CS331 – Computer Networks
Assignment 1

Jaidev Sanjay Khalane – 22110103

Sriram Srinivasan – 22110258

Group 13

4.pcap

Contents

Part 1

Part 1: Metrics and Plots (40 pts)

From the chosen *X*.pcap file, extract and generate the following metrics for the data as captured by your program when you perform the pcap replay using tools like tcpreplay:

1. 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).
2. Find unique source-destination pairs (source IP:port and destination IP:port) in the captured data.
3. 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.
4. 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.

Approach

To approach this problem in the right way, I was required to know the details of the protocols used. So, I decided to go from the basics. I first created a raw socket in Python using the sockets module. Since the raw socket gives us raw packets, that is, in our case, the ethernet packet, I disintegrated the packet using the reference [1]. So, I sliced the header of the frame using the Struct module in Python following the format stated in [1]. Then, I analysed the different types of protocols in the next layer, which was the network layer. The Ethertype number from this header was used to find the corresponding protocol used in the network layer. The detailed results are given in the supplementary information section. Using this Ethertype number, I referred to [2] in order to find the corresponding protocols. The major protocols used in this layer were IPv4 and IPv6. I then used [3] to understand the structure of the IPv4 header. It was then disintegrated similarly to find the types of protocols used in the Transport layer. The exact count of the findings is given in the supplementary section. The major types of protocols found were TCP and UDP, with minor traces of other protocols. The headers and payload of TCP and UDP were also disintegrated using [4] and [5] references. A similar approach was also used for ICMP [6] and IPv6 [7].

Once this was done, global lists, sets, and dictionaries were initialised and used throughout the program to capture various required parameters for Part 1. The program also has the feature of automatically ending sniffing after 5 seconds of inactivity, that is, 5 seconds of not receiving any packets automatically. I implemented this using the 'signal' library. On the whole, I used libraries 'socket' [8] in Python for capturing the data, 'struct' [9] for disintegrating the headers, 'signal' [10] for automatically ending the program after 5 seconds of inactivity and 'matplotlib' for plotting the histogram as well as 'time' [11] for finding the total time.

Question 1

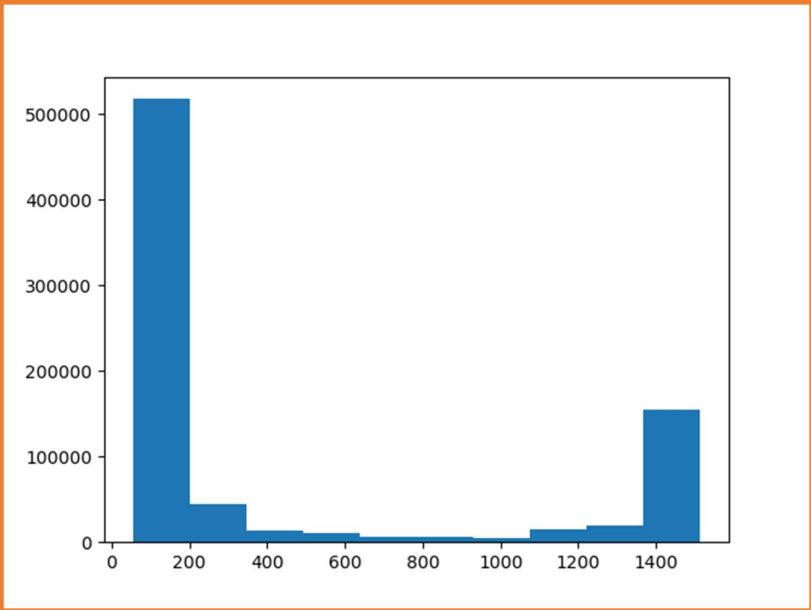
```
Total Packets: 792179

Total Amount of Data Transferred: 355624326 Bytes

Minimum Packet Size: 54 Bytes

Maximum Packet Size: 1514 Bytes
```

```
Average Packet Size: 448.9191533731644 Bytes
```



X-Axis: Size of Packets (Bytes)

Y-Axis: Number of Packets

Question 2

Since the result is too large to be displayed here, I have given the results for this Question in the file Problem1_results2.txt.

Question 3

Source Flows: {'166.131.131.6': 1, '80.239.144.76': 193, …}

Destination Flows: {'80.239.144.76': 222, '81.131.131.6': 208, …}

Since the result is too large to be displayed here, I have given the results for this Question in the file Problem1_results3.txt.

The flow considered here is the number of packets.

Maximum Data Transfer:

Source IP: 172.16.133.95 Source Port: 49358

Destination IP: 157.56.240.102 Destination Port: 443

Question 4

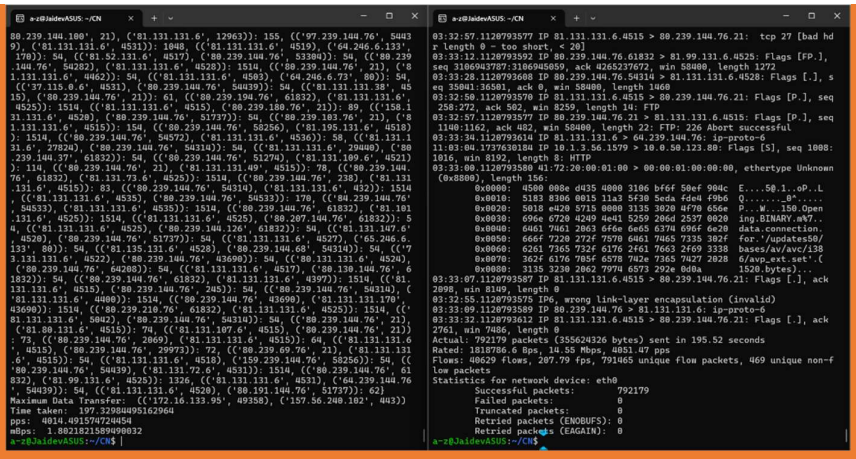
Results for the same VM (Ubuntu 24.04):

Time taken: 197.3298

pps: 4014.491574724454

mBps: 1.8021821589490032

mbps: 14.4174572715920256



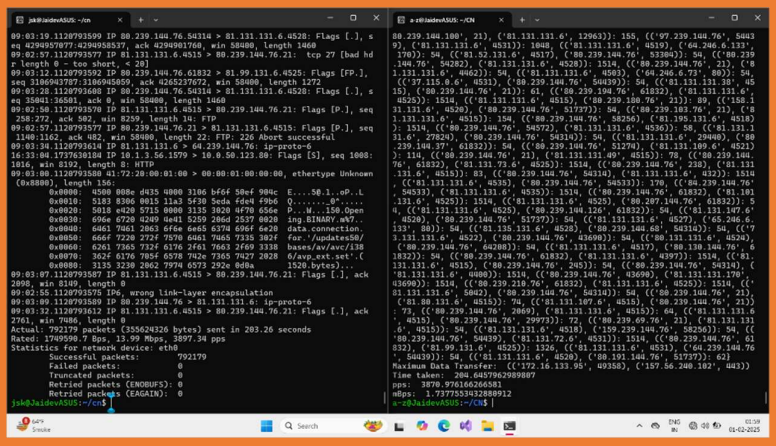
Results for the different VMs (Ubuntu 24.04 and Ubuntu 20.04):

