CS 331: Computer Networks

Assignment 1

Team 17: Hrriday Ruparel (22110099), Kishan Ved (22110122)

Repository Link

Part 1: Metrics and Plots

For this question, we made a sniffer using `c++`. Packets were sent via `tcpreplay` on interface `lo`. The sniffer simply stored captured packets in a `pcap` file without processing them. **It was observed that every packet is sniffed twice (ie; every packet is duplicated)**. We believe this happens because the same packet is sniffed when sent from and when received by the interface `lo`. Also, our code sniffs TCP and UDP based packets since we observed that all the packets contain only TCP and UDP as Transport Layer Protocols.

Wireshark · Protocol Hierarchy Statistics · 8.pcap

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	328623	100.0	324313533	251 k	0	0	0
Ethernet	100.0	328623	1.4	4600722	3,570	0	0	0
Internet Protocol Version 6	0.0	68	0.0	2720	2	0	0	0
User Datagram Protocol	0.0	68	0.0	544	0	0	0	0
Multicast Domain Name System	0.0	68	0.0	13055	10	68	13055	10
Internet Protocol Version 4	99.9	328421	2.0	6568420	5,097	0	0	0
User Datagram Protocol	58.0	190680	0.5	1525440	1,183	0	0	0
Session Traversal Utilities for NAT	0.0	26	0.0	676	0	26	676	0
QUIC IETF	32.3	106047	31.3	101557048	78 k	104577	100625738	78 k
Malformed Packet	0.1	307	0.0	0	0	307	0	0
Multicast Domain Name System	0.0	69	0.0	13328	10	69	13328	10
ISO 8602/X.234 CLTP ConnectionLess Transport Protocol	0.0	7	0.0	560	0	0	0	0
R-GOOSE	0.0	1	0.0	1174	0	1	1174	0
Malformed Packet	0.0	6	0.0	0	0	6	0	0
ETSI Distribution & Communication Protocol (for DRM)	0.0	8	0.0	9729	7	2	2500	1
Malformed Packet	0.0	6	0.0	0	0	6	0	0
Domain Name System	0.4	1364	0.1	242076	187	1364	242076	187
Data	25.7	84322	26.4	85767025	66 k	84322	85767025	66 k
Transmission Control Protocol	41.9	137741	38.3	124167320	96 k	113336	99584200	77 k
Transport Layer Security	7.7	25361	37.4	121169143	94 k	23203	89500424	69 k
Malformed Packet	0.0	1	0.0	0	0	1	0	0
Hypertext Transfer Protocol	0.0	8	0.0	9152	7	7	8280	6
Online Certificate Status Protocol	0.0	1	0.0	471	0	1	471	0
Domain Name System	0.3	858	0.1	227659	176	858	227659	176
Data	0.1	197	0.1	179499	139	197	179499	139
Apache JServ Protocol v1.3	0.0	139	0.0	73176	56	138	72910	56
Address Resolution Protocol	0.0	134	0.0	5480	4	134	5480	4

Upon interaction with other groups, it was found that using a different interface enables sniffing the packets only once. However, when we tried sniffing on the only other interface available on our laptops (`wlp` interface), the sniffing process was **extremely slow (<100 pps)**. The same

code when run on a laptop with `eht0` interface sniffs fast and only one packet at a time. Thus, we have used `lo`, and then run a simple command to drop all duplicate packets.

```
editcap -d input.pcap output.pcap
```

The `output.pcap` obtained after doing this contains no duplicate packets. The following questions are answered based on the analysis of the `output.pcap` file. In all the codes written for performing analysis on the `output.pcap` file, the packets with source or destination IP as `127.0.0.1` are ignored. These packets are extra packets that are captured while using the `lo` interface. We observed that 8.pcap has no packets with `127.0.0.1` as the source or the destination ip, hence, ignoring these does not change the answers.

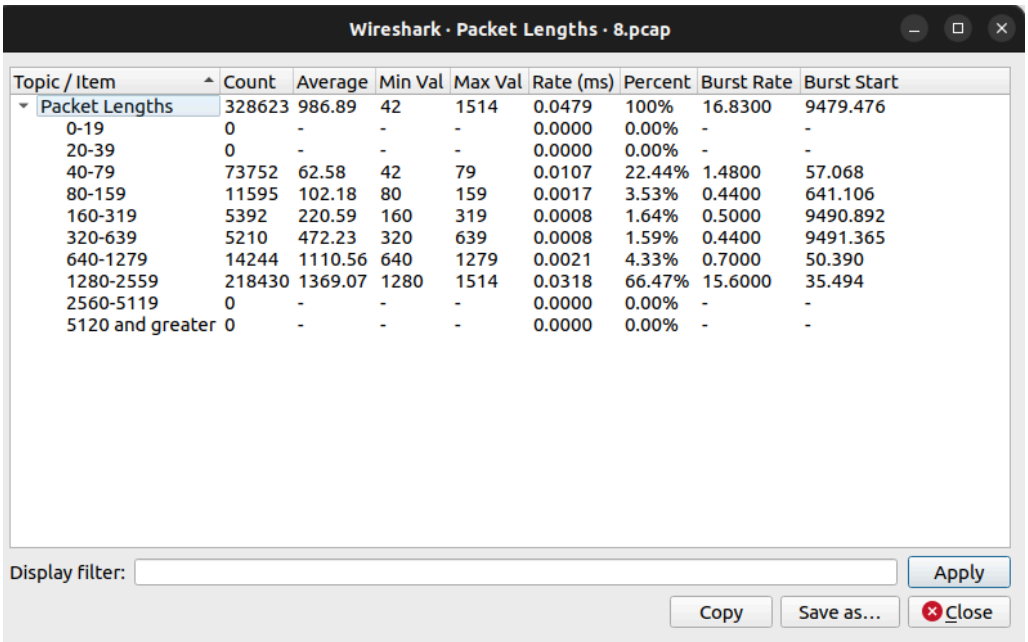
For our team, the `input.pcap` file was `8.pcap`

All media / images used in this report can be found in the GitHub repository

Q1) Find the total amount of data transferred (in bytes), the total number of packets transferred, and the minimum, maximum, and average packet sizes. Also, show the distribution of packet sizes (e.g., by plotting a histogram of packet sizes).

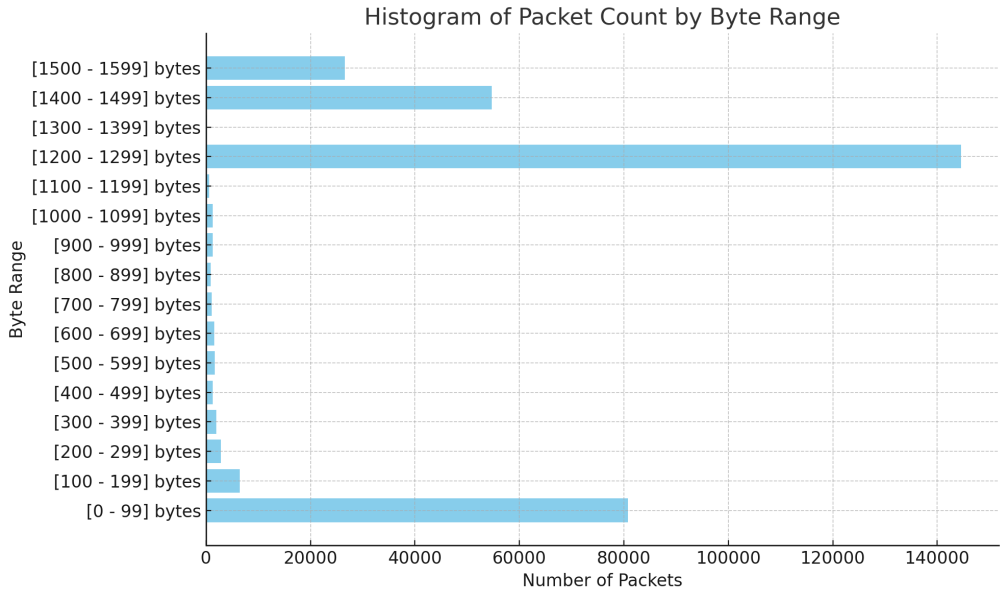
A)
Total Data Transferred: 323,446,252 bytes (Actual Data Size: 329,571,525 bytes)
Total Packets Received: 327,775 (Actual Number of Packets: 328,623)
Minimum Packet Size: 42 bytes (Actual Min Packet Size: 42 bytes)
Maximum Packet Size: 1514 bytes (Actual Max Packet Size: 1514 bytes)
Average Packet Size: 986.794 bytes (Average Packet Size: 986.89 bytes)

