INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR

Assignment Report
Computer Networks

Submitted By

Bhavik Patel - 22110047
Jinil Patel - 22110184

Github Repository Link

Part 1: Metrics and Plots

Objectives

The main objectives of this assignment were:

1. Capture and analyze network traffic using `packetSniffer.py`.
2. Extract key metrics from captured traffic, including:
    - Total data transferred (bytes)
    - Total packets transferred
    - Min, Max, and Avg packet size
    - Packet size distribution
    - Unique `<SourceIP:SrcPort , DestinationIP:DestPort>` pairs
    - Dictionary of `{SourceIP: Total Flows}` and `{DestinationIP: Total Flows}`
    - `<SourceIP:Port , DestinationIP:Port>` pair transferring the most data
3. Determine the maximum speed (pps/mbps) at which the sniffer can capture data loss-free in two scenarios:
    - Running `tcpreplay` and `packetSniffer.py` on the same machine.
    - Running `tcpreplay` on one machine and `packetSniffer.py` on another.
4. Verify results by implementing `pcapAnalyzer.py` to read `.pcap` files and extract the same metrics found by `packetSniffer.py`.

Implementation Details

Packet Sniffer (`packetSniffer.py`)

The `packetSniffer.py` program captures packets in real-time, records their attributes, and computes the required metrics. It was designed to handle high-speed packet capture efficiently while minimizing data loss.

PCAP Analyzer (`pcapAnalyzer.py`)

Since `packetSniffer.py` captures packets dynamically, we implemented `pcapAnalyzer.py` to analyze pre-captured `.pcap` files and extract the same metrics for validation. This helped us confirm the correctness of our sniffer's results.

Packet Replay using `tcpreplay`

To test the performance and accuracy of our sniffer, we replayed packets from a chosen `5.pcap` file using `tcpreplay`. This allowed us to simulate real-world traffic and compare captured results with ground truth data from the `.pcap` file.

Packet Capture Speed `speedTest.py`

To evaluate the efficiency of our packet sniffer, we implemented `speedTest.py`, which measures the rate at which packets are captured during `tcpreplay` execution. By systematically adjusting replay speeds through a trial-and-error approach, we determine the maximum rate at which our sniffer can capture packets without loss. This analysis helps optimize performance and ensures reliable data collection under high-speed traffic conditions.

Running the Programs

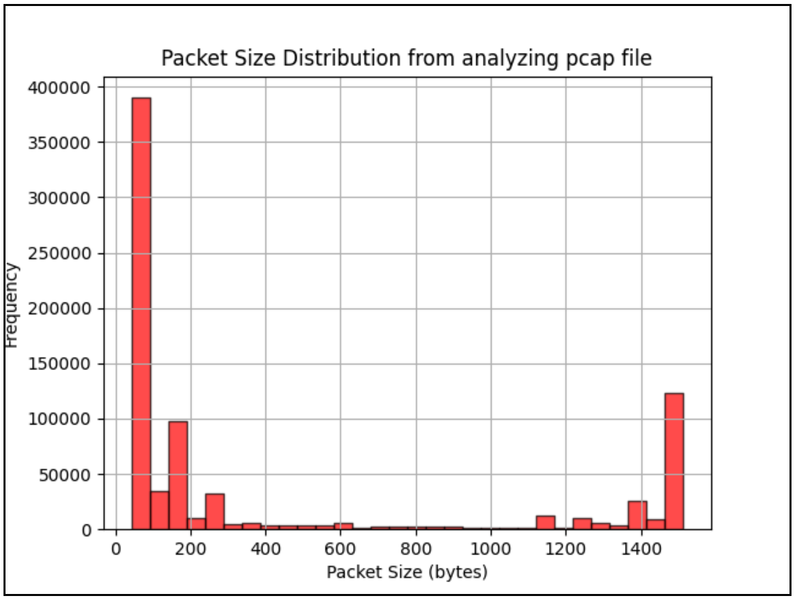
Order of program execution:

1. pcapAnalyzer.py: Analyzes the **5.pcap** file to extract and validate key metrics.
2. speedTest.py: Determines the optimal **tcpreplay** speed for packet capture without loss, using a trial-and-error approach.
3. packetSniffer.py: Once the optimal speed is identified, captures packets in real time and computes metrics with zero packet loss.

Running `pcapAnalyzer.py`

Command: **python .\pcapAnalyzer.py**

```
PS C:\Users\DELL\Downloads\Computer_Networks\CS331-Computer-Networks-Assignment1> python .\pcapAnalyser.py
Reading pcap file...
Read 805996 packets in 381.08 seconds.
Analyzing results...

** Packet Sniffer Metrics **
Total Data Transferred: 364640811 bytes
Total Packets Transferred: 805996
Minimum Packet Size: 42 bytes
Maximum Packet Size: 1514 bytes
Average Packet Size: 452.41 bytes

Total unique Source-Destination Pairs: 41860
172.16.133.95:49358 -> 157.56.240.102:443: 17342229 bytes
172.16.133.57:53807 -> 68.64.21.62:1853: 17318420 bytes
172.16.133.36:64953 -> 67.217.64.99:443: 15842153 bytes
67.217.64.99:443 -> 172.16.133.26:53037: 14997326 bytes
172.16.128.201:1060 -> 172.16.133.6:1731: 5264163 bytes

Top 5 source IPs by flow count:
172.16.133.57: 30563 flows
67.217.64.99: 17993 flows
172.16.133.36: 17249 flows
172.16.133.95: 17225 flows
172.16.133.116: 16228 flows

Top 5 destination IPs by flow count:
172.16.139.250: 181258 flows
68.64.21.62: 25733 flows
172.16.133.57: 19723 flows
172.16.133.26: 17767 flows
67.217.64.99: 16875 flows
Results saved in pcapAnalyzer_logs folder.
Analyzed results in 79.48 seconds.
Pcap file analysis completed.
```