Time taken: 204.6458

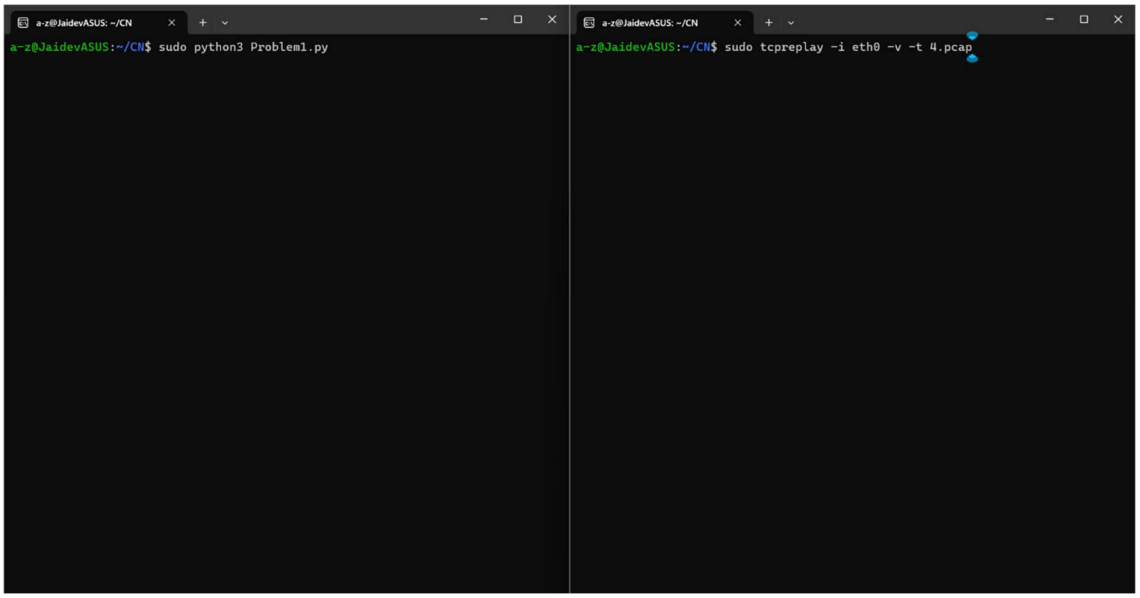
pps: 3870.9761

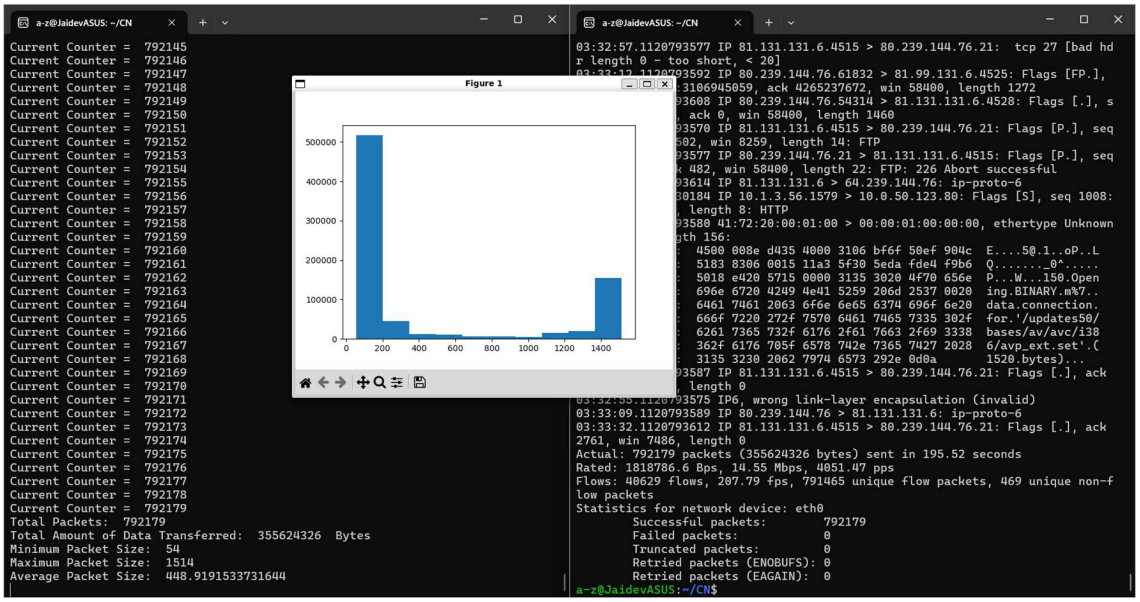
mBps: 1.7377

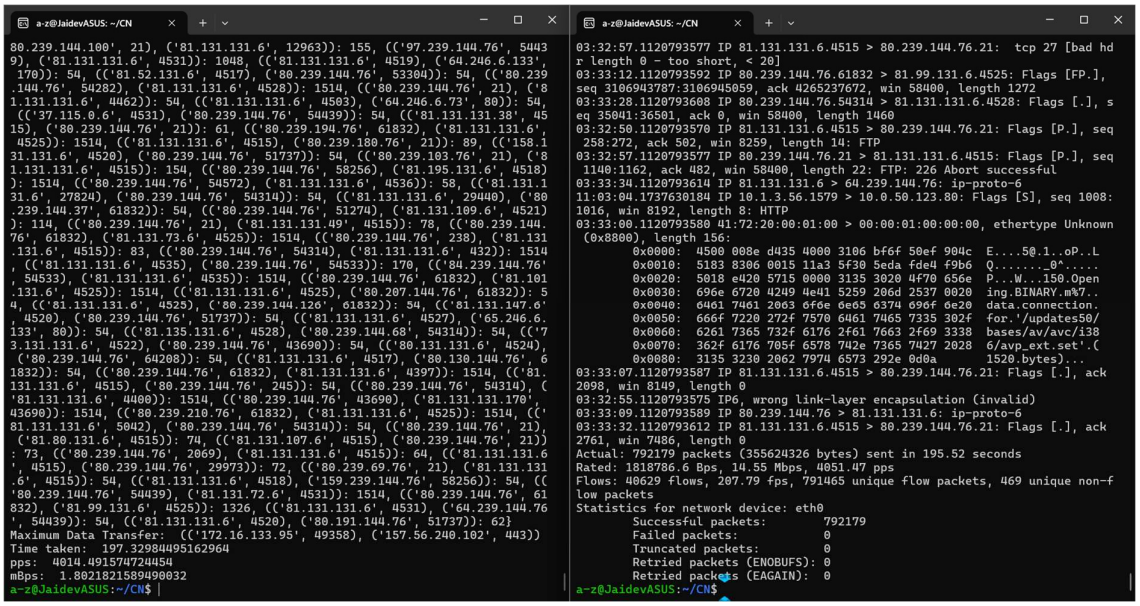
mbps: 13.9016



Supplementary Screenshots for Results







Part 2

4.pcap

Hiding a Message in TCP Packet Payload

Q1.Can you extract the hidden message from the packet payload?

Hint : Filter packet with source port 1579 search keyword CS331.

Q2.How many packets contain the hidden message?

Q3.What protocol is used to transmit the packet containing the hidden message?

Q4.What is the checksum of the TCP segment containing the hidden message?

Approach

While analysing the sniffed packets in the previous problem, instead of storing the parameters like the Source flows, etc., I instead checked for the condition that the source port should be 1579 as given in the hint, and the TCP payload should contain the keyword ‘CS331’. I solved this question assuming the original encoding to be of utf-8 type.

Question 1

Yes, I can extract the hidden message using the hint from the packet payload.

The hidden message is:

```
Welcome to Computer Networks CS331
```

Question 2

A total of `11` packets contain the hidden message.

Question 3

Since this message was first obtained from the raw socket with an ethernet header, the first protocol (at the datalink layer) was Ethernet II. Then, at the Network layer, we had the IPv4 Protocol, and at the Transport layer, the protocol was TCP.