Original `8.pcap` Statistics



Packet Size Distribution (Histogram) from `output.pcap`:
[0 - 99] bytes: 80740 packets
[100 - 199] bytes: 6477 packets
[200 - 299] bytes: 2870 packets
[300 - 399] bytes: 1985 packets
[400 - 499] bytes: 1296 packets
[500 - 599] bytes: 1711 packets
[600 - 699] bytes: 1603 packets
[700 - 799] bytes: 1048 packets
[800 - 899] bytes: 871 packets
[900 - 999] bytes: 1325 packets
[1000 - 1099] bytes: 1285 packets
[1100 - 1199] bytes: 637 packets
[1200 - 1299] bytes: 144545 packets
[1300 - 1399] bytes: 166 packets
[1400 - 1499] bytes: 54659 packets
[1500 - 1599] bytes: 26557 packets

The above data is used to plot this histogram



Q2) Find unique source-destination pairs (source IP:port and destination IP:port) in the captured data.

A) Number of unique source-destination pairs: **4439 pairs**

Q3) Display a dictionary where the key is the IP address and the value is the total flows for that IP address as the source. Similarly display a dictionary where the key is the IP address and the value is the total flows for that IP address as the destination. Find out which source-destination (source IP:port and destination IP:port) have transferred the most data

A)
Dictionaries are uploaded here: https://github.com/Kishan-Ved/cn_a1/blob/main/p1q3_flows.txt

Flow with the most data transferred:
Source IP: 54.230.112.80:443 -> **Destination IP**: 10.7.11.235:61198 transferred **72893512 bytes**

Q4) List the top speed in terms of 'pps' and 'mbps' that your program is able to capture the content without any loss of data when i) running both tcpreplay and your program on the same machine (VM), and ii) when running on different machines: Two student group should run the program on two different machines eg. tcpreplay on physical-machine of student1 and sniffer program physical-machine of student2. Single students should run between two VMs.

A)

For the same machine with WiFi turned off and interface set to `lo` we were able to achieve:

- **10000 pps (with packet loss of <800 packets)**
- **1000 pps (no packet loss)**

To replicate the scenario between 2 different machines, we have used a **docker container**, and run tcpreplay within the container. The packets were captured by running the sniffer in the host machine, outside the docker container. The interface of the docker container had a different IP compared to the host machine. The WiFi was turned off during the replay and the docker bridge was used to establish connection between the docker container and host machine . Please refer to this video. The video shows the transfer of some packets from the docker container to the local machine.

Part 2: Catch Me If You Can

Q1) The absolute difference of the Source port and Destination port value is given as 54286, The ACK flag is set, and the last 4 digits of the Acknowledgement number is ‘1203’. Find the Source IP and Destination IP.

A)
Protocol: TCP
Source IP: 10.7.11.235, Destination IP: 180.149.59.137
Source Port: 54729, Destination Port: 443

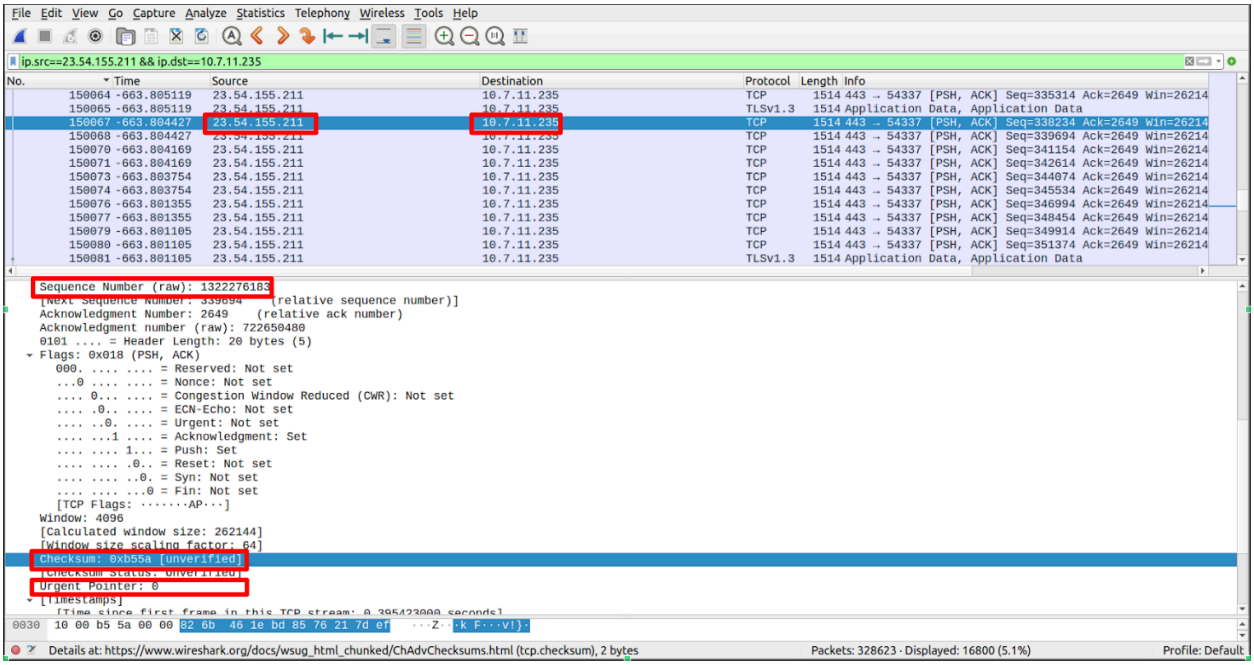
Protocol: TCP
Source IP: 10.7.11.235, Destination IP: 180.149.59.137
Source Port: 54729, Destination Port: 443

Total number of packets found: 2

Q2) The first two digits of the checksum is ‘b5’ (it is of the form 0xb5___, hexadecimal notation), the urgent data pointer is set to 0 and the last 4 four digits of Raw Sequence Number is ‘6183’. Find the source IP, destination IP and the complete checksum of this packet.