Total Packets Captured	805996
Total Data Transferred	364640811 bytes
Minimum Packet Size	42 byte
Maximum Packet Size	1514 byte
Average Packet Size	452.41 byte
Highest SrcIP-DstIP pair in terms of data	172.16.133.95:49358 → 157.56.240.102:432 (Data = 17342229 bytes)
Total Unique Scr - Dst IP	41860

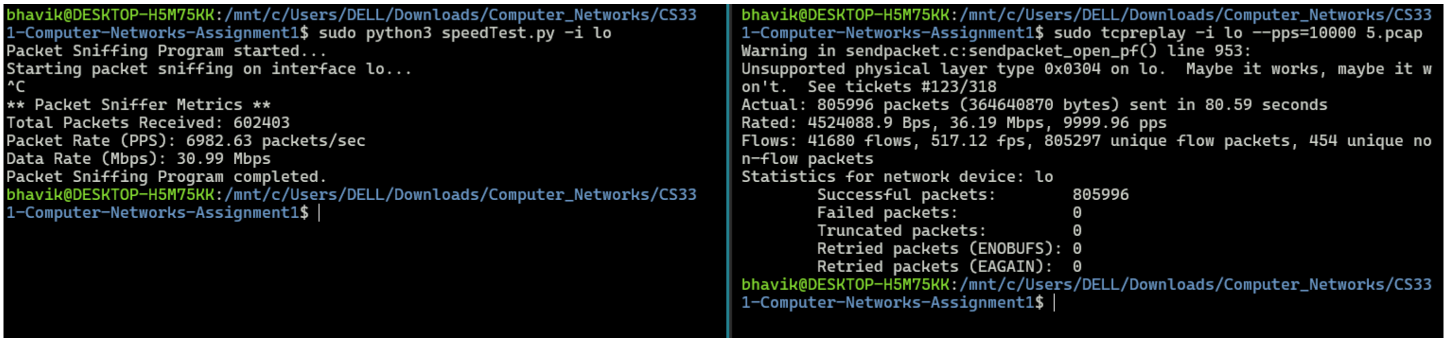
After executing the pcapAnalyzer.py file all the output logs will be stored in a folder called **pcapAnalyzer_logs.** The dictionary of sourceIP:Port to total flow and destinationIP:Port to total flow are stored as a json file inside the logs folder.

Running `speedTest.py and tcpreplay`

1. Open two terminals side by side.
2. In one terminal, run tcpreplay; in another, run the speedTest.py file.

Command for speedTest: sudo python3 packetSniffer.py -i <interface> -[t <timeout>]
Command for tcpreplay: sudo tcpreplay -i <interface> --pps=<speed> <pcapFilePath>

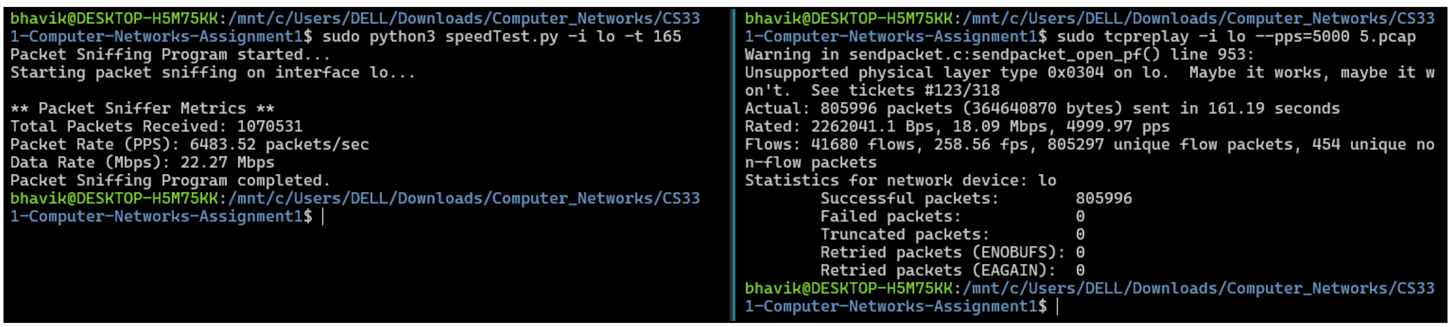
First, we ran tcpreplay with speed pps(packet per second)=10000 but our program captured packets with speed pps=6982.63



```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo python3 speedTest.py -i lo
Packet Sniffing Program started...
Starting packet sniffing on interface lo...
^C
** Packet Sniffer Metrics **
Total Packets Received: 602403
Packet Rate (PPS): 6982.63 packets/sec
Data Rate (Mbps): 30.99 Mbps
Packet Sniffing Program completed.
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo tcpreplay -i lo --pps=10000 5.pcap
Warning in sendpacket.c:sendpacket_open_pf() line 953:
Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe it w
on't.  See tickets #123/318
Actual: 805996 packets (364640870 bytes) sent in 80.59 seconds
Rated: 4524088.9 Bps, 36.19 Mbps, 9999.96 pps
Flows: 41680 flows, 517.12 fps, 805297 unique flow packets, 454 unique no
n-flow packets
Statistics for network device: lo
        Successful packets:        805996
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

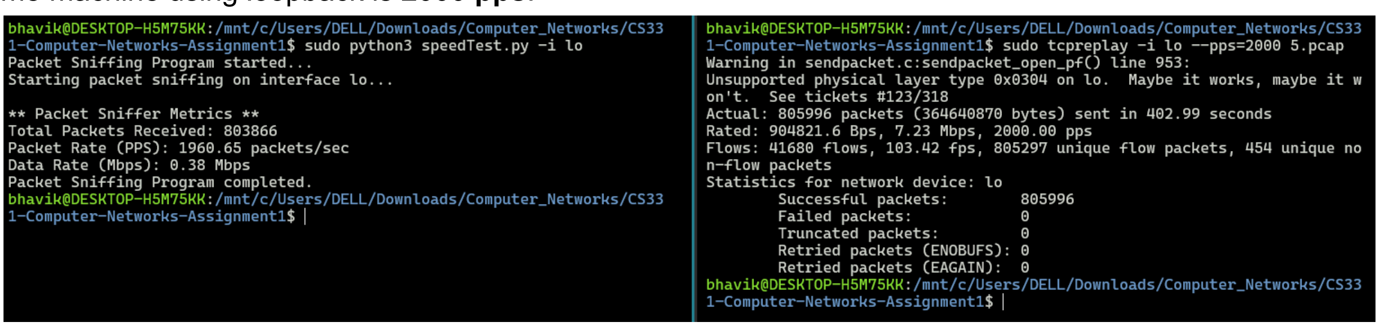
Output with tcpreplay speed = 5000pps.



