

ASSIGNMENT 1: RAW SOCKETS

CS 331: COMPUTER NETWORKS

Team 2:

Shubham Agrawal - 22110249

Vraj Shah - 22110292

Supervisor:

Prof. Sameer G. Kulkarni

Indian Institute of Technology Gandhinagar

Saturday 1st February, 2025

Contents

1	Introduction	1
2	Packet Sniffer Implementation	1
3	Extension of Packet Sniffer Implementation	1
4	Part 1: Metrics and Plots	1
4.1	Total Data and Packet Statistics	1
4.2	Unique Source-Destination Pairs	2
4.3	Source IP and total flow	3
4.4	Destination IP and total flow	3
4.5	Maximum Flow Analysis	3
4.6	Top Speed Capture without Data Loss	4
5	Part 2: Capture the Flag, PCAP specific questions	5
6	Part 3: Real-Time Packet Capture	7
6.1	Wireshark Packet Capture	7
6.2	Website Packet Analysis	10
7	Files and Code Location	14
8	Replicating the Results	15
8.1	Clone the repository	15
8.2	Install dependencies	15
8.3	Install PCAP File	15
8.4	Set up your environment	15
8.5	Run the packet sniffing script	15
8.6	Run tcpreplay	16
8.7	Check results	16

1 Introduction

This report outlines the work for CS 331: Computer Networks Assignment #1. The task required the implementation of a packet sniffer and the examination of packet capture (PCAP) files. The subsequent sections detail the computed metrics, responses to specific PCAP queries, and the process of real-time packet capturing.

2 Packet Sniffer Implementation

The packet sniffer was implemented in Python using socket programming. It was designed to capture packets from the specified network interface and perform real-time analysis of the network packets.

3 Extension of Packet Sniffer Implementation

The same python program was extended, which utilized the original sniffer to capture packets and process them to generate various required matrices as mentioned in part 1 of the Assignment and then subsequently answers the pcap related queries as required in part 2 of the Assignment.

4 Part 1: Metrics and Plots

4.1 Total Data and Packet Statistics

The program analyzed the replayed 2.pcap file [2 mod 9 = 2] to get the following inferences:

- **Total Data Transferred:** 364642055 bytes
- **Total Packets Transferred:** 805997 packets
- **Minimum Packet Size:** 42 bytes
- **Maximum Packet Size:** 1514 bytes
- **Average Packet Size:** 452.4112 bytes

The following figure shows a histogram showing the distribution of packet sizes:

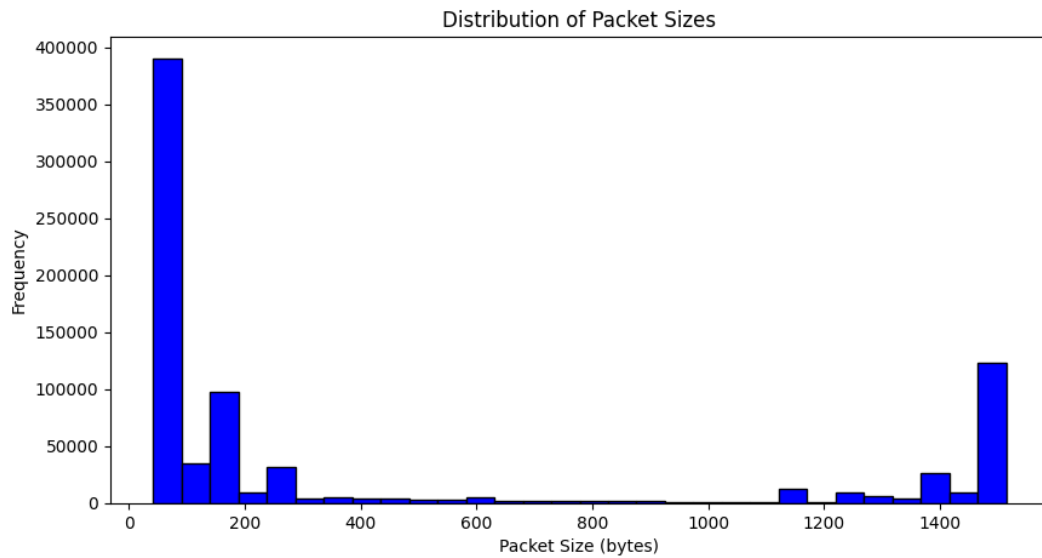


Figure 1: Histogram of packet size vs frequency

```
---Question 1---
Total Packets: 805997
Total Data Transferred: 364642055 bytes
Min Packet Size: 42 bytes
Max Packet Size: 1514 bytes
Average Packet Size: 452.4111814311964 bytes
```