A)
Protocol: TCP
Source IP: 23.54.155.211, Destination IP: 10.7.11.235
Checksum: 0xb55a, Sequence number: 1322276183

Verification with original `8.pcap`



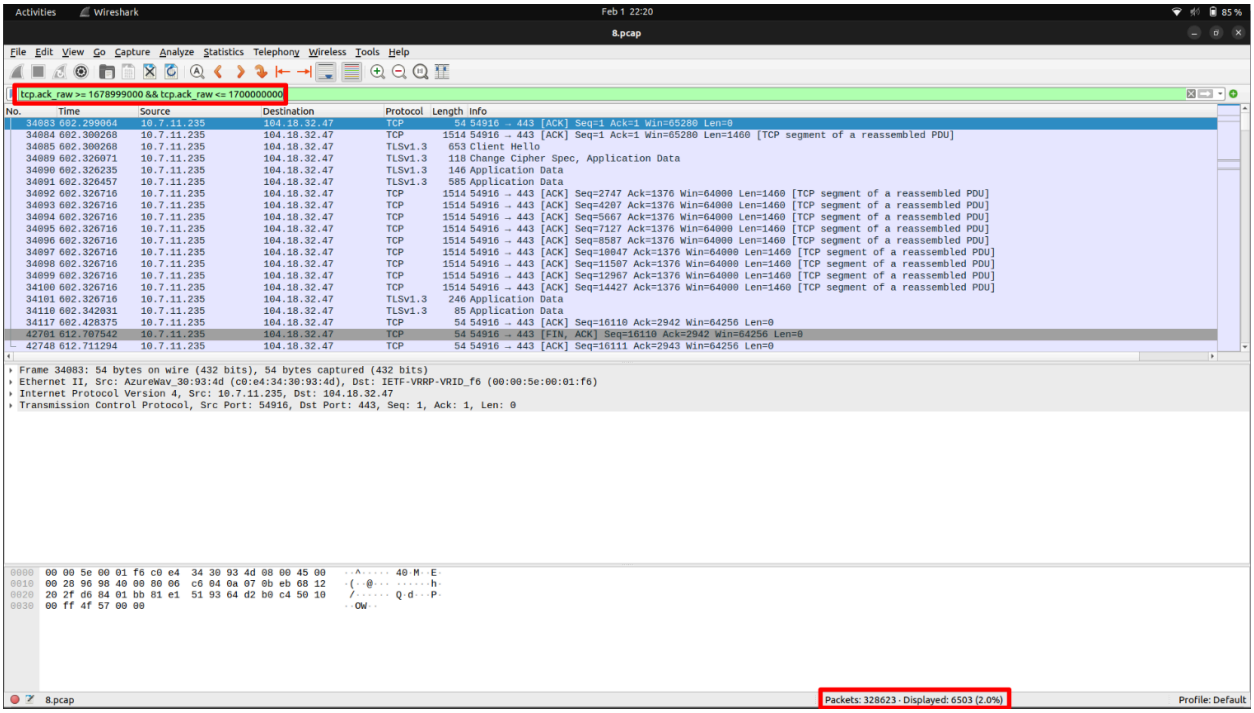
Q3) Tell the number of packets where the sum of source and destination ports is between 10,000 and 20,000.

A) **69 packets**

Q4) Find all such packets whose acknowledgement number (raw) is in between 1678999000 <= ack_number <= 1700000000.

A) **6503 packets**

Verification with original `8.pcap`



Part 3: Capture the Packets

Q1) 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.

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.

A)
The captured pcap file named `q3.pcap` (captured using Wireshark) is available at:
https://github.com/Kishan-Ved/cn_a1/blob/main/q3.pcap

Some application layer protocols not discussed in classroom are:

- **SSH (Secure SHell) [RFC 4251]**
    - SSH is used to securely log into remote machines over an unsecured network, providing encrypted communication. It typically operates on port 22.
    - There are 3 main components to it:
        - The Transport Layer Protocol: SSH-TRANS
        - The User Authentication Protocol: SSH-USERAUTH
        - The Connection Protocol: SSH-CONNECT
    - SSH and its variant protocols work at multiple levels of the network stack.

- **LDAP (Lightweight Directory Access Protocol) [RFC 4511]**
    - LDAP is used for accessing and managing directory services, such as storing user credentials in a directory. It operates primarily over TCP and UDP port 389.
    - It is a lightweight, simplified version of DAP. It is more efficient and easier to implement, especially in environments with limited resources like low-bandwidth networks.
    - Widely used in modern applications, such as for user authentication and directory services.

- **SNMP (Simple Network Management Protocol) [RFC 1157]**
    - SNMP is used for network management, allowing network devices to be monitored and controlled remotely.
    - SNMP is used to communicate management information between the network management stations and the agents in the network elements.

- **AnaSIP (Session Initiation Protocol) [RFC 2543]**
    - SIP is used for signaling and controlling multimedia communication sessions, such as voice and video calls over IP networks.

- Platforms like Zoom and Google Meet use this protocol to set up and manage real-time communication sessions.

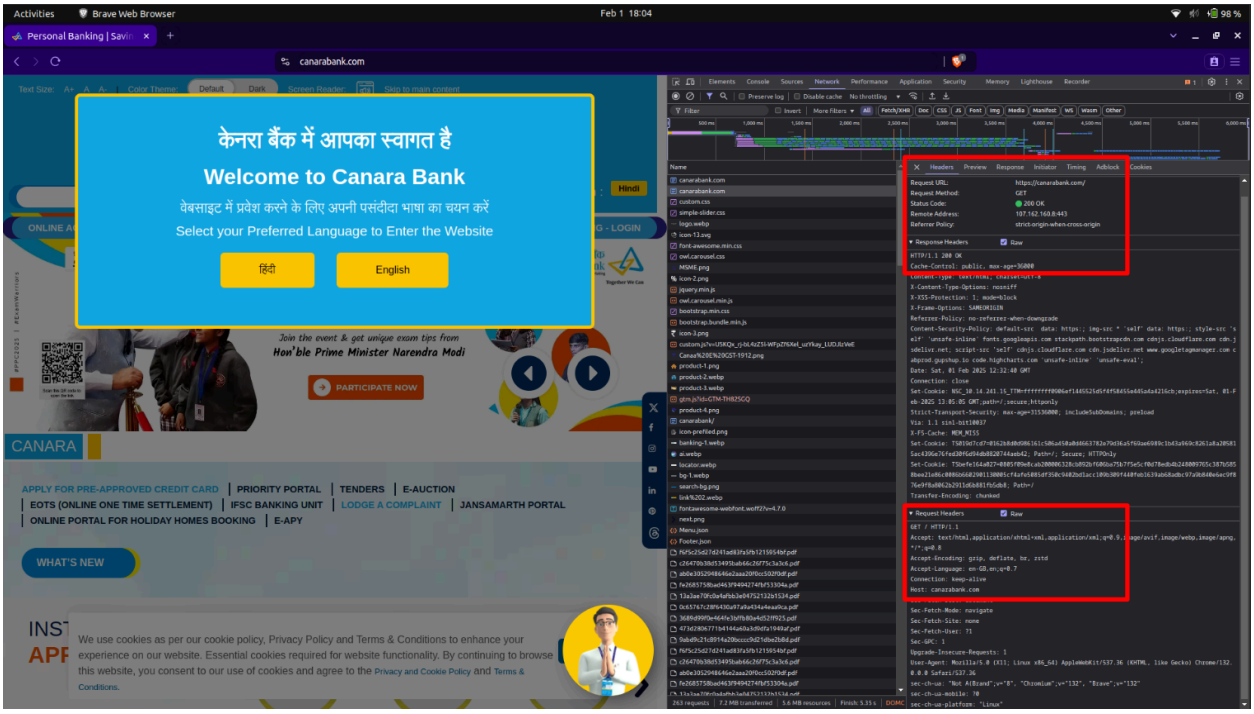
- **HLS (HTTP Live Streaming) [RFC 8216]**
  - HLS is a non-standard protocol used for streaming multimedia content over HTTP. It breaks the content into smaller chunks, allowing adaptive bitrate streaming to improve the user experience.
  - Used by live streaming platforms like Twitch, YouTube and Meta and OTT platforms like Apple TV+ and Netflix for streaming VOD.