```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo python3 speedTest.py -i lo -t 165
Packet Sniffing Program started...
Starting packet sniffing on interface lo...

** Packet Sniffer Metrics **
Total Packets Received: 1070531
Packet Rate (PPS): 6483.52 packets/sec
Data Rate (Mbps): 22.27 Mbps
Packet Sniffing Program completed.
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo tcpreplay -i lo --pps=5000 5.pcap
Warning in sendpacket.c:sendpacket_open_pf() line 953:
Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe it w
on't.  See tickets #123/318
Actual: 805996 packets (364640870 bytes) sent in 161.19 seconds
Rated: 2262041.1 Bps, 18.09 Mbps, 4999.97 pps
Flows: 41680 flows, 258.56 fps, 805297 unique flow packets, 454 unique no
n-flow packets
Statistics for network device: lo
        Successful packets:        805996
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

At tcpreplay speed = 2000 pps, we are able to capture almost all the packets, and if the speed is further decreased, the number of packets captured decreases. So the optimal speed for capturing packets within the same machine using loopback is **2000 pps**.



```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo python3 speedTest.py -i lo
Packet Sniffing Program started...
Starting packet sniffing on interface lo...

** Packet Sniffer Metrics **
Total Packets Received: 803866
Packet Rate (PPS): 1960.65 packets/sec
Data Rate (Mbps): 0.38 Mbps
Packet Sniffing Program completed.
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

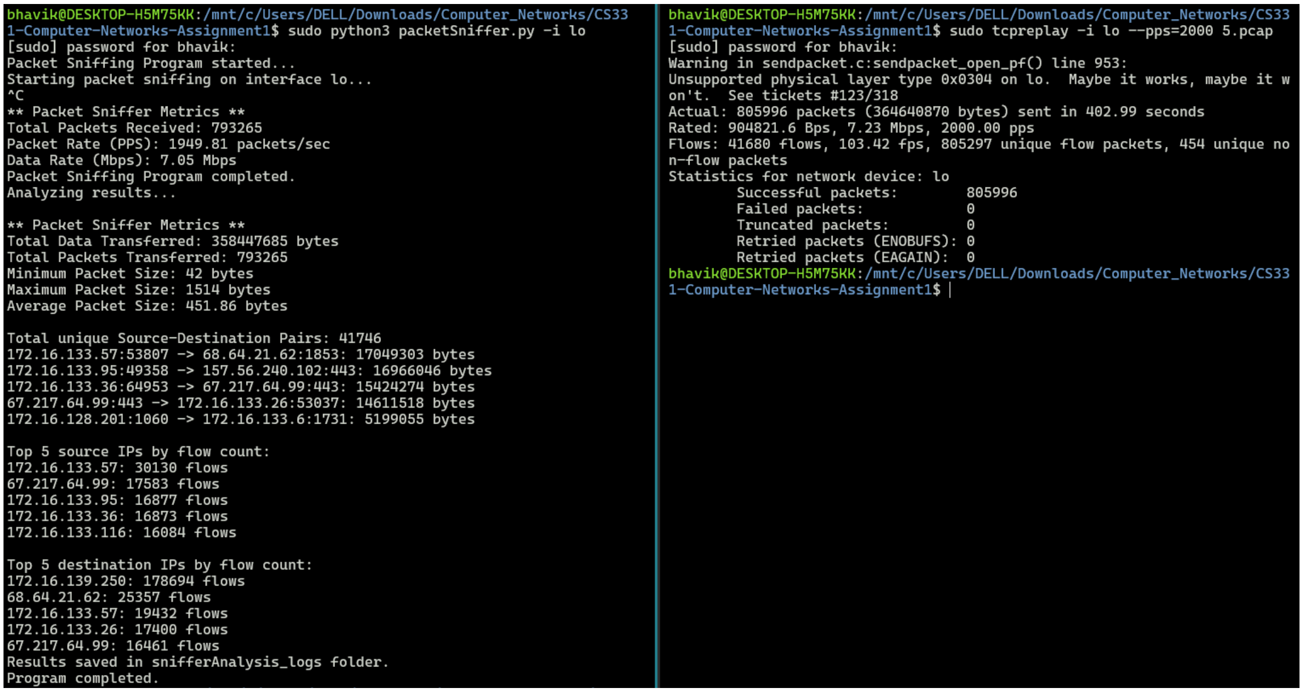
```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo tcpreplay -i lo --pps=2000 5.pcap
Warning in sendpacket.c:sendpacket_open_pf() line 953:
Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe it w
on't.  See tickets #123/318
Actual: 805996 packets (364640870 bytes) sent in 402.99 seconds
Rated: 904821.6 Bps, 7.23 Mbps, 2000.00 pps
Flows: 41680 flows, 103.42 fps, 805297 unique flow packets, 454 unique no
n-flow packets
Statistics for network device: lo
        Successful packets:        805996
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```

Now we calculate metrics by sniffing packets at 2000 pps within the same machine.