Figure 2: Result: Part-1 Q1

4.2 Unique Source-Destination Pairs

The unique source-destination IP pairs were identified and stored in a text file named `unique_pairs.txt`, included in the GitHub repository. Only IP packets having TCP or UDP layers were considered. The stored contents are in the form `source-ip:port → destination-ip:port`.

```

---Question 2---
Unique Source-Destination IP Pairs
No. of Unique Source-Destination Pairs: 41056
Showing only 5 Entire here (refer to unique_pair.txt for all)
('205.188.93.194', 80, '172.16.133.54', 64598)
('172.16.133.97', 59523, '172.16.139.250', 5440)
('172.16.133.6', 57663, '8.8.8.8', 53)
('96.43.146.22', 443, '172.16.133.41', 51452)
('8.8.8.8', 53, '172.16.133.6', 57918)

```

Figure 3: Result: Part-1 Q2

4.3 Source IP and total flow

Unique source IP's and their corresponding total flows were stored in a text file named `unique_src_ip.txt`, included in the GitHub repository. Only IP packets were considered. The stored contents are in the form `source-ip : total flow count`.

4.4 Destination IP and total flow

Unique destination IP's and their corresponding total flows were stored in a text file named `unique_dst_ip.txt`, included in the GitHub repository. Only IP packets were considered. The stored contents are in the form `destination-ip : total flow count`.

4.5 Maximum Flow Analysis

To identify the source-destination pair (source IP:port and destination IP:port) that transferred the most data, a dictionary, `flow_data`, was maintained where the key represents the source-destination pair and the value stores the cumulative data transfer for that pair.

After updating the dictionary with the size of each transferred packet for each source-destination pair, we identified the source-destination pair with the highest total transfer.

Maximum Flow: The source IP:port and destination IP:port pair that transferred the most data is `172.16.133.95:49358 → 157.56.240.102:443`, with a total of 17342229 bytes transferred.

```

---Question 3---
Flow Counts Per Source IP
No. of Unique Source IP: 2077
Showing only 5 Entire here (refer to unique_src_ip.txt for all)
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

Flow Counts Per Destination IP
No. of Unique Destination IP: 2004
Showing only 5 Entire here (refer to unique_dst_ip.txt for all)
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

Maximum Flow
Maximum Flow: 172.16.133.95:49358 -> 157.56.240.102:443 transferred 17342229 bytes

```

Figure 4: Result: Part-I Q3

4.6 Top Speed Capture without Data Loss

The performance of the program was evaluated by capturing the content without any loss of data, under the following two conditions:

1. Running both tcpreplay and the sniffer program on the same machine (VM)
2. Running tcpreplay on one machine and the sniffer program on a different machine.

In both cases, the performance was measured in terms of the number of packets per second (pps) and the bandwidth in megabits per second (mbps) to determine the maximum speed at which the program could successfully capture all the data without the loss of any bytes/packets.

```

shubham@4099@ShubhamAg-PC:~/mnt/d/Sem-2/Computer Networks/Assignment-1$ sudo python3
temp2.py -i eth0
Listening on eth0
Duration of 75 seconds reached. Exiting...

===== Part-1 =====

---Question 1---
Total Packets: 805997
Total Data Transferred: 364642055 bytes
Min Packet Size: 42 bytes
Max Packet Size: 1514 bytes

shubham@4099@ShubhamAg-PC:~/mnt/d/Sem-2/Computer Networks/Assignment-1$ sudo tcpreplay
-i eth0 --pps 14000 2.pcap
Actual: 805997 packets (364642055 bytes) sent in 57.57 seconds
Rated: 6333761.9 Bps, 50.67 Mbps, 14000.01 pps
Flows: 41719 flows, 724.65 fps, 805298 unique flow packets, 454 unique non-flow pack
ets
Statistics for network device: eth0
Successful packets:      805997
Failed packets:         0
Truncated packets:      0
Retried packets (ENBUFS): 0
Retried packets (EAGAIN): 0

```

Figure 5: Top Speed Same Machine

Figure 6: Top Speed Different Machine

The results obtained on different machines are not identical to previous observations due to the use of an Ethernet port, which inherently transmits additional packets. This variation in packet transmission can lead to discrepancies in the captured data across different systems.