Q2) Analyze the following details by visiting the following websites in your favourite browser:
i) canarabak.in
ii) github.com
iii) netflix.com

a. Identify 'request line' with the version of the application layer protocol and IP address. Also, identify whether the connection(s) is/are persistent or not.

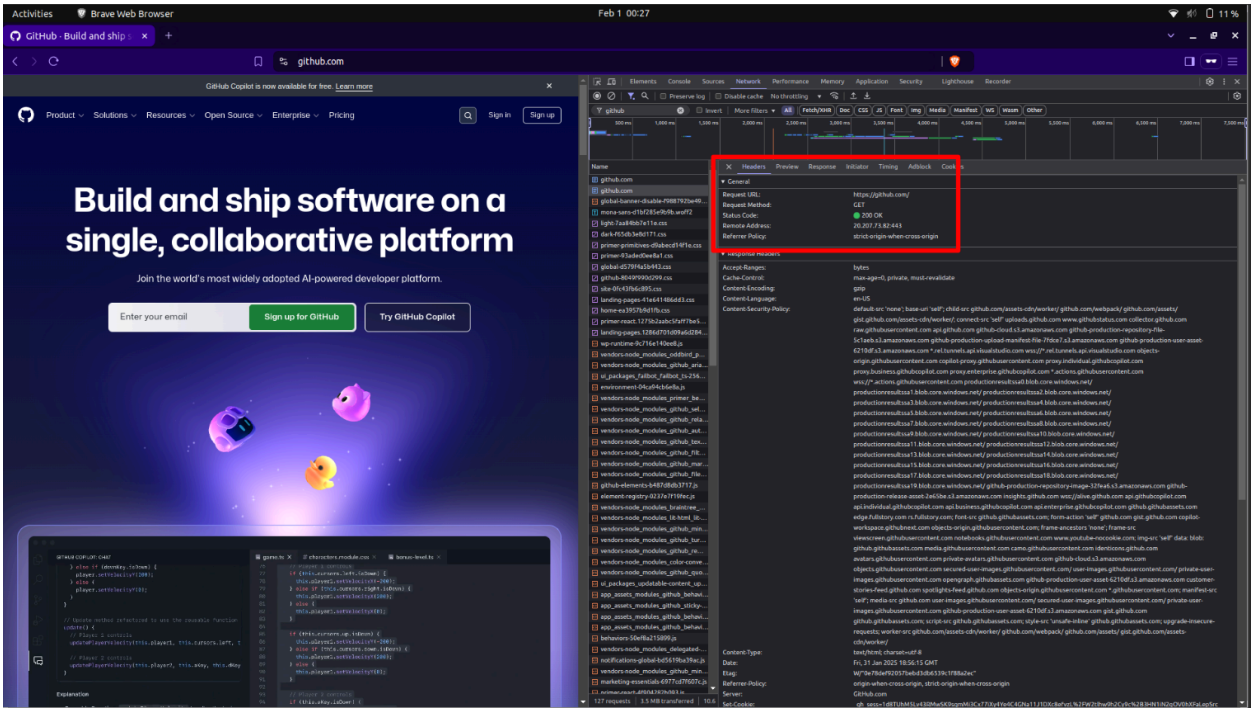
A)

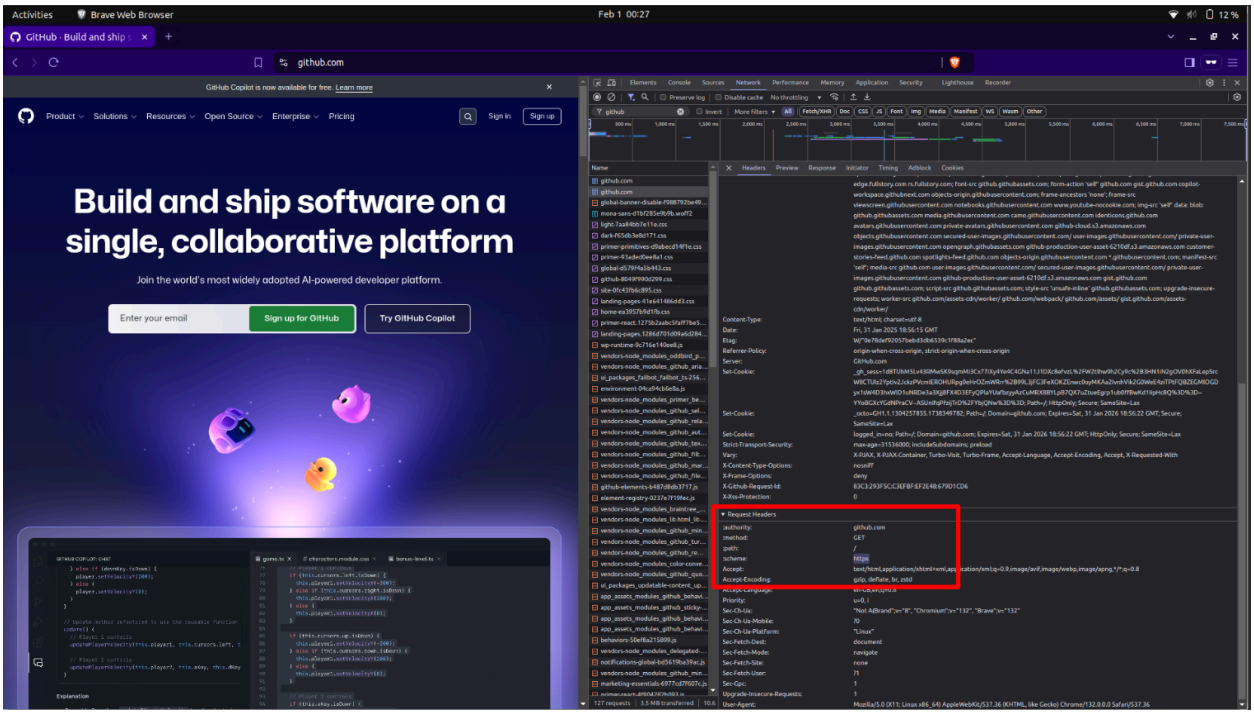
Canara Bank



Request line: GET / HTTP/1.1
Version: HTTP/1.1
IP Address: 107.162.160.8
Port: 443
Connection: Persistent (keep-alive)