Running `packetSniffer.py`

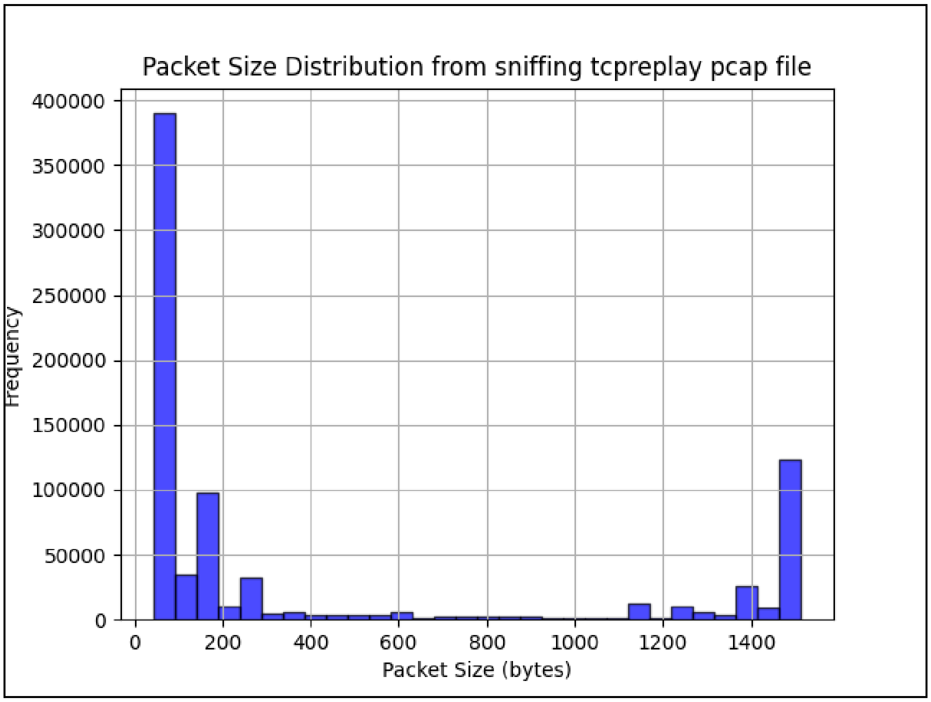
Command: **sudo python3 packetSniffer.py -i <interface>**

Where `<interface>` is the network interface on which packets are captured.



```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo python3 packetSniffer.py -i lo
[sudo] password for bhavik:
Packet Sniffing Program started...
Starting packet sniffing on interface lo...
^C
** Packet Sniffer Metrics **
Total Packets Received: 793265
Packet Rate (PPS): 1949.81 packets/sec
Data Rate (Mbps): 7.05 Mbps
Packet Sniffing Program completed.
Analyzing results...

** Packet Sniffer Metrics **
Total Data Transferred: 358447685 bytes
Total Packets Transferred: 793265
Minimum Packet Size: 42 bytes
Maximum Packet Size: 1514 bytes
Average Packet Size: 451.86 bytes

Total unique Source-Destination Pairs: 41746
172.16.133.57:53807 -> 68.64.21.62:1853: 17049303 bytes
172.16.133.95:49358 -> 157.56.240.102:443: 16966046 bytes
172.16.133.36:64953 -> 67.217.64.99:443: 15424274 bytes
67.217.64.99:443 -> 172.16.133.26:53037: 14611518 bytes
172.16.128.201:1060 -> 172.16.133.6:1731: 5199055 bytes

Top 5 source IPs by flow count:
172.16.133.57: 30130 flows
67.217.64.99: 17583 flows
172.16.133.95: 16877 flows
172.16.133.36: 16873 flows
172.16.133.116: 16084 flows

Top 5 destination IPs by flow count:
172.16.139.250: 178694 flows
68.64.21.62: 25357 flows
172.16.133.57: 19432 flows
172.16.133.26: 17400 flows
67.217.64.99: 16461 flows
Results saved in snifferAnalysis_logs folder.
Program completed.
```

```
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ sudo tcpreplay -i lo --pps=2000 5.pcap
[sudo] password for bhavik:
Warning in sendpacket.c:sendpacket_open_pf() line 953:
Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe it w
on't.  See tickets #123/318
Actual: 805996 packets (364640870 bytes) sent in 402.99 seconds
Rated: 904821.6 Bps, 7.23 Mbps, 2000.00 pps
Flows: 41680 flows, 103.42 fps, 805297 unique flow packets, 454 unique no
n-flow packets
Statistics for network device: lo
        Successful packets:        805996
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
bhavik@DESKTOP-H5M75KK:/mnt/c/Users/DELL/Downloads/Computer_Networks/CS33
1-Computer-Networks-Assignment1$ |
```



Metric Calculated sniffing tcpreplay

Total Packets Captured	803866
Total Data Transferred	358447685 bytes
Minimum Packet Size	42 byte
Maximum Packet Size	1514 byte
Average Packet Size	451.86 byte
Highest SrcIP-DstIP pair in terms of data	172.16.133.95:49358 → 157.56.240.102:432 (Data = 17049303 bytes)
Total Unique Scr - Dst IP	41746

The dictionary of sourceIP:Port to total flow and destinationIP:Port to total flow are stored as a json file inside the logs folder.