In summary,

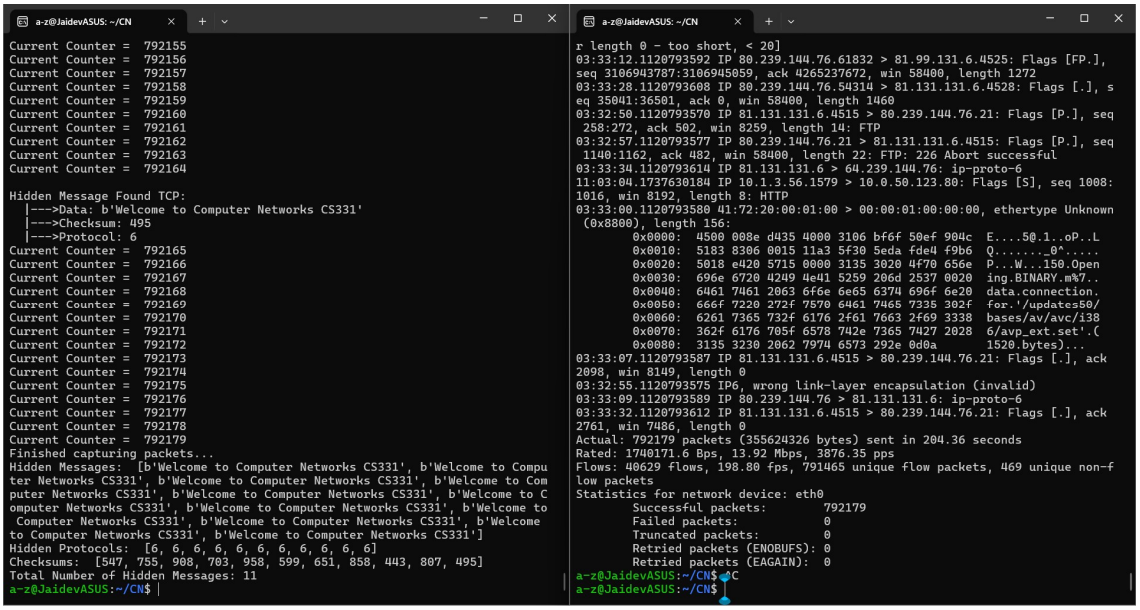
Layer	Protocol
Transport	TCP
Network	IPv4
Datalink	Ethernet II

Question 4

The checksum values are as given below:

547, 755, 908, 703, 958, 599, 651, 858, 443, 807, 495

Supplementary Screenshot for Results



Part 3

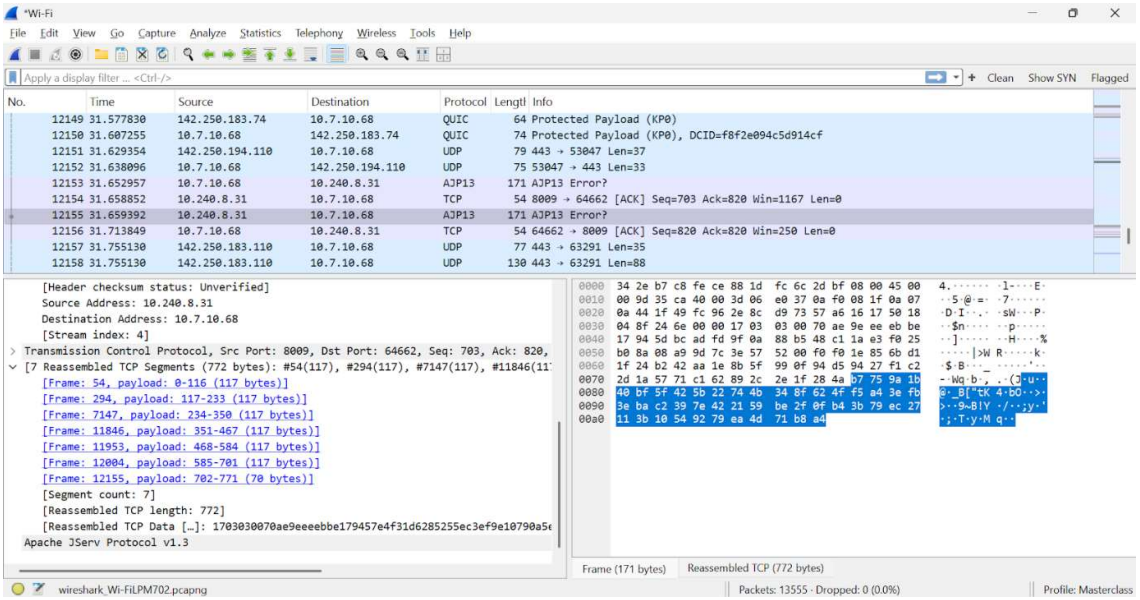
Part 3: Capture the packets (20 points)

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.

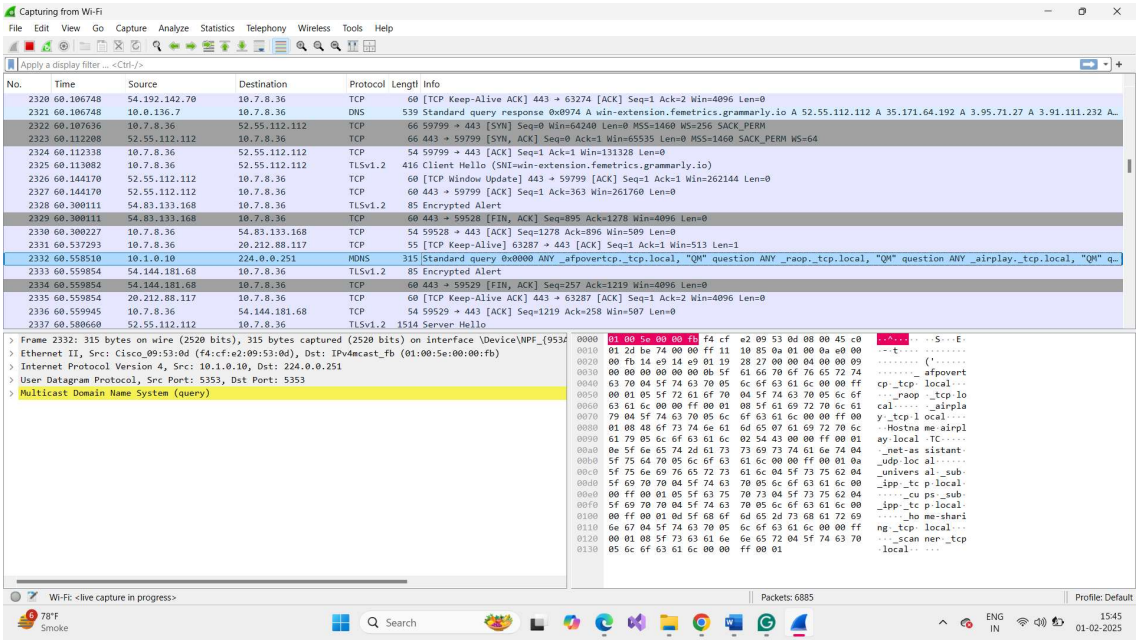
2. Analyze the following details by visiting the following websites in your favourite browser.
   i) canarabank.in
   ii) github.com
   iii) netflix.com
   a. Identify `request line` with the version of the application layer protocol and the IP address. Also, identify whether the connection(s) is/are 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.

Question 1

The first protocol we saw was AJP.

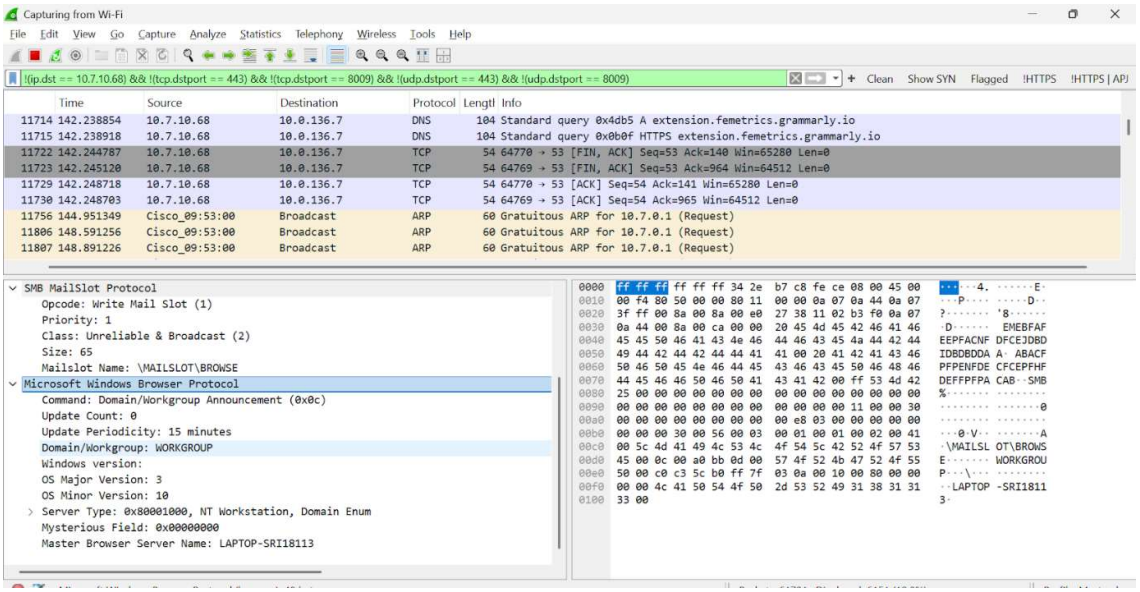


*"The **Apache JServ Protocol** (**AJP**) is a binary protocol that can proxy inbound requests from a web server to an application server behind the web server. It is used at Application layer"* [15]