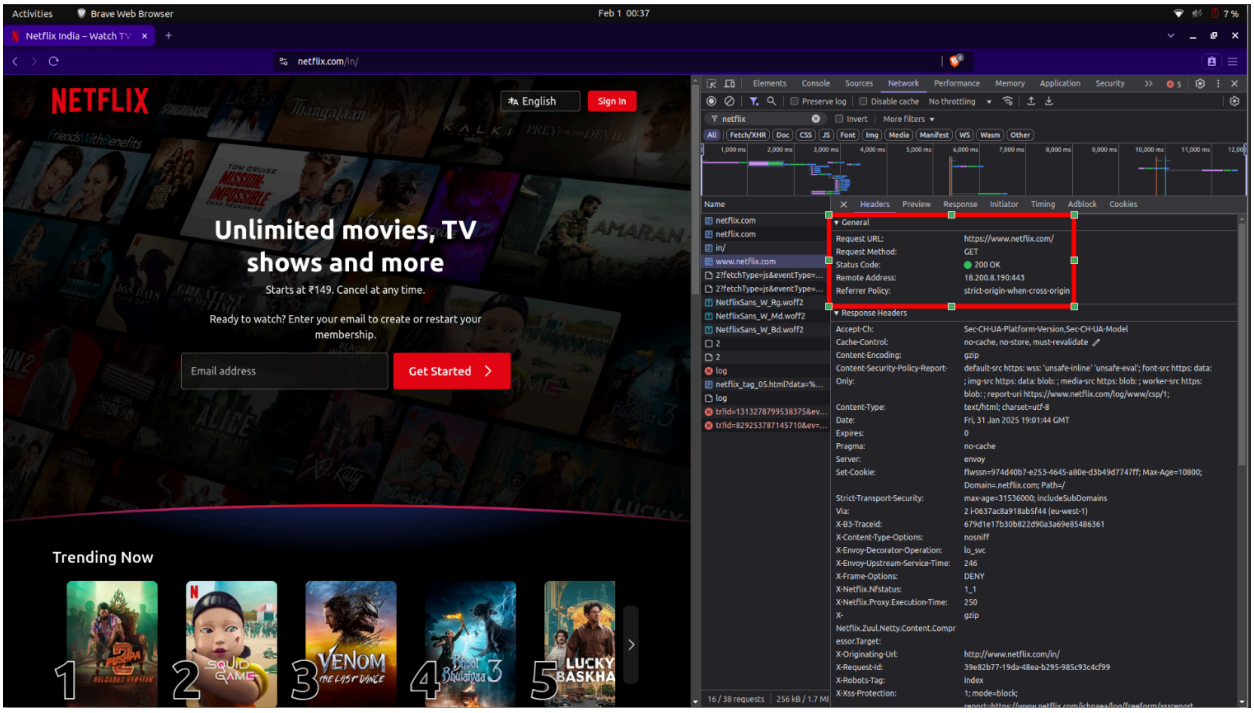
Github

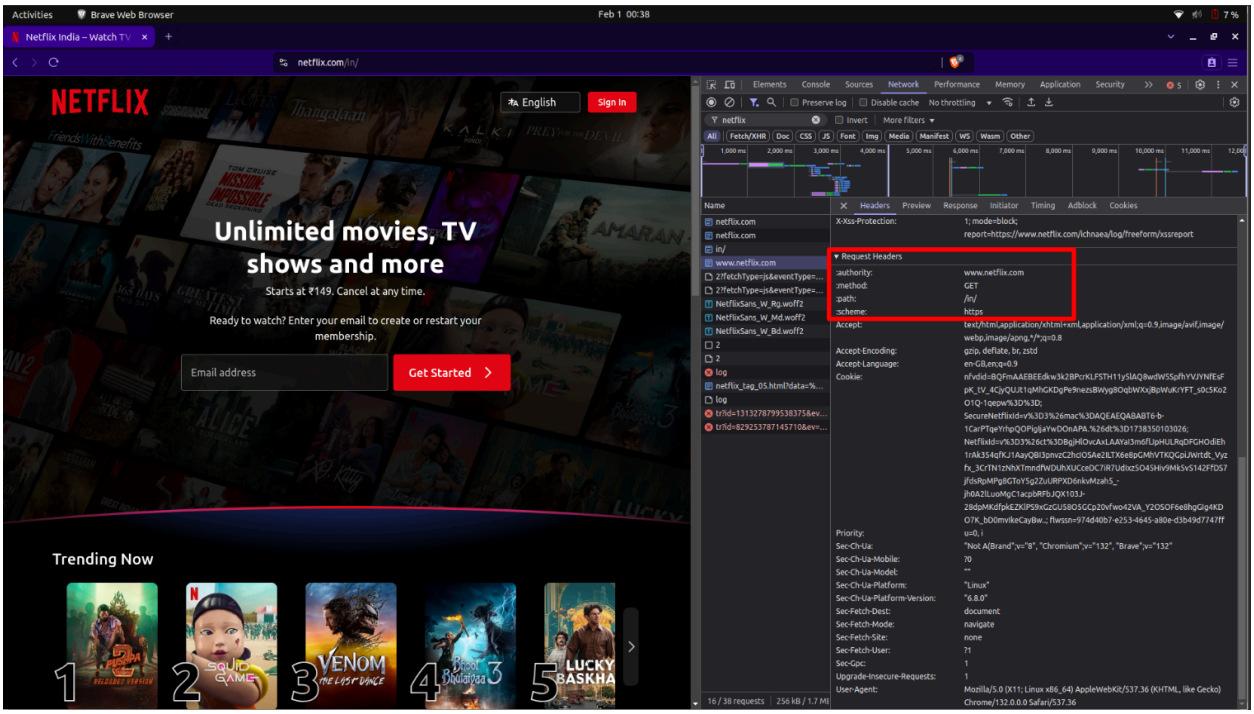




Request line: GET / HTTPS
Version: HTTPS
IP Address: 20.205.243.166
Port: 443
Connection: Persistent (keep-alive)