NOTE: After running pcapAnalyzer.py or packetSniffer.py, a log folder will be created which stores all the metrics that we got through the program.

Packet transfer from one machine to other machine using Ethernet Cable:

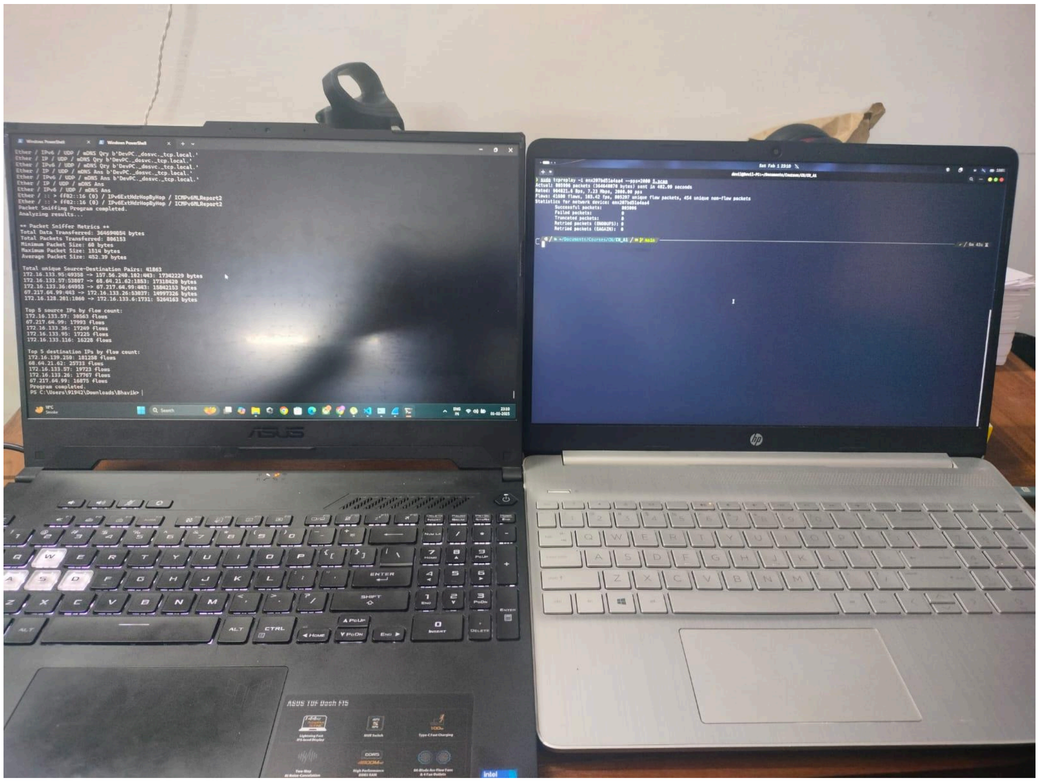
To ensure accurate packet capture, we used an **Ethernet cable** to create a direct communication channel between two machines:

Disabled Wi-Fi on both machines to avoid interference.

1. **Connected both machines via an Ethernet cable**, forming a dedicated data transfer channel.
2. **Ran `tcpreplay` on one machine** to send packets at a controlled rate.
3. **Executed `packetSniffer.py` on the other machine** to capture and analyze packets.

Video showing the packet transfer via ethernet cable→ Link

The speed at which we were able to send the data was **2000 pps** with minimal packet loss.



Part 2: Catch Me If You Can

For this part add we need to run packetSnifferPart2.py python script to get the below results
Command: **python packetSnifferPart2.py**

Q1: Finding TCP Packet Containing a Filename

- **Filename Extraction:**
    - Searched for <The name of file is = > in packet payloads.
    - Extracted filename: *networking_Questions.pdf*
- **TCP Checksum of the Packet:** *35409*
- **Source IP Address:** *10.20.30.200*

Q2: Counting Packets from the Source IP=10.20.30.200

- **Total packets from source IP:** *30*

Q3: Finding the Phone's Company Name from Localhost Request

- **Company Name Extraction:**
    - Searched for <Company of phone is = > in packet payloads.
    - Extracted company name: *Samsung*
- **Port Used by Localhost:** *1001*
- **Total packets from localhost:** *30*

Program Output: packetSnifferPart2.py

```
(base) zeenu@JINIL /mnt/d/Sem-6/Computer Networks/CS331-Computer-Networks-Assignment1  main ±  python packetSnifferPart2.py 5.pcap
Packet Sniffing Program started...
Packet Sniffing Completed...

** TCP Packet Containing File Name **
String Containing File Name: 'The name of file is = networking_Questions.pdf'
File Name: networking_Questions.pdf
TCP Checksum: 35409
Source IP Address: 10.20.30.200
Total packets from 10.20.30.200: 30

** Localhost Phone Company Request Found **
Port Used by Localhost: 1001
String Containing Company Name: 'Company of phone is = Samsung'
Company Name: Samsung
Total packets from localhost: 30
```

Part 3: Capture the packets

Q1)

QUIC (Quick UDP Internet Connections)

- **Operation/Usage**: QUIC is a transport layer protocol designed to improve performance by reducing latency and supporting multiplexed connections over UDP. It is primarily used for HTTP/3.
- **Layer**: Transport Layer
- **RFC**: RFC 9000

ARP (Address Resolution Protocol)