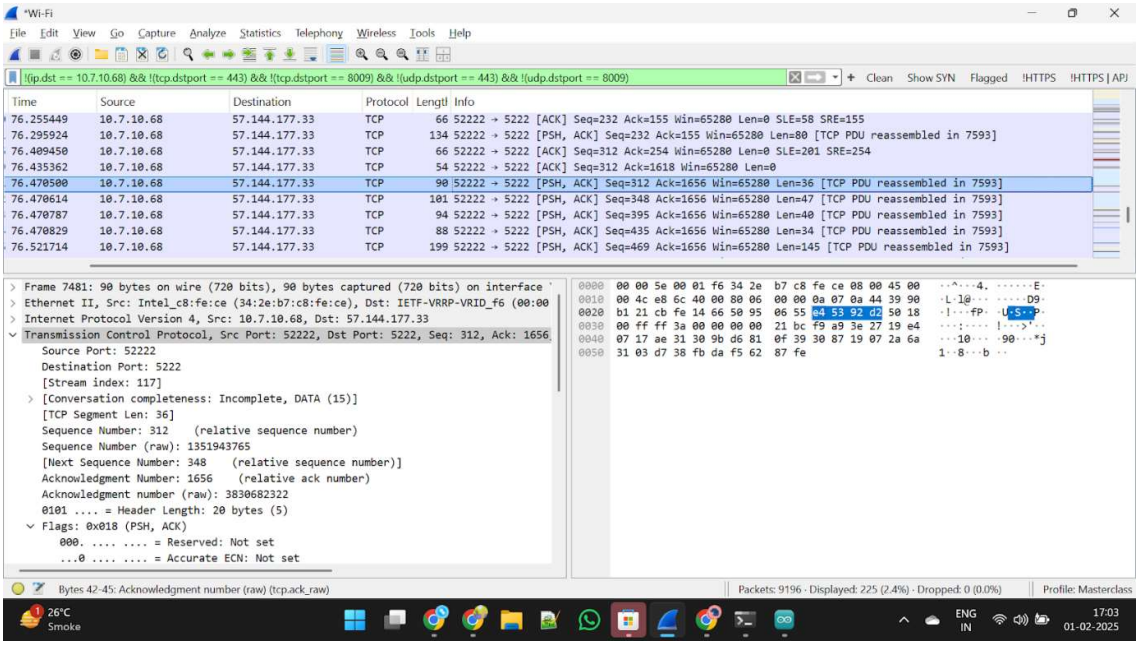
The next protocol was MDNS.

*"**Multicast DNS (mDNS)** is a computer networking protocol that resolves hostnames to IP addresses within small networks that do not include a local name server. It works at the Application layer, similar to DNS. RFC 6763."* [16]

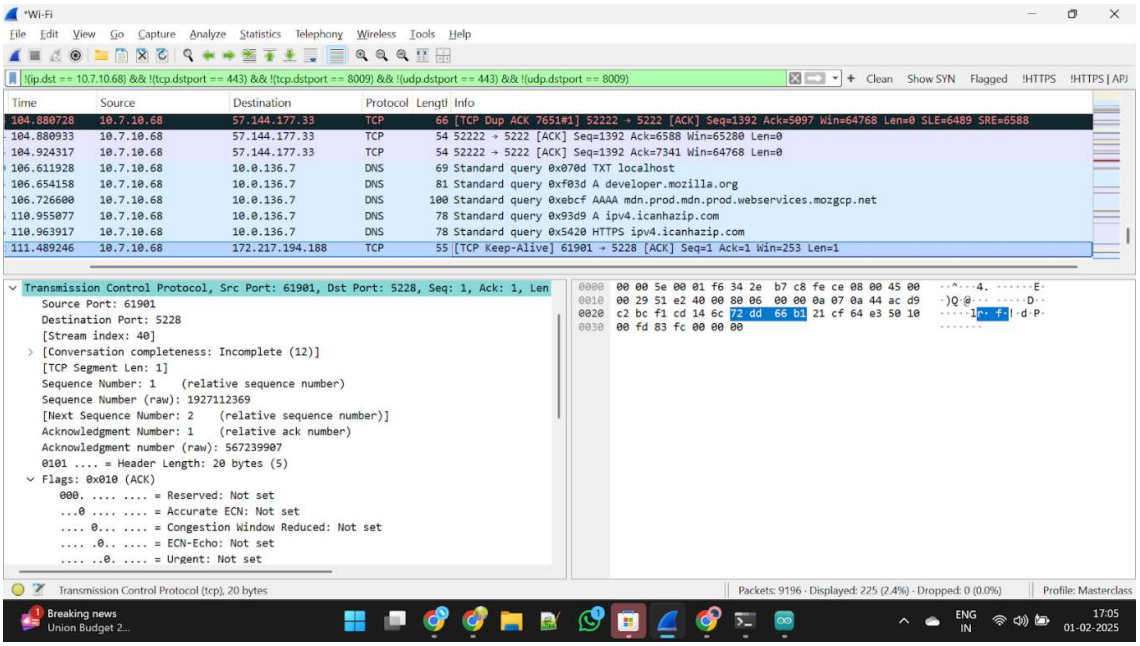


*"The **Server Message Block (SMB)** Protocol is a network file sharing protocol, and as implemented in Microsoft Windows, it is known as Microsoft SMB Protocol. It works at Application Layer."* [17]

"***Microsoft Windows Browser Protocol*** *is also an application layer protocol, and it is used in browser applications on Microsoft Windows Devices (using Microsoft Browser)"* [18]



*"XMPP, **Extensible Messaging and Presence Protocol,** is an open communication protocol designed for instant messaging (IM), presence information, and contact list maintenance. It operates on Port 5222. It was defined in RFC 6120."* [19]

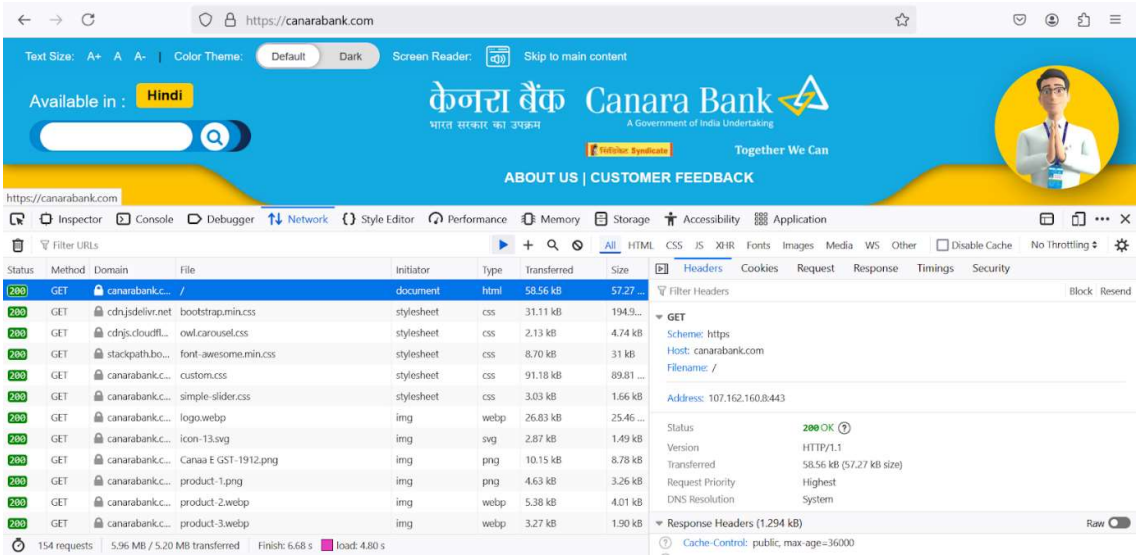


*"**HP Virtual Rooms Protocol** uses Port 5228. It works on the application layer and is used for facilitating online team meetings."* [20]