The results of these tests are shown in the table below:

Scenario	PPS	Mbps
Same Machine	14000	50.67
Different Machines	5000	18.09

Table 1: Top Speed Results

*The top speed varied every time, this is an approximation pertaining to the last few runs.

5 Part 2: Capture the Flag, PCAP specific questions

In this section, we address the following questions based on the TCP packet analysis.

Q1. *There is a message in TCP packets which contains my IP address. Find that IP address.*

Answer: The IP address was identified by searching for the string “My ip address = ” in the payload of the TCP packets. The IP address was identified as 10.1.2.200.

Q2. *Find the number of packets with that IP address.*

Answer: After identifying the IP address, the number of packets containing this IP address, either as a source or as a destination was counted. The total number of packets with this IP address was found out to be 80.

Q3. *I have sent the name of the laptop in one of the TCP packets.*

1. Find the name of the laptop.

Answer: The name of the laptop was found in the payload of the TCP packet by searching for the string “The name of laptop =”. The laptop name was found to be lenovo.

```
==== Part-2 ====

---Question 1---
My IP Address: 10.1.2.200

---Question 2---
Packets containing IP 10.1.2.200: 80

---Question 3---
Laptop Name: lenovo
TCP Checksum: 0xa61

---Question 4---
Packets containing 'Order Successful': 40
```

Figure 7: Result: Part-2

2. Find the TCP checksum of that packet.

Answer: The TCP checksum for the packet containing the laptop name was calculated and found to be 0x0a61.

Q4. *Find the number of packets which contain the message "Order Successful".*

Answer: By searching for the string "Order Successful" in the payload of the TCP packets, the total number of packets containing the message was determined to be 40 packets.

***The network and its contents were analyzed using Wireshark while replaying using tcpreplay to identify the exact string that needed to be searched for in the Python code.** This approach allowed us to inspect the raw packet data and extract the relevant string patterns, ensuring the correct matches were identified during the analysis.

6 Part 3: Real-Time Packet Capture

In this section, we capture and analyze network packets using the Wireshark tool to gain insights into various application layer protocols and network performance.

6.1 Wireshark Packet Capture

1(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 the protocol and its layer of operation, along with the associated RFC number if any.

Answer: Some unique application-layer protocols observed are:

1. **SSHv2:** The Secure Shell (SSH) version 2 is a cryptographic network protocol used for secure data communication. It is primarily used for remote login to computer systems and for secure file transfer. SSHv2 provides strong authentication and encrypted data communication, replacing older protocols like Telnet and rlogin. **Associated Document:** RFC 4251.

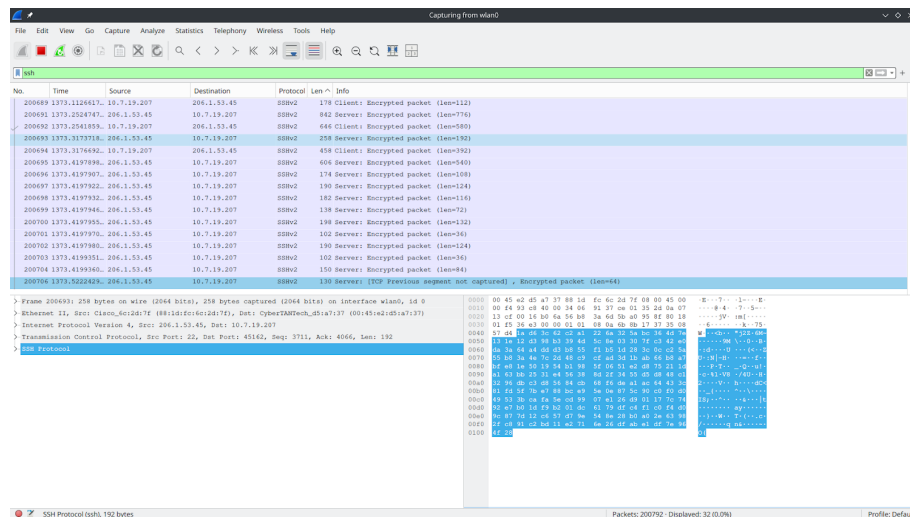


Figure 8: SSHv2

2. **SMB2:** Server Message Block version 2 (SMB2) is a network file sharing protocol that allows applications to read and write to files and request services from server programs in a computer network. It is an improved version of the SMB protocol, providing faster file sharing, and enhanced security and reliability. **Associated Document:** *Microsoft Document titled '[MS-SMB2]'.*