- **Operation/Usage**: ARP is used to map a 32-bit IP address to a MAC address in a local network, enabling communication at the data link layer.
- **Layer**: Data Link Layer
- **RFC**: RFC 826

TLSv1.2 (Transport Layer Security version 1.2)

- **Operation/Usage**: TLSv1.2 provides security for communications over a computer network, ensuring confidentiality, integrity, and authentication at the transport layer.
- **Layer**: Transport Layer
- **RFC**: RFC 5246

DNS (Domain Name System)

- **Operation/Usage**: DNS resolves domain names (e.g., example.com) to IP addresses, enabling human-readable addresses to be translated into machine-readable addresses.
- **Layer**: Application Layer
- **RFC**: RFC 1035

TLSv1.3 (Transport Layer Security version 1.3)

- **Operation/Usage**: TLSv1.3 is the latest version of TLS, providing faster and more secure communication by reducing handshake latency and removing obsolete features.
- **Layer**: Transport Layer
- **RFC**: RFC 8446

IGMPv3 (Internet Group Management Protocol version 3)

- **Operation/Usage**: IGMPv3 manages membership of IP multicast groups, allowing hosts to join or leave multicast groups, primarily used in IP multicast communication.
- **Layer**: Network Layer

- **RFC**: RFC 3376

ICMPv6 (Internet Control Message Protocol version 6)

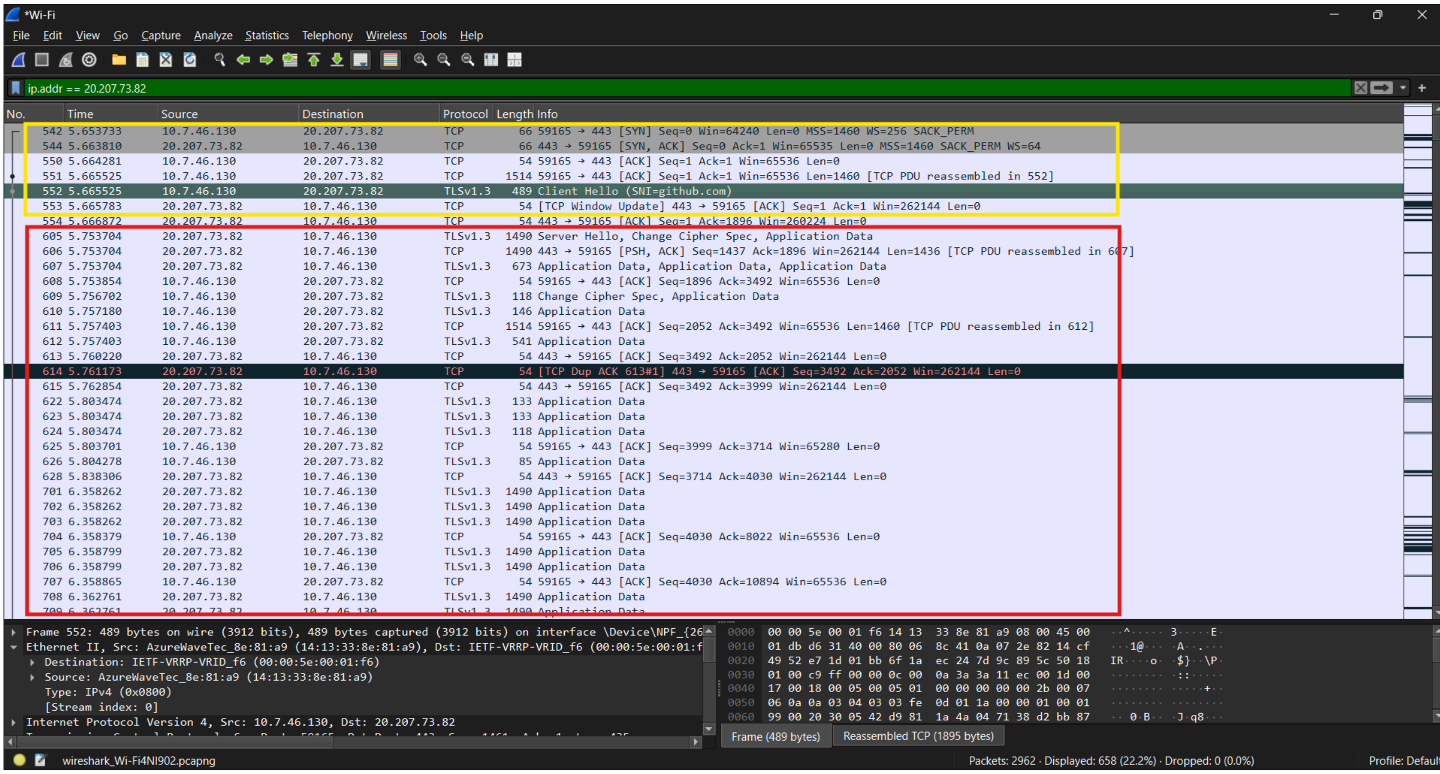
- **Operation/Usage**: ICMPv6 is used for error reporting and diagnostics in IPv6 networks, supporting functionalities like neighbor discovery and path MTU discovery.
- **Layer**: Network Layer
- **RFC**: RFC 4443

Q2) Analyze the following details by visiting the following websites in your favourite browser.