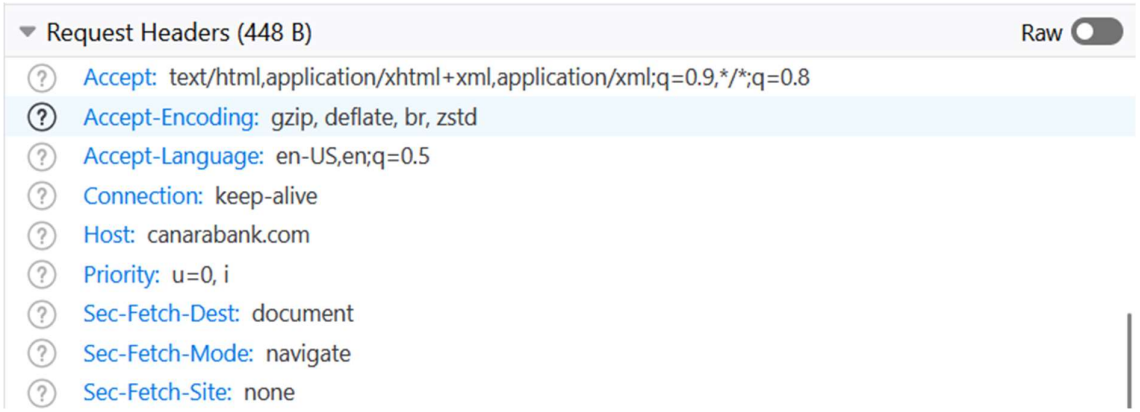
Question 2

canarabank.in did not exist. So, we used canarabank.com.

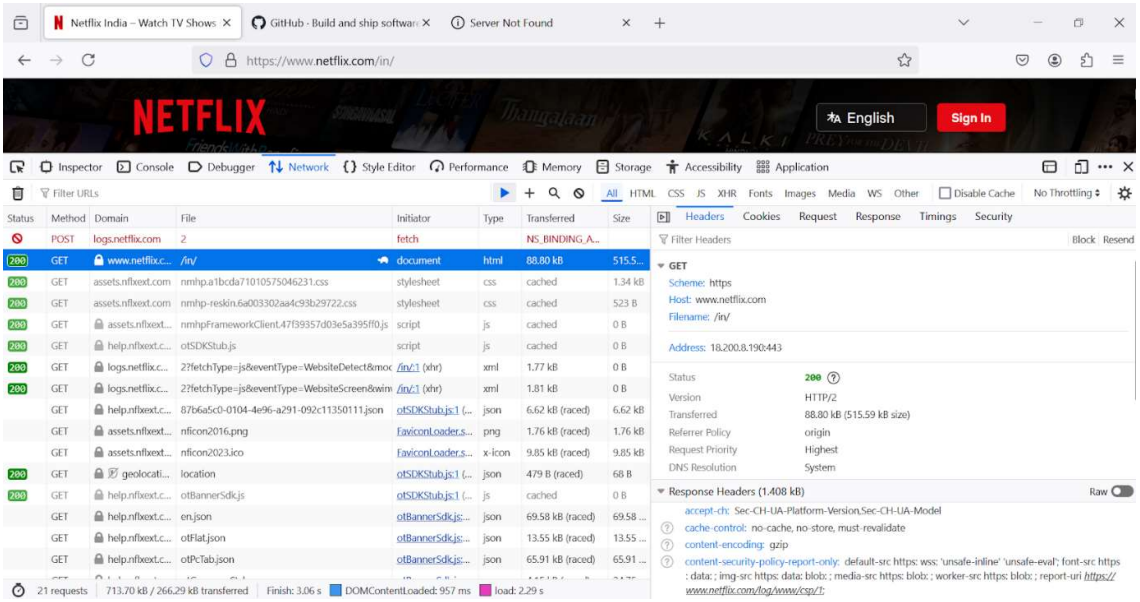
Part (a)



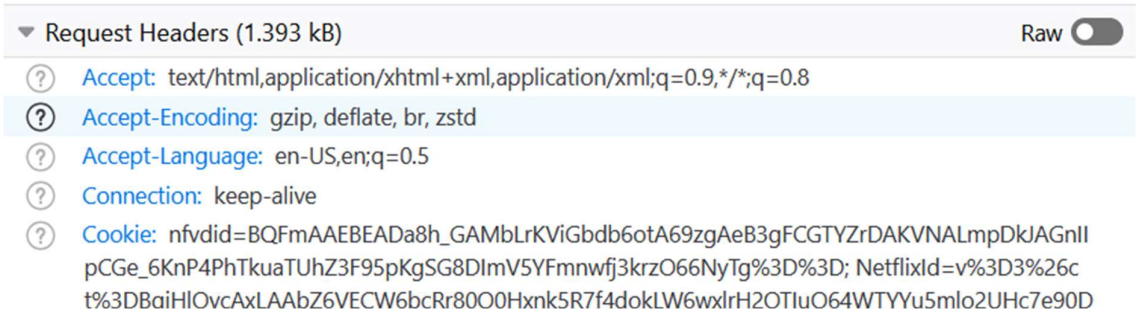
From the above snippet, it's clear that the application layer protocol is HTTP (Version 1.1) and the IP address of the server is 107.162.160.8 running on port 443 (corresponding to Secure HTTP).



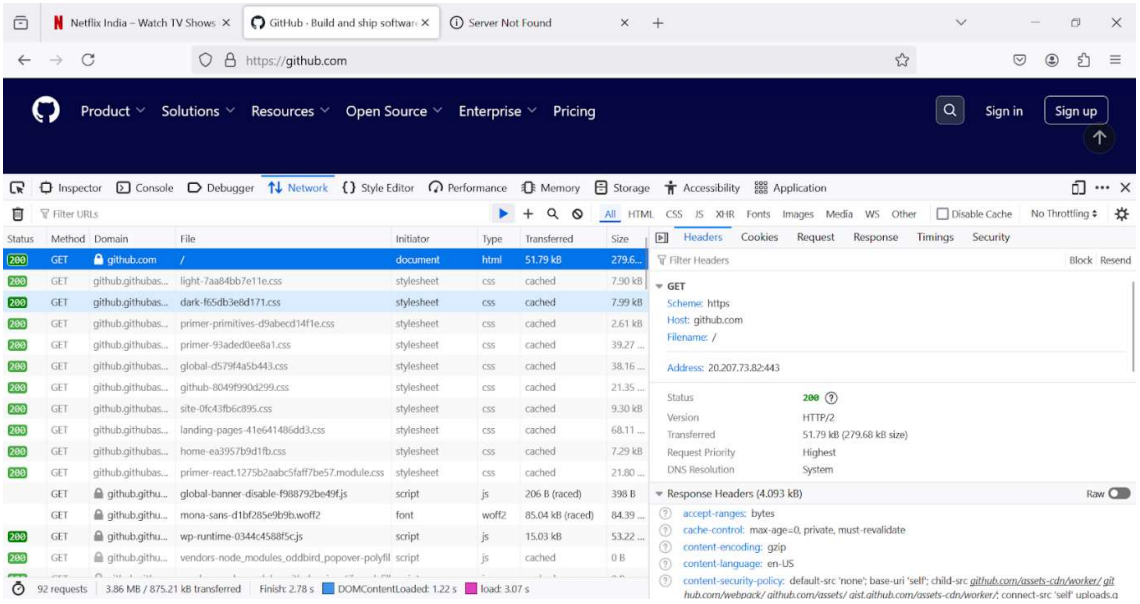
The connection is persistent as the value of the *connection parameter is 'keep-alive'*.



From the above snippet, it's clear that the application layer protocol is HTTP (Version 2) and the IP address of the server is 18.200.8.190 running on port 443 (corresponding to Secure HTTP).



The connection is persistent as the value of the *connection parameter is 'keep-alive'*.