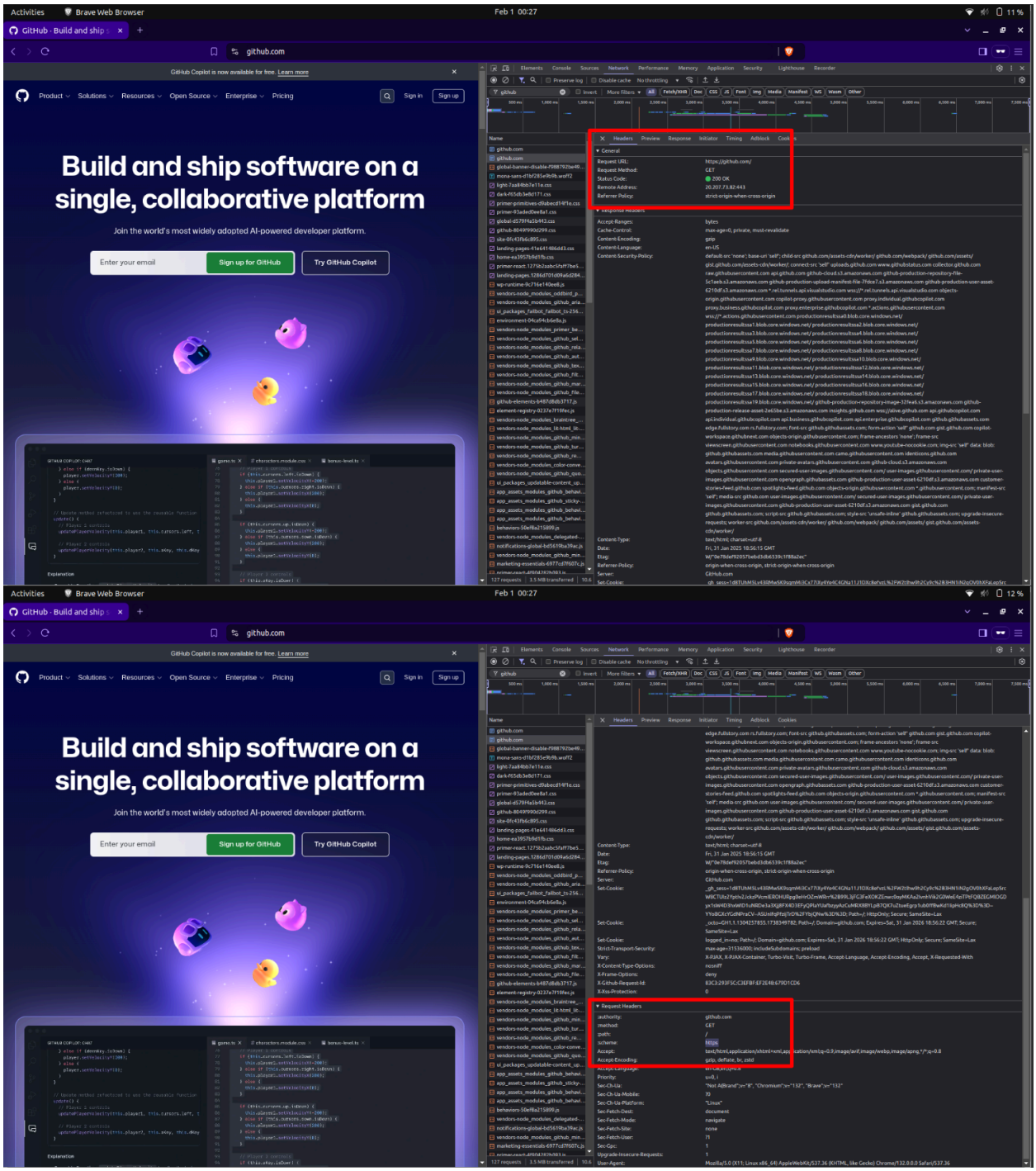
Netflix





Request line: GET /in/ HTTPS
Version: HTTPS
IP Address: 54.74.73.31
Port: 443
Connection: Persistent (keep-alive)

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



Request Header Fields:

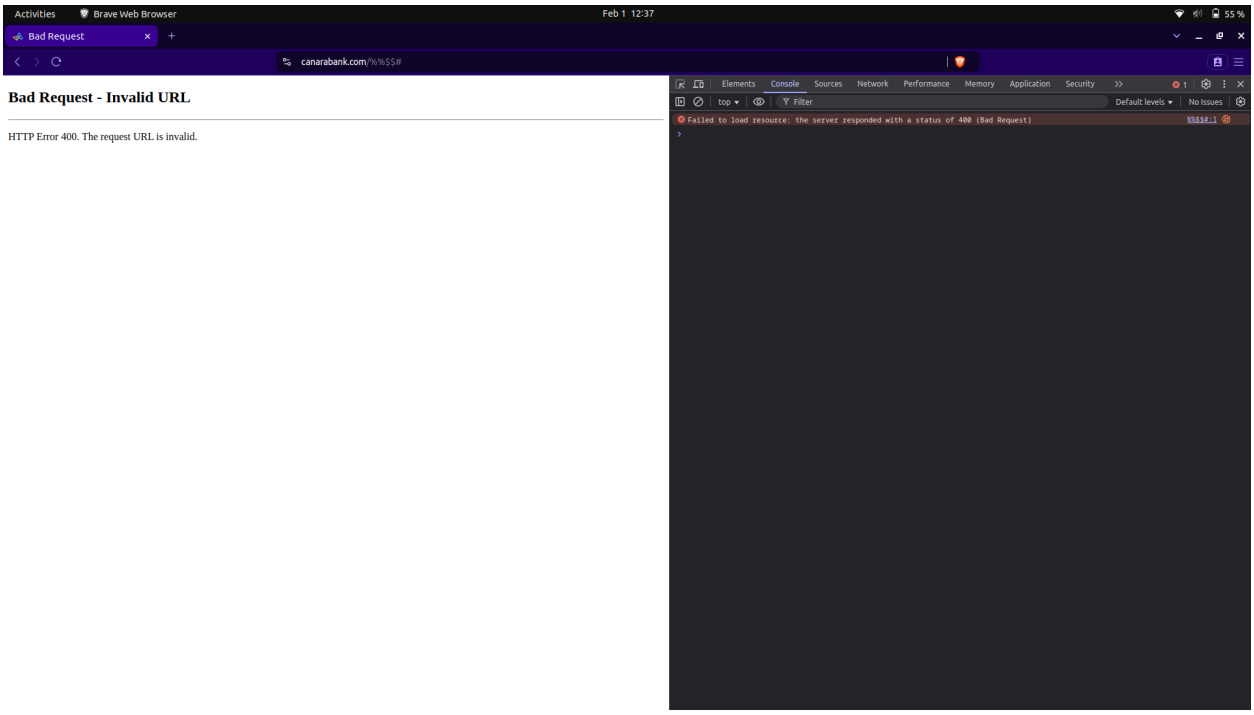
1. authority: github.com
2. method: GET
3. path: /

Response Header Fields:

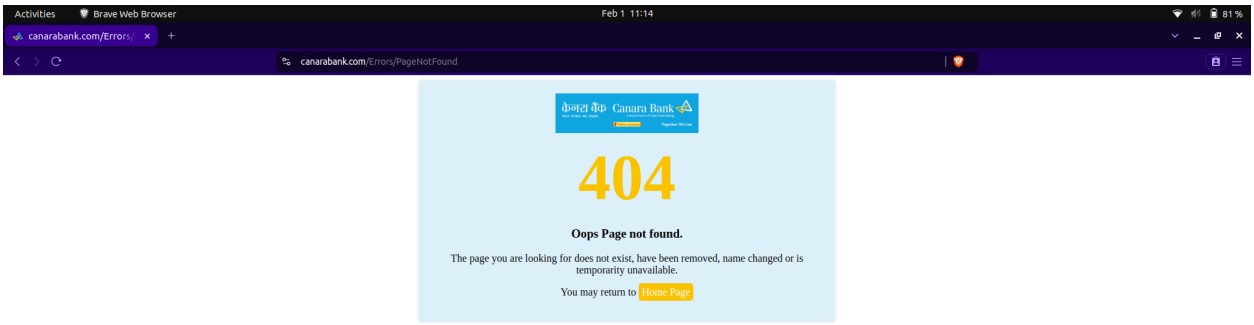
1. Content-type: text/html; charset =utf-8
2. Server: GitHub.com
3. Date: Fri, 31 Jan 2025 18:56:15 GMT

Three HTTP Error codes encountered are:

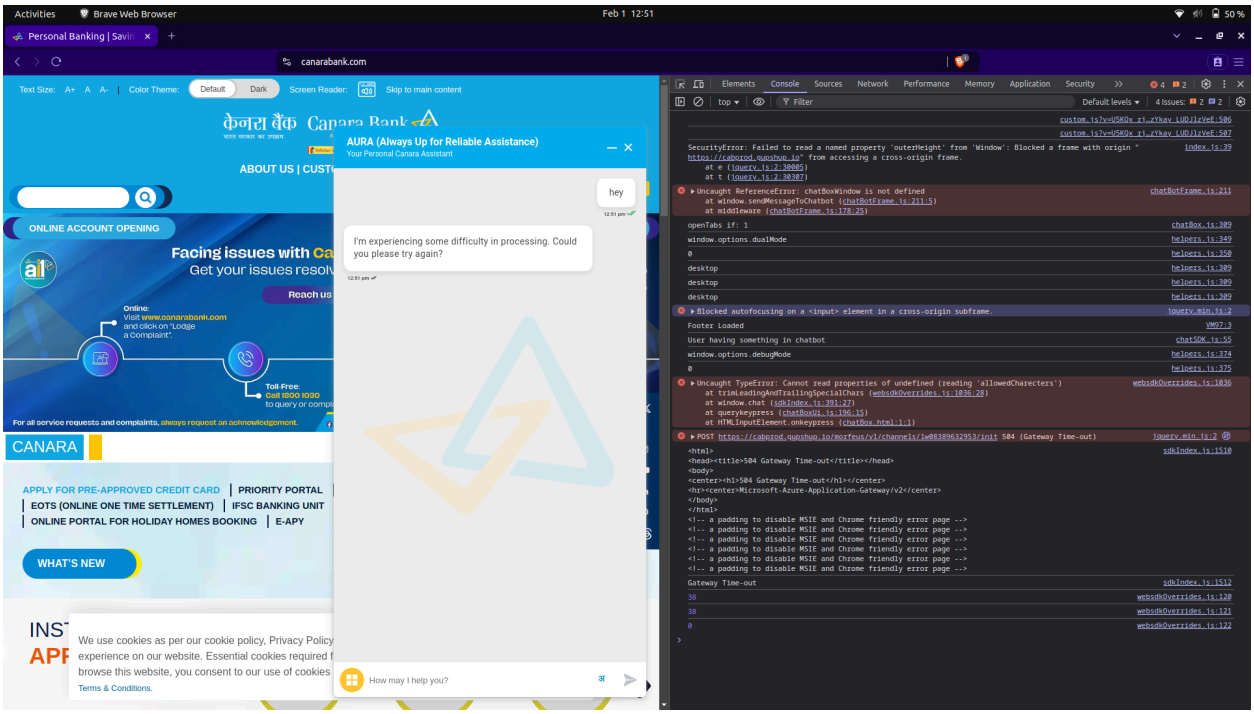
1. **Error 400: Bad Request:** This is a client-side HTTP error and the reason for a 400 response is typically due to malformed request syntax, invalid request message framing, or deceptive request routing.



2. **Error 404: Page Not Found:** This is another client-side HTTP error response code indicating that the server cannot find the requested resource. Links that lead to a 404 page are often called broken or dead links.



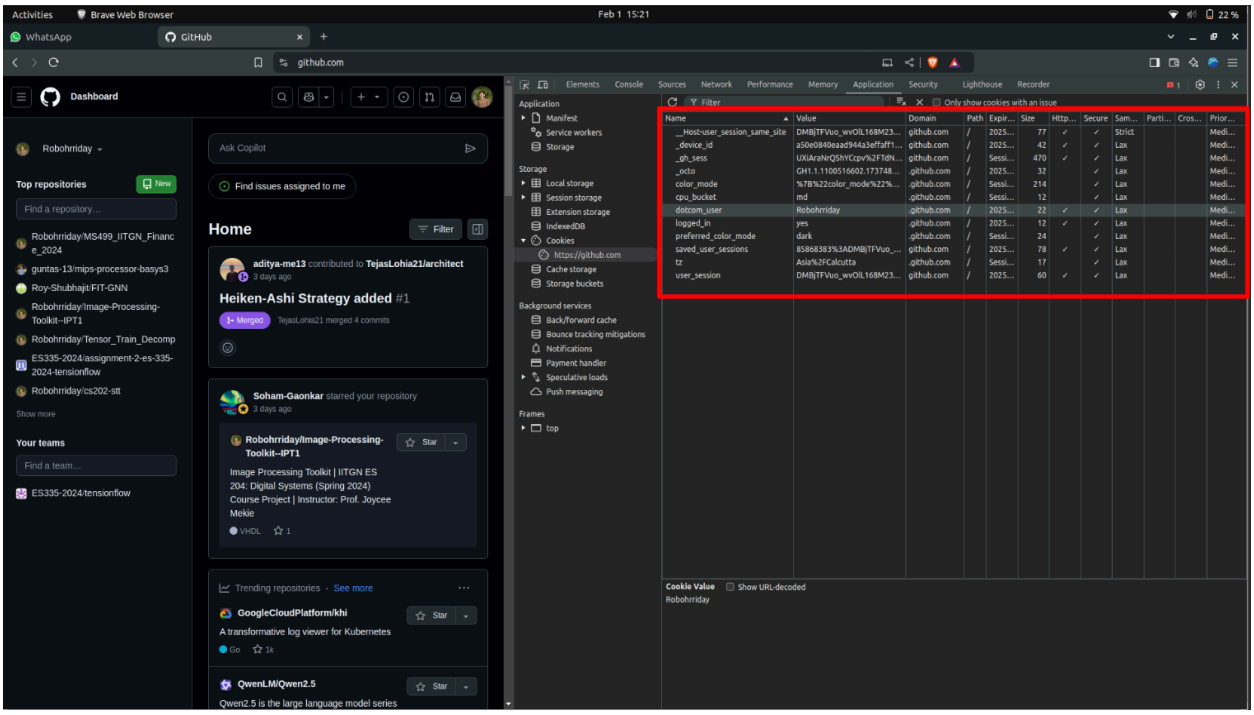
3. **Error 504: Gateway Timeout:** This server error response code indicates that the server, while acting as a gateway or proxy, did not get a response in time from the upstream server in order to complete the request.



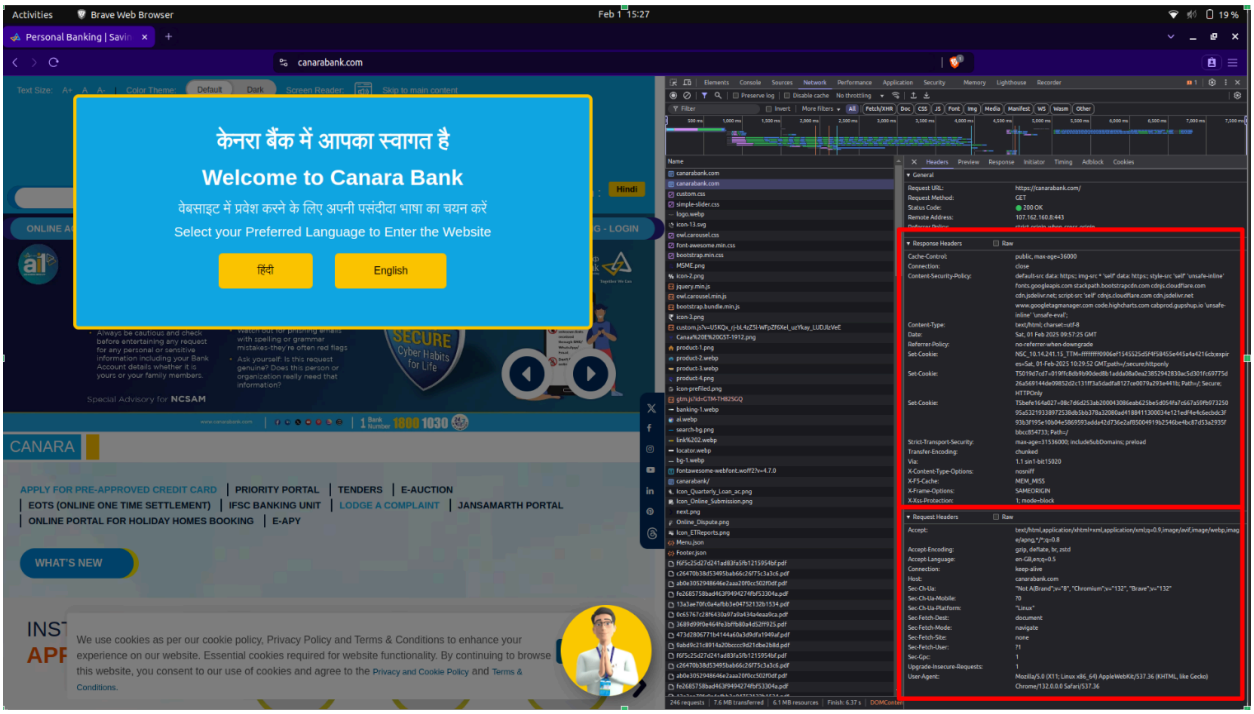
c. Capture the Performance metrics that your browser records when a page is loaded and also report the list the cookies used and the associated flags in the request and response headers. Please report the browser name and screenshot of the performance metrics reported for any one of the page loads.