1. canarabank.in
2. github.com
3. netflix.com

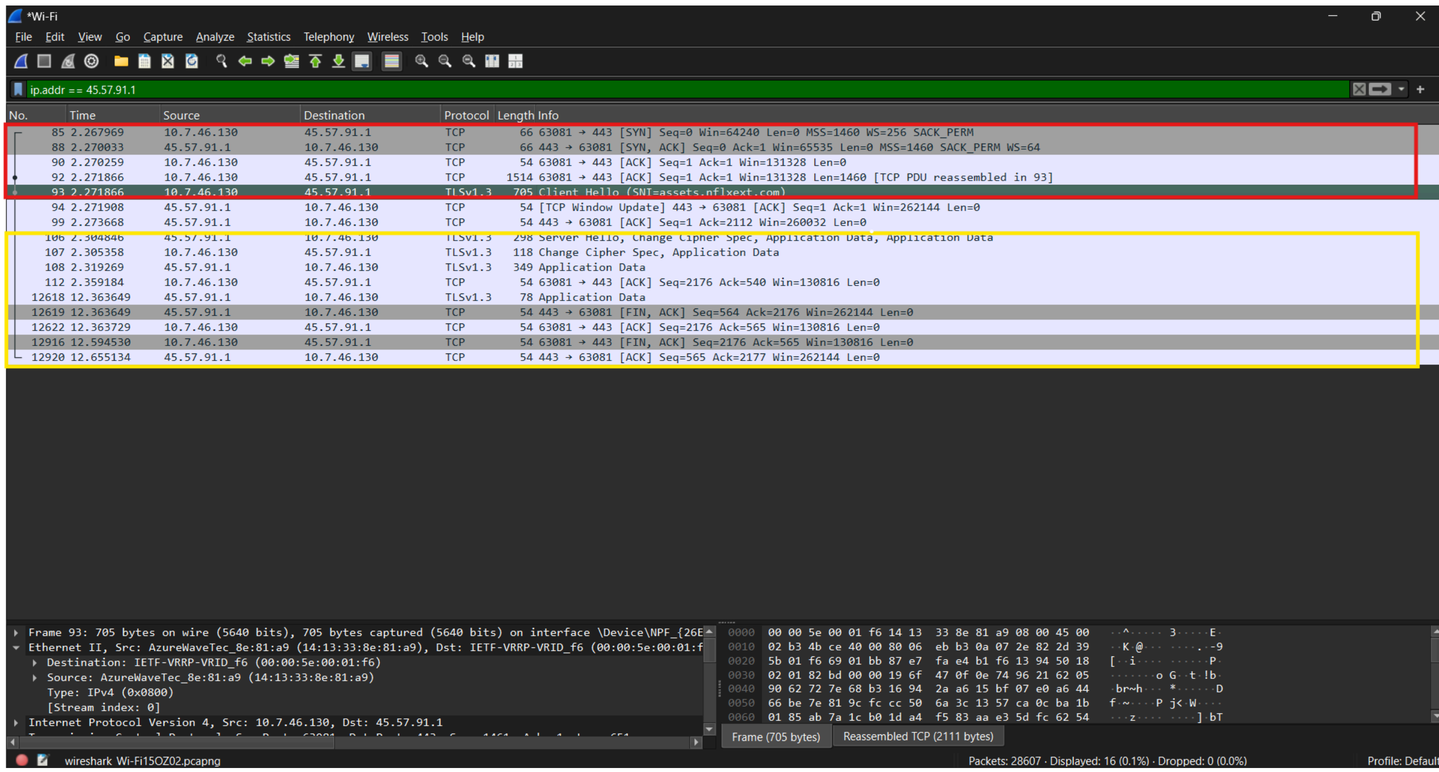
a. Identify `request line` with the version of the application layer protocol and the IP address. Also, identify whether the connection(s) is/are persistent or not.

1. **github.com**



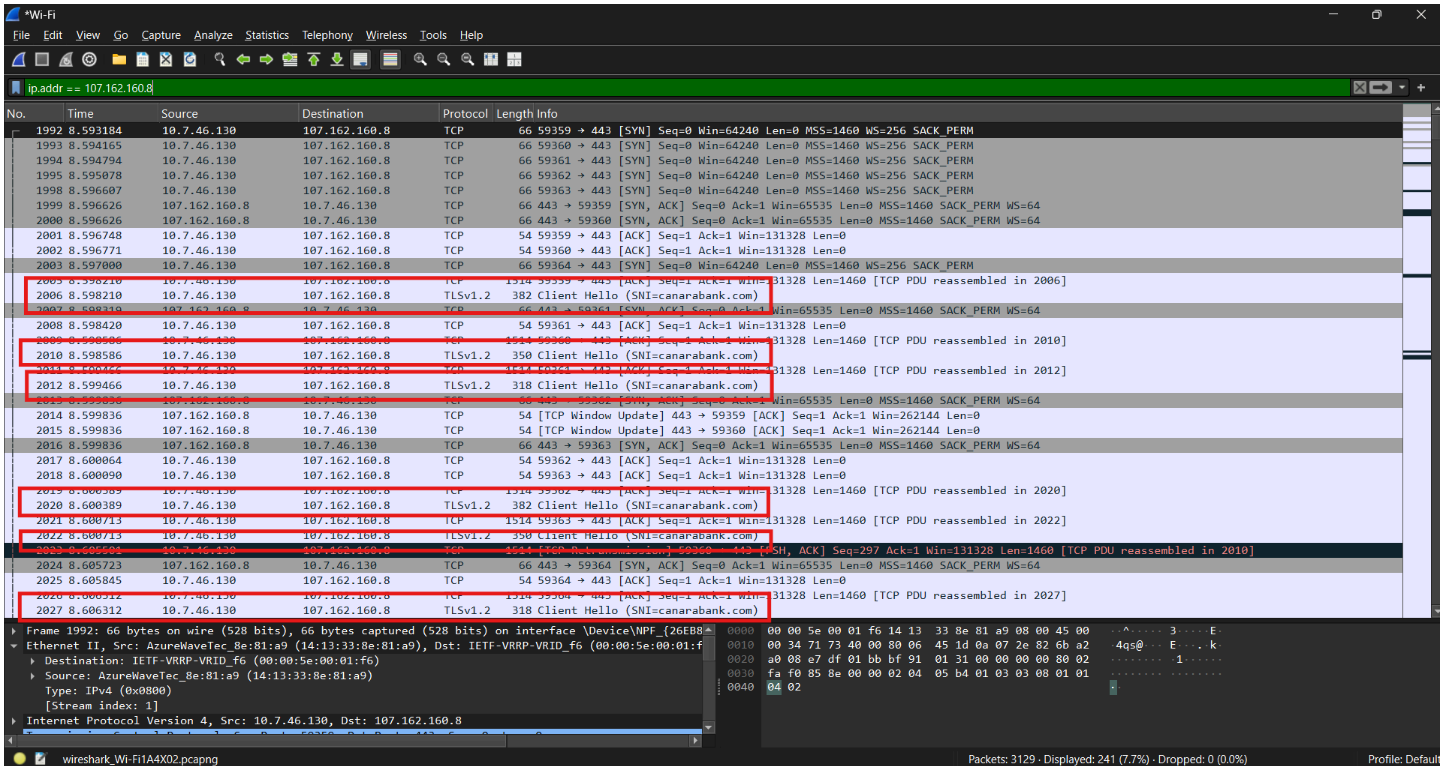
- **Persistent Connection:-** The connection is persistent. This is determined from the observed packet exchanges in the TCP and TLS layers. After the initial connection is established using the SYN, ACK packets,in a round-robin fashion, and the ClientHello and ServerHello messages for TLSv1.3, the connection remains open for further communication. There is no indication of the connection being closed after the initial handshake; instead, data continues to be exchanged using the established connection, confirming that it is persistent.