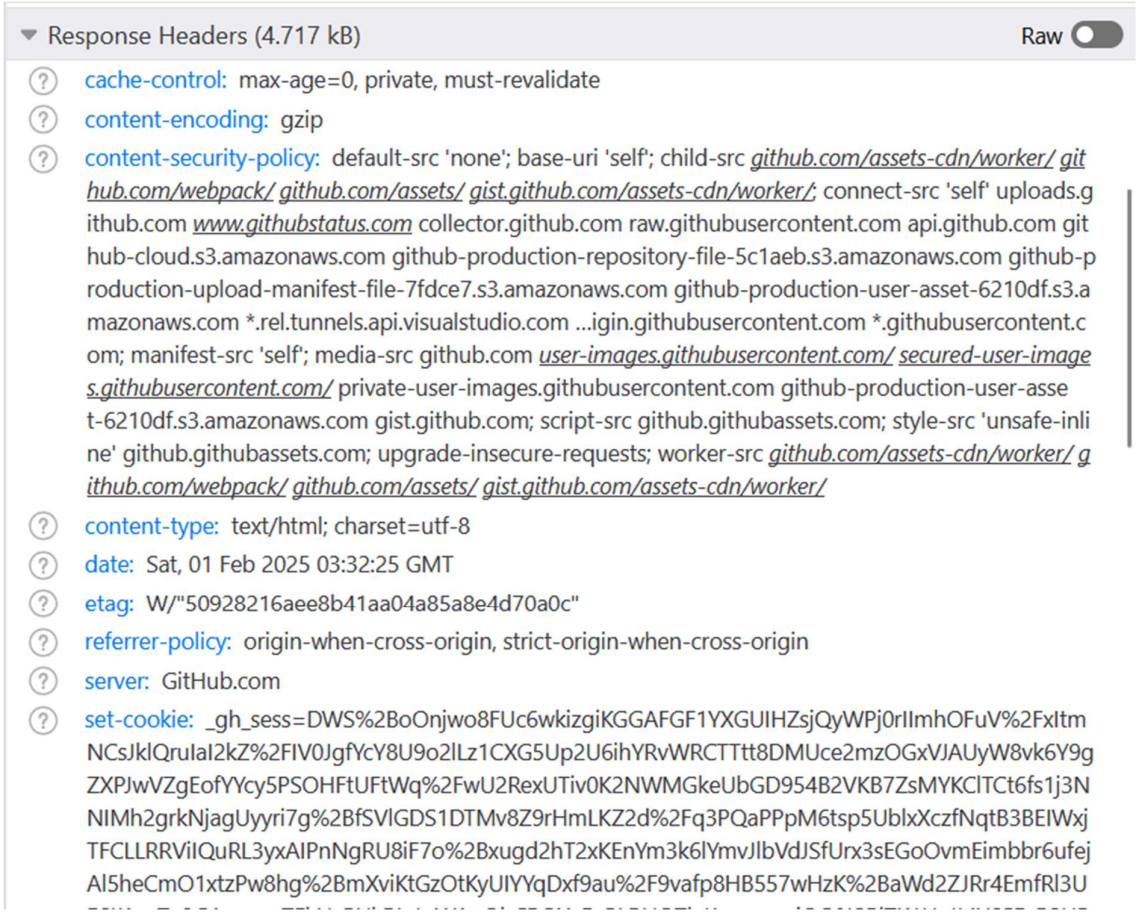


From the above snippet, it's clear that the application layer protocol is HTTP (Version 2) and the IP address of the server is 20.207.73.82 running on port 443 (corresponding to Secure HTTP).

The connection is persistent as the value of the *connection parameter is 'keep-alive'*.

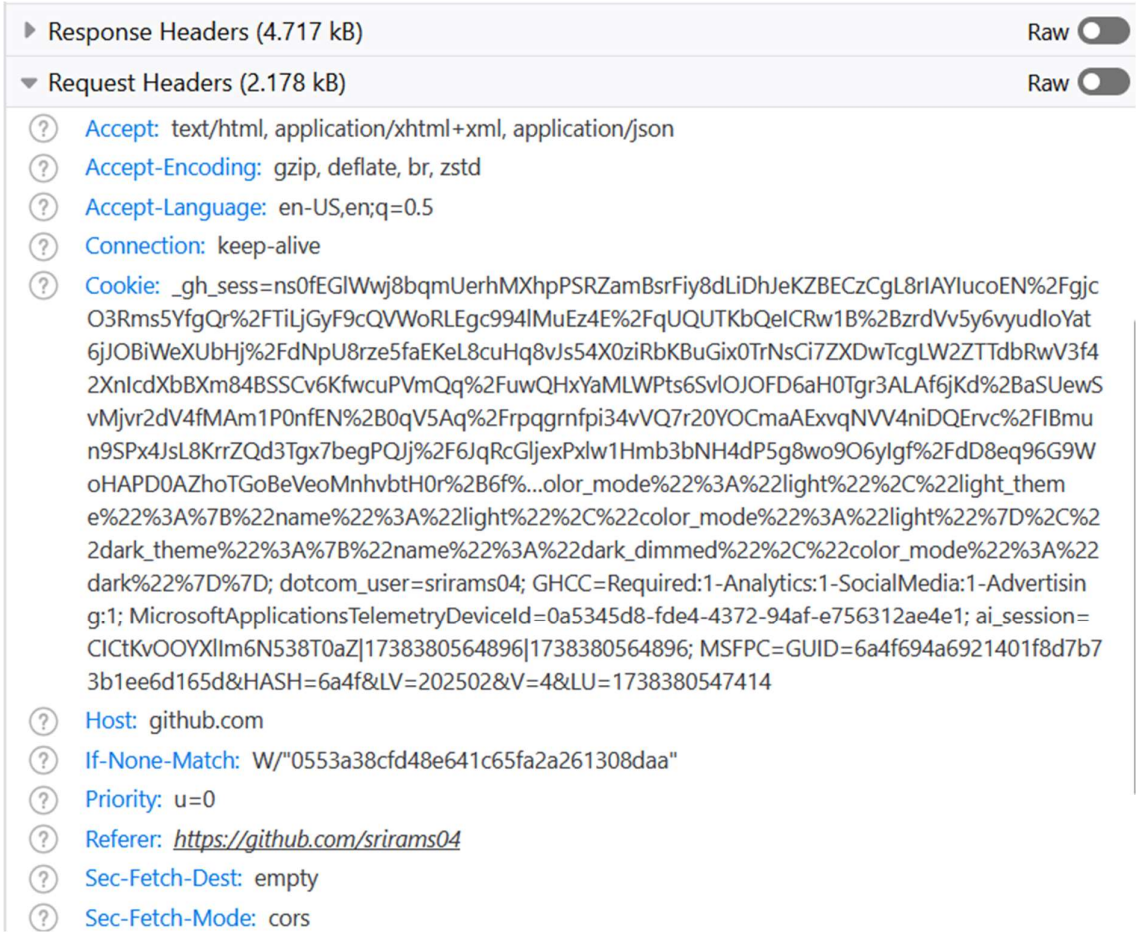
Part (b)

The below analysis was performed on the *github.com* website.