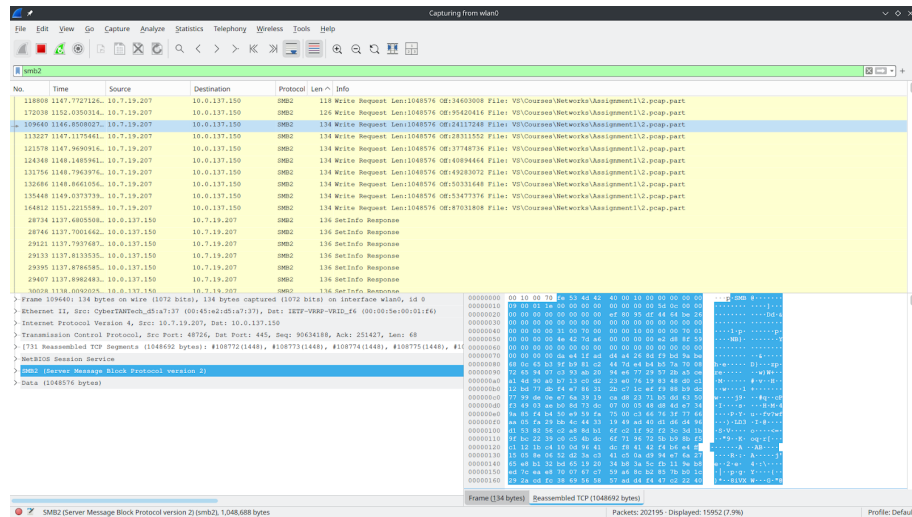


Figure 9: SMB2

3. **CoAP:** The Constrained Application Protocol (CoAP) is a lightweight web transfer protocol designed for constrained devices and low-power networks. It is used in the Internet of Things (IoT) to enable simple communication between devices over UDP. **Associated Document:** *RFC 7252.*

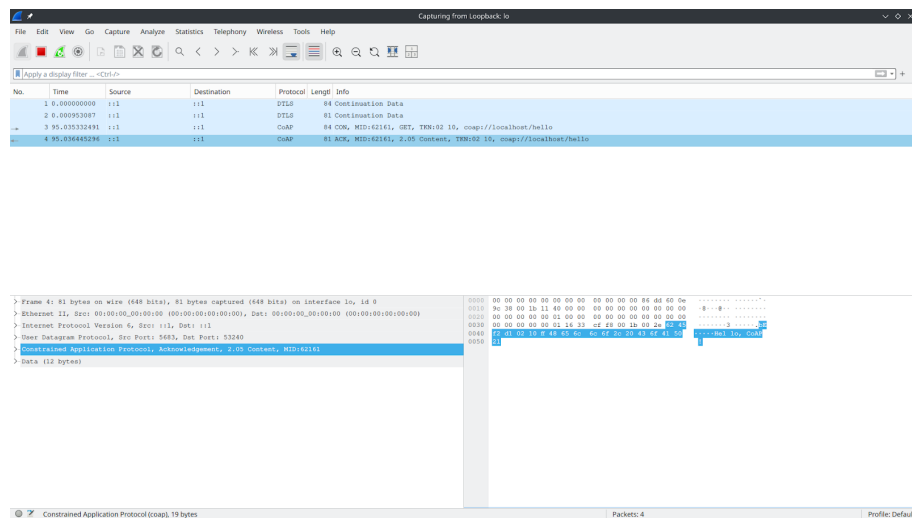


Figure 10: CoAP

4. **Telnet:** Telnet is a network protocol used to provide a command-line interface for communication with remote devices over a TCP/IP network. While historically used for remote server management, it is now considered insecure due to lack of encryption and has been largely replaced by SSH. **Associated Document:** *RFC 854*.