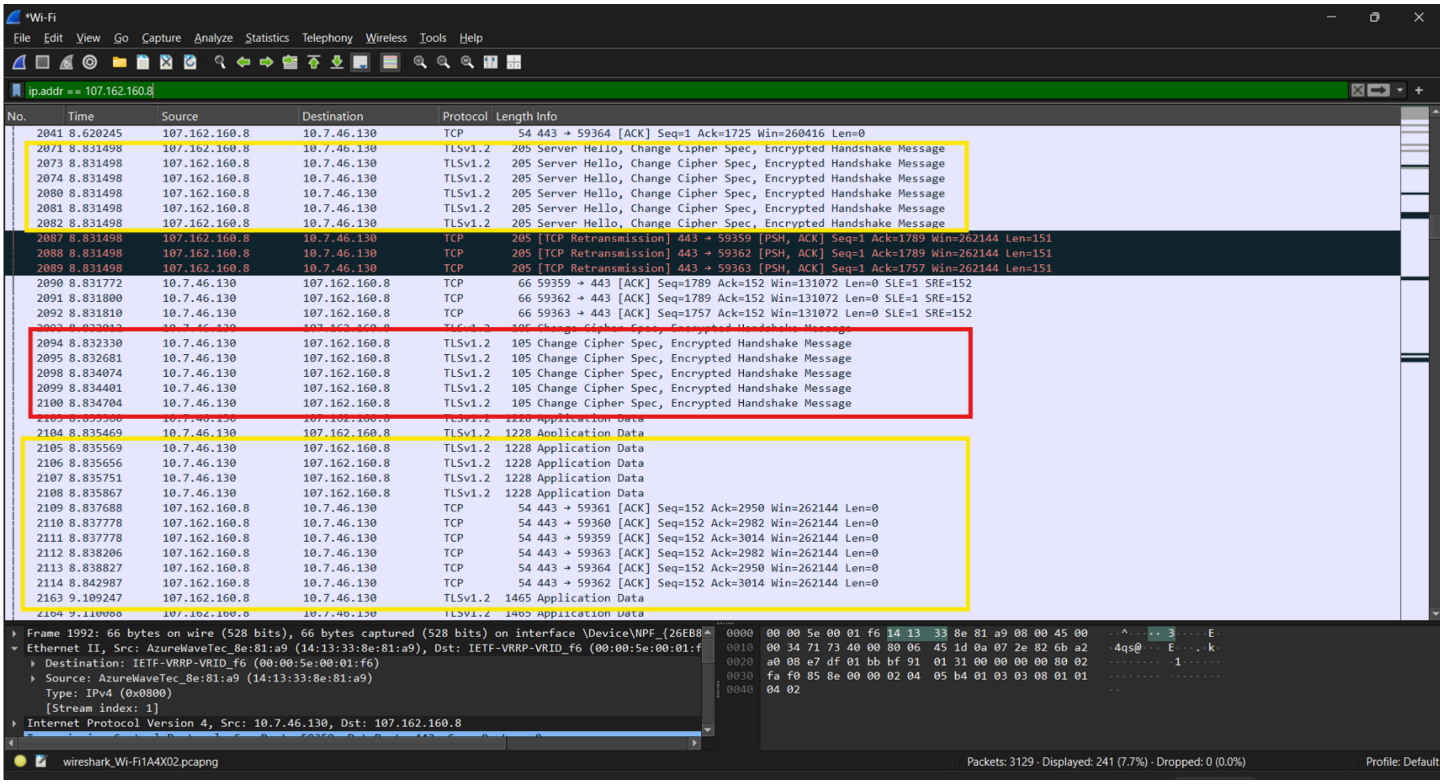
2. **Netflix.com**



- **Persistent Connection:-** The connection