Response Headers:

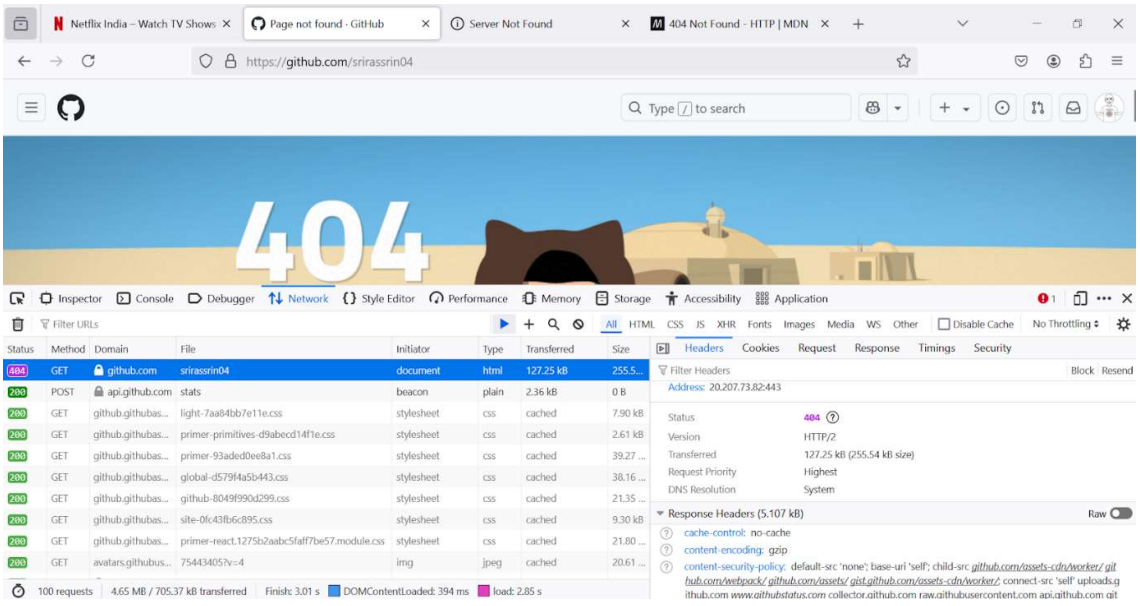
- cache-control: max-age=0, private, must-revalidate
- content-encoding: gzip
- date: Sat, 01 Feb 2025 03:32:25 GMT



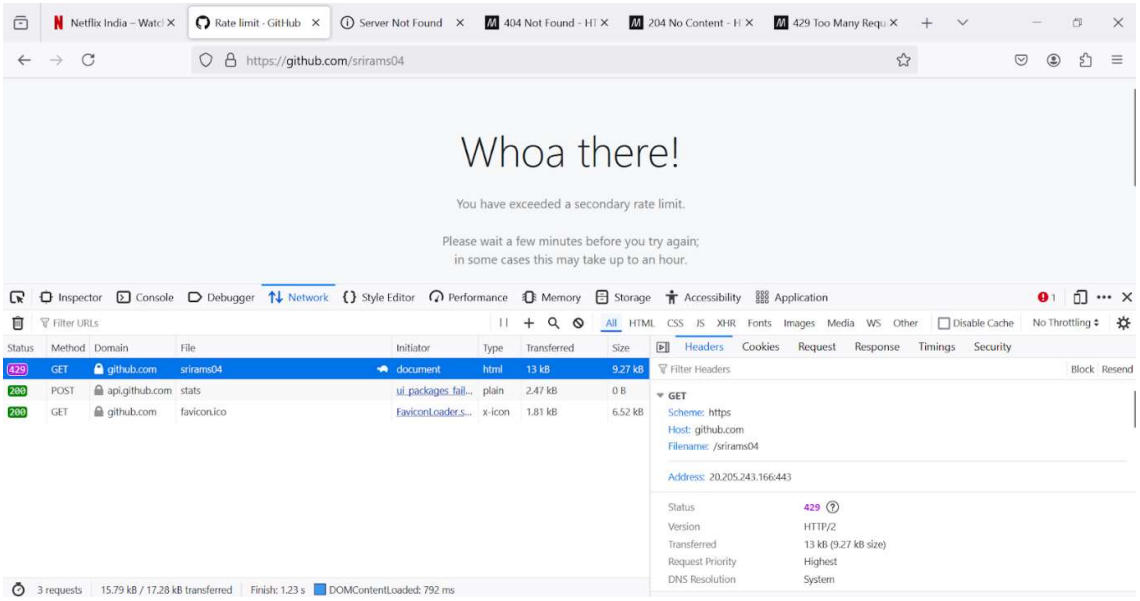
Request Headers:

- Accept-language: en-US,en;q=0.5
- Accept-encoding: gzip, deflate, br, zstd
- Priority: u=0

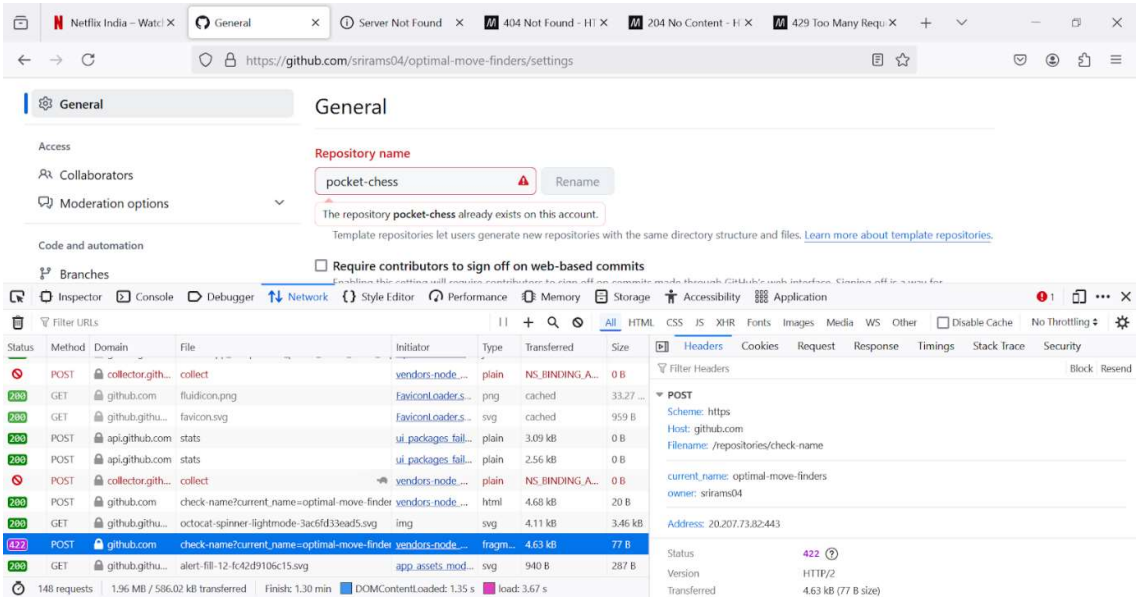
Errors:



*"The HTTP **404 Not Found** client error response status code indicates that the server cannot find the requested resource. Links that lead to a 404 page are often called broken or dead links and can be subject to link rot. A 404 status code only indicates that the resource is missing without indicating if this is temporary or permanent. If a resource is permanently removed, servers should send the 410 Gone status instead."* [12]



*"The HTTP **429 Too Many Requests** client error response status code indicates the client has sent too many requests in a given amount of time. This mechanism of asking the client to slow down the rate of requests is commonly called "rate limiting". A Retry-After header may be included to this response to indicate how long a client should wait before making the request again."* [13]

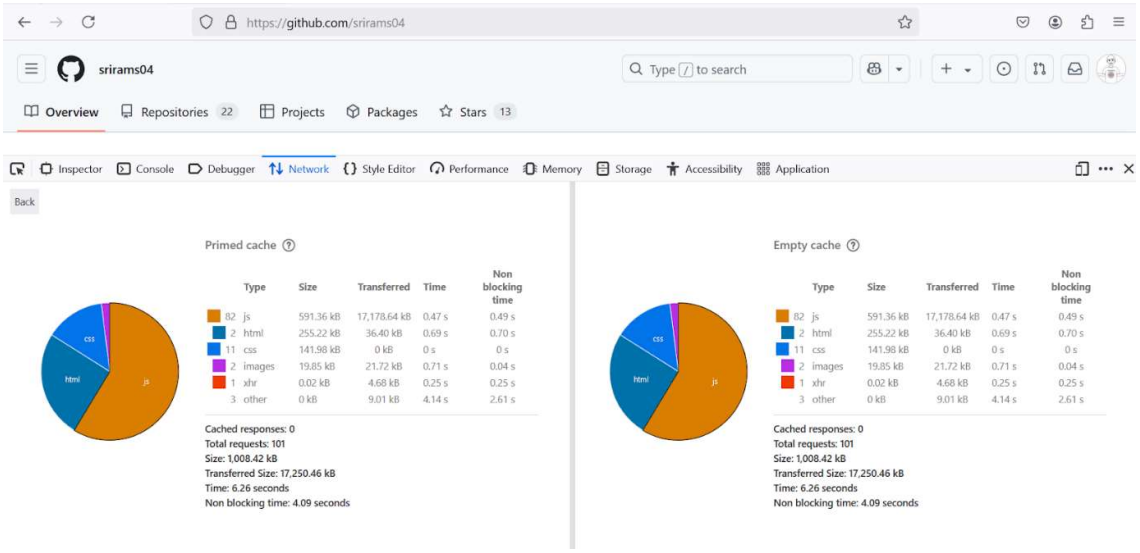


*"The HTTP **422 Unprocessable Content** client error response status code indicates that the server understood the content type of the request content, and the syntax of the request content was correct, but it was unable to process the contained instructions. Clients that receive a 422 response should expect that repeating the request without modification will fail with the same error."* [14]