Browser used: Brave

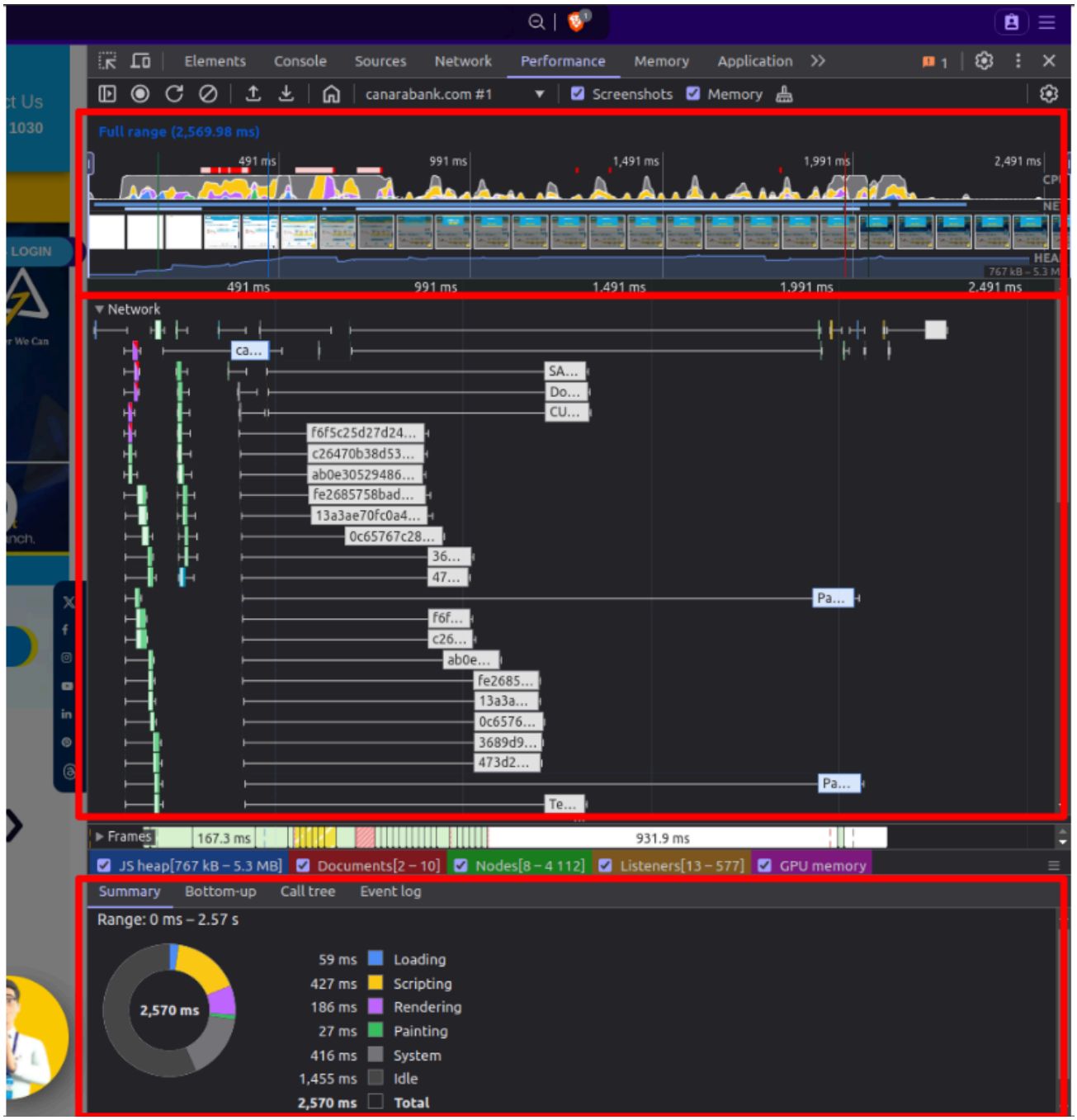
Cookies used for github.com



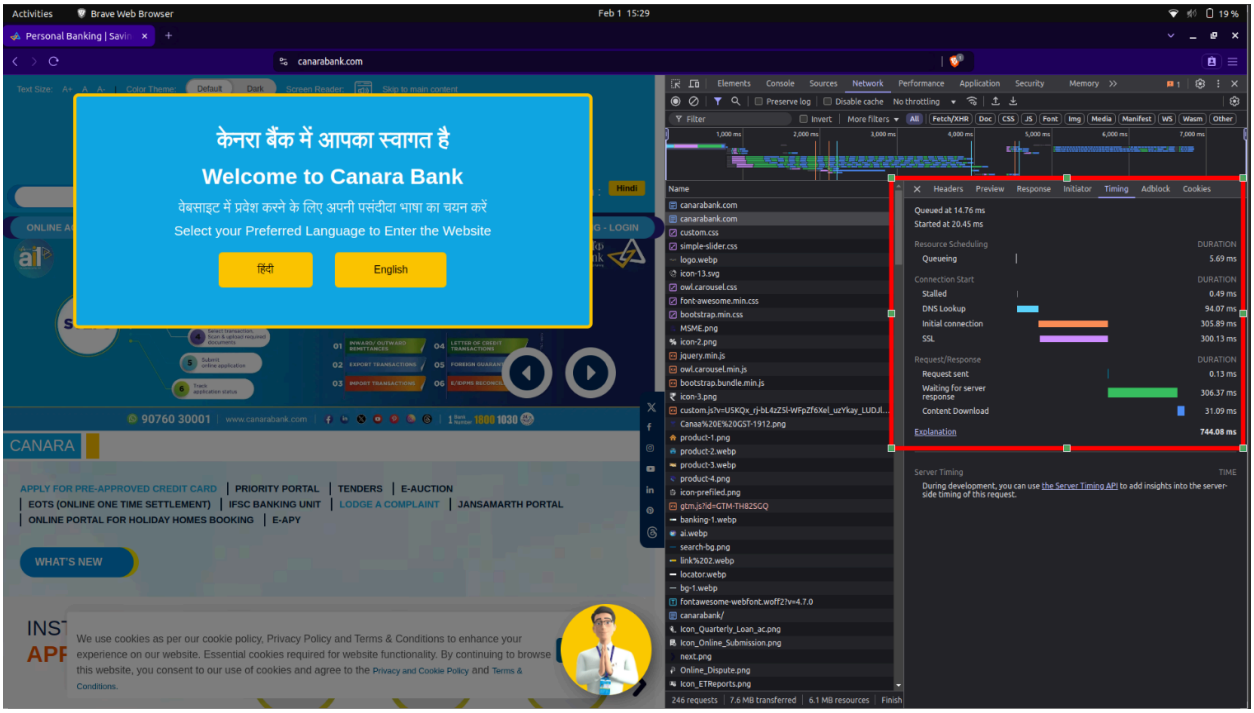
Request and Response Headers of HTTP request to GET main page of canarabank.com



Network performance and summary while loading canarabank.com (entire profile)



Network performance for a single http request while loading canarabank.com



Acknowledgements

Some codes are inspired by GitHub repositories, StackOverflow and ChatGPT.