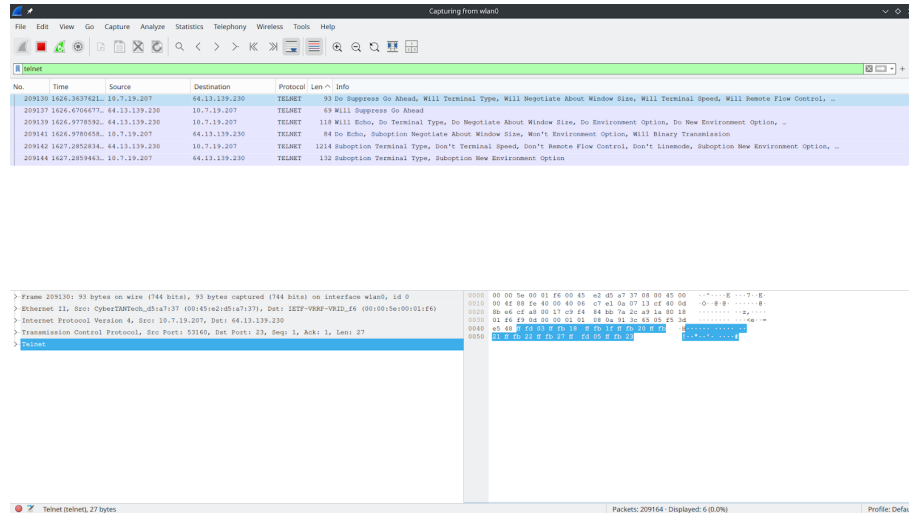


Figure 11: TELNET

5. **Z39.50:** Z39.50 is an information retrieval protocol designed for searching and retrieving bibliographic and related information from remote databases. It is widely used in libraries and academic institutions for searching catalog records. **Associated Document:** *RFC 1729*.

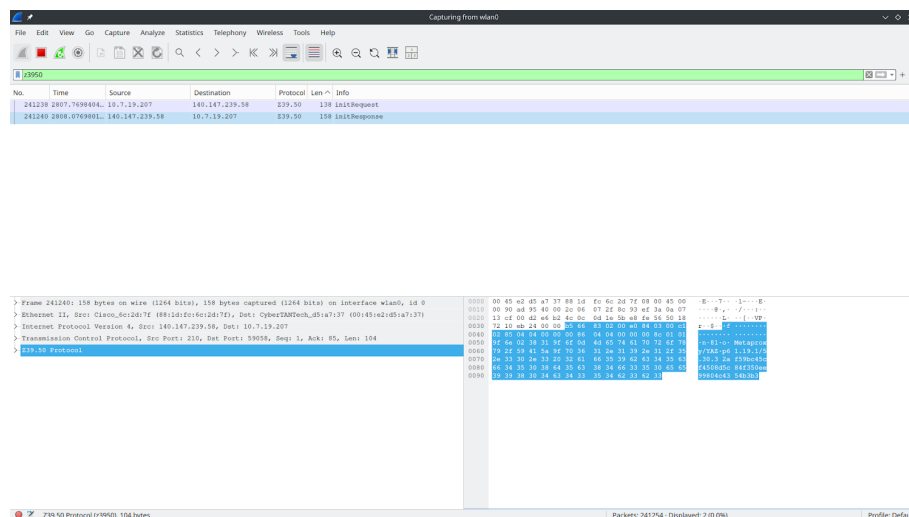


Figure 12: Z39.50

6.2 Website Packet Analysis

2(a). Identify the request line with the version of the application layer protocol and the IP address for the websites *canarabank.com*, *github.com*, and *netflix.com*. Also, identify whether the connection(s) is/are persistent or not.

Answer: The following information was gathered using the `curl -v` command to observe the request headers and connection details for each website listed.

1. **canarabank.com:**

- Request Line: GET / HTTP/1.1
- Application layer protocol version: HTTP/1.1
- IP Address: 107.162.160.8
- Persistent Connections: No

```
shubham0409@ShubhamAg-PC:~$ curl -v canarabank.com
* Host canarabank.com:80 was resolved.
* IPv6: 2401:8800:a50:4::3
* IPv4: 107.162.160.8
* Trying 107.162.160.8:80...
* Connected to canarabank.com (107.162.160.8) port 80
> GET / HTTP/1.1
> Host: canarabank.com
> User-Agent: curl/8.5.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 302 Moved Temporarily
< Location: https://canarabank.com/
< Via: HTTP/1.1 bit29005.sin1.defense.net
< Connection: close
< Content-Length: 0
<
* Closing connection
```

Figure 13: Canarabank

2. github.com:

- Request Line: GET / HTTP/1.1
- Application layer protocol version: HTTP/1.1
- IP Address: 20.205.243.166
- Persistent Connections: Yes

```
shubham0409@ShubhamAg-PC:~$ curl -v github.com
* Host github.com:80 was resolved.
* IPv6: (none)
* IPv4: 20.205.243.166
* Trying 20.205.243.166:80...
* Connected to github.com (20.205.243.166) port 80
> GET / HTTP/1.1
> Host: github.com
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Content-Length: 0
< Location: https://github.com/
<
* Connection #0 to host github.com left intact
```

Figure 14: Github

3. netflix.com:

- Request Line: GET / HTTP/1.1
- Application layer protocol version: HTTP/1.1
- IP Address: 44.234.232.238, 44.237.234.25, 44.242.60.85
- Persistent Connections: Yes

```
shubham0409@ShubhamAg-PC:~$ curl -v netflix.com
* Host netflix.com:80 was resolved.
* IPv6: 2600:1f14:62a:de81:b848:82ee:2416:447e, 2600:1f14:62a:de80:69a8:7b12:8e5f:855d, 2600:1f14:62a:de82:822d:a423:9e4c:da8d
* IPv4: 44.234.232.238, 44.237.234.25, 44.242.60.85
* Trying 44.234.232.238:80...
* Trying [2600:1f14:62a:de81:b848:82ee:2416:447e]:80...
* Immediate connect fail for 2600:1f14:62a:de81:b848:82ee:2416:447e: Network is unreachable
* Trying [2600:1f14:62a:de80:69a8:7b12:8e5f:855d]:80...
* Immediate connect fail for 2600:1f14:62a:de80:69a8:7b12:8e5f:855d: Network is unreachable
* Trying [2600:1f14:62a:de82:822d:a423:9e4c:da8d]:80...
* Immediate connect fail for 2600:1f14:62a:de82:822d:a423:9e4c:da8d: Network is unreachable
* Connected to netflix.com (44.234.232.238) port 80
> GET / HTTP/1.1
> Host: netflix.com
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: https://netflix.com/
< Content-Length: 0
< Via: 1.1 1-8eddc854ee7e59e2 (us-west-2)
< X-Xss-Protection: 1; mode=block; report=https://www.netflix.com/ichnaea/Log/freeform/xssreport
< X-Content-Type-Options: nosniff
< Strict-Transport-Security: max-age=31536000; includeSubDomains
< X-Originating-URL: http://netflix.com/
< Set-Cookie: nfeid=80F8AE8EBD11kg9lpa7CbcyfiFY5BhZaekDwd0YXdlkhQTOUfqbEylISNKY5rF10b3zyIE960GcCRTtd1T45SHvNPFwLR5ss-HIhw03kM19WNT8FwK3Dk3D; Domain=.netflix.com; Path=/; Max-Age=31536000
< X-Netflix-nfstatus: 1.21
< X-Netflix.proxy.execution-time: 2
* Connection #0 to host netflix.com left intact
```

Figure 15: Netflix

*[If connection closed, then not persistent connection. If connection left intact, then persistent connection. Refer the screenshots attached]

2(b). *For any one of the websites, list any three header field names and corresponding values in the request and response message. Also, list any three HTTP error codes obtained while loading one of the pages with a brief description.*

1. **Website:** <https://github.com>

2. **Request Headers:**

- **Origin:** `https://github.com/` Indicates the URL of the origin page.
- **accept-language:** `en-GB,en-US;q=0.9,en;q=0.8` Indicates the preferred languages for the response
- **Sec-CH-UA-Platform:** `"Windows"` Shows that the platform is Windows.

3. **Response Headers:**

- **Accept-Ranges:** `bytes` Indicates that the server supports range requests, allowing clients to request specific byte ranges of a resource.
- **Content-Type:** `text/css` Specifies that the content of the response is CSS (Cascading Style Sheets) code.
- **Content-Length:** `21799` Specifies the size of the response body in bytes, in this case, 21,799 bytes.

4. **HTTP Error Codes:**

- **404 Not Found:** Indicates that the requested resource could not be found on the server.
- **403 Forbidden:** The server understands the request but refuses to authorize it.
- **429 Too Many Requests:** The user has sent too many requests in a given amount of time.