Part (c)

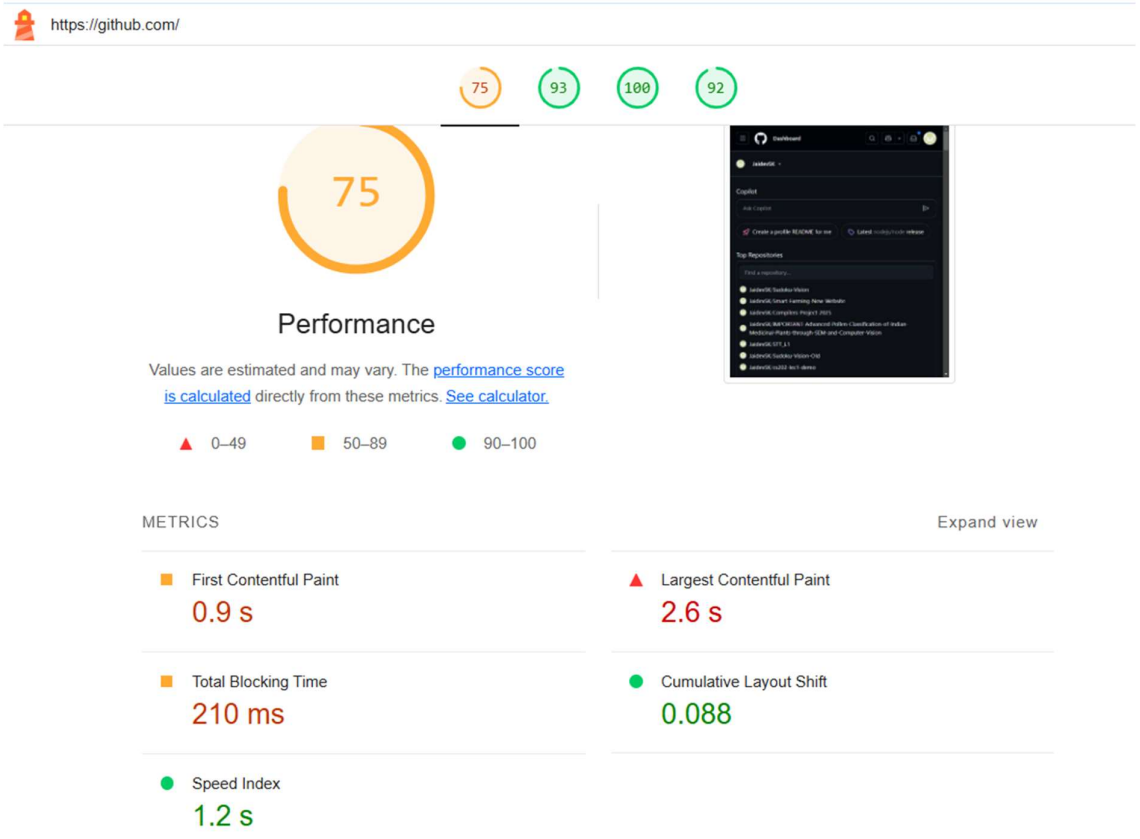
Browser Name: Mozilla Firefox

Screenshots of Performance Metrics:

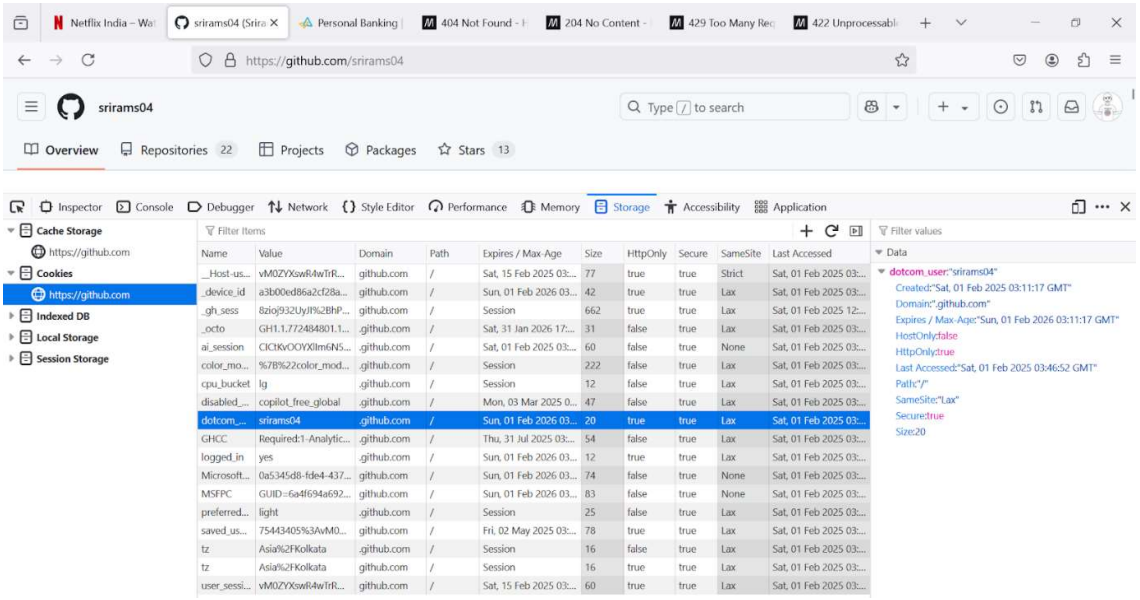


Browser Name: Chrome

Screenshots of Performance Metrics:



Cookies:



All the header-value pairs for the selected cookie are visible on the right-hand side of the above image. The image below displays the flags associated with the request and response headers on reloading github.com.

Response:

strict-transport-security: max-age=31536000; includeSubdomains; preload
vary: X-PJAX, X-PJAX-Container, Turbo-Visit, Turbo-Frame
vary: Accept-Encoding, Accept, X-Requested-With
x-content-type-options: nosniff
X-Firefox-Spdy: h2
x-frame-options: deny
x-github-request-id: 8914:0A6F:5F35AF:7CB2FC:679D9919
x-xss-protection: 0

Request:

Host: github.com
Priority: u=4
Referer: *https://github.com/srirams04*
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
TE: trailers
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
X-Requested-With: XMLHttpRequest

References

[1] https://en.wikipedia.org/wiki/Ethernet_frame

[2] https://en.wikipedia.org/wiki/EtherType

[3] https://en.wikipedia.org/wiki/IPv4

[4] https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[5] https://en.wikipedia.org/wiki/User_Datagram_Protocol

[6] https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

[7] https://en.wikipedia.org/wiki/IPv6

[8] https://docs.python.org/3/library/socket.html

[9] https://docs.python.org/3/library/struct.html

[10] https://docs.python.org/3/library/signal.html

[11] https://docs.python.org/3/library/time.html

[12] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404

[13] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/429

[14] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422

[15] https://en.wikipedia.org/wiki/Apache_JServ_Protocol

[16] https://en.wikipedia.org/wiki/Multicast_DNS

[17] https://learn.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview

[18] https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-brws/3cfbad92-09b3-4abc-808f-c6f6347d5677

[19] https://en.wikipedia.org/wiki/XMPP

[20] https://www.hp.com/hpinfo/newsroom/press_kits/2009/domorewithless/HPVirtualRooms.PDF

[21] https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Supplementary Documents and Data

I have also provided the supplementary code, which displays my stepwise approach to analyse the problem. The data obtained in each step is also provided as comments in the remaining part of the program files.