo-Visit, Accept-Encoding, Accept, X-Requested-With
x-content-type-options: nosniff
X-Firefox-Spdy: h2
x-frame-options: deny
x-github-request-id: 9A6A:15D61E:DE55E0:1184ADF:679E4953
x-xss-protection: 0
```

```
Host: github.com
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
TE: trailers
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:122.0) Gecko/20100101 Firefox/122.0
```