2(c). *Capture the Performance metrics that your browser records when a page is loaded and also report the list of 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.*

1. **Webpage:** <https://github.com>

2. **Browser Name:** Google Chrome

3. **Performance Metrics:**

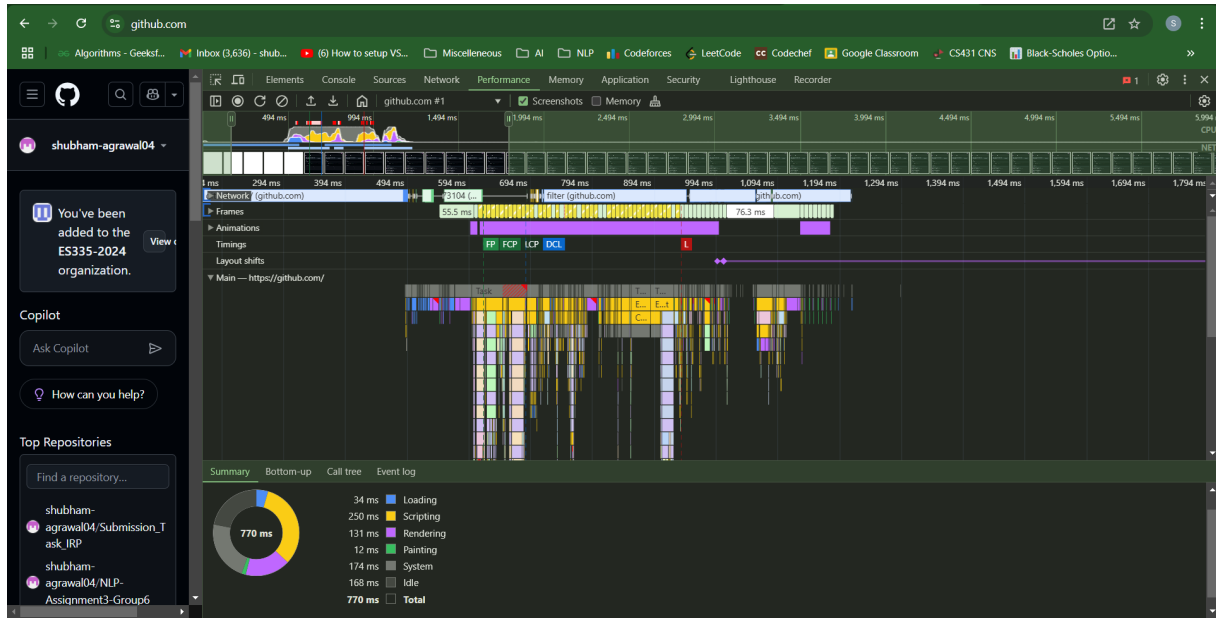


Figure 16: Performance Metrics

4. **Cookies and Flags:**

Key Cookies:

- `_gh_sess` (Session Cookie)
- `logged_in=yes`
- `preferred_color_mode=dark`
- `cpu_bucket=xlg`
- `tz=Asia/Calcutta`
- `dotcom_user=shubham-agrawal04`

Flags: HTTPOnly and Secure are applied to the session cookie `_gh_sess`

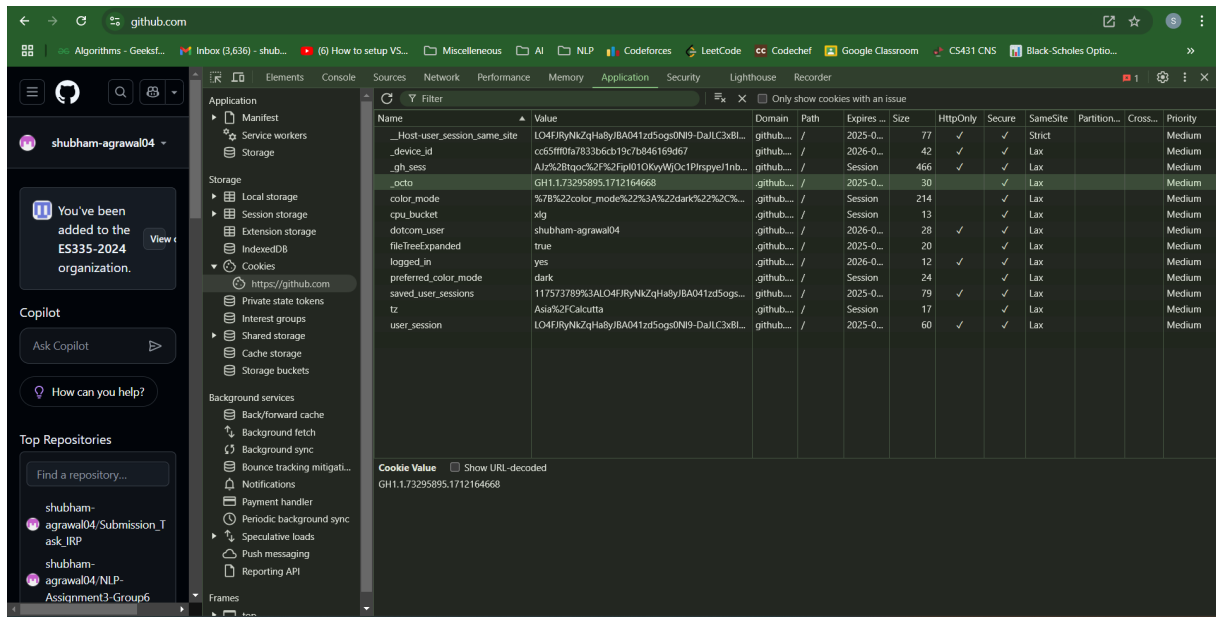


Figure 17: Cookies

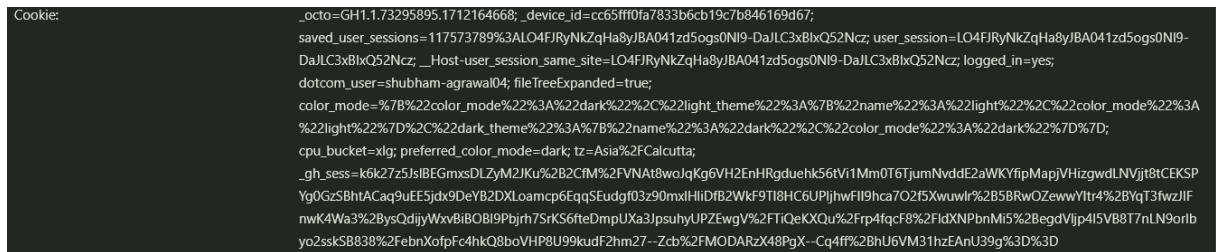


Figure 18: Request-header Cookies

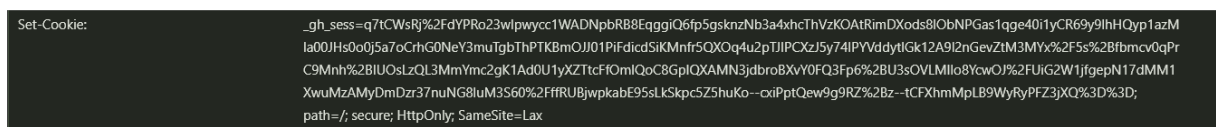


Figure 19: Response-header Cookies

7 Files and Code Location

All files and code related to this assignment can be found in the following GitHub repository:

https://github.com/shubham-agrawal04/CN_Assignment-1

8 Replicating the Results

To replicate the results of this assignment for Part-1 and Part-2, follow these steps:

8.1 Clone the repository

Clone the repository using the following command:

```
git clone https://github.com/shubham-agrawal04/CN_Assignment-1.git
cd repo-directory
```

8.2 Install dependencies

Ensure that you have Python installed on your system as well as the necessary Python libraries.

8.3 Install PCAP File

Ensure that you have 2.pcap file installed on your device. Move the file into the repo-directory.

Download here: https://drive.google.com/drive/folders/1n84jGddZ38fDjy9jKH3qw3J_H0SaKThu

8.4 Set up your environment

- Configure the Ethernet connection between two machines correctly.
- It is recommended to use the `eth0` port of your WSL instance in Windows for both `tcpreplay` and the packet sniffing script.
- Ensure that your Wi-Fi internet connection is turned off to avoid interference with the Ethernet-based communication.

8.5 Run the packet sniffing script

On the machine where you are sniffing packets, execute the following command:

```
sudo python3 temp2.py -i [interface/port-name]
```

8.6 Run tcpreplay

On the machine where tcpreplay is set up, use the following command to replay the packets:

```
sudo tcpreplay -i [interface/port-name] --pps=[pps-speed] 2.pcap
```

(Any pps less than equal to 14000 will ideally give correct results for same machine run)

(May need to try for multiple pps for getting optimal results for different machine run)

8.7 Check results

Wait for the script to complete executing and voila! You have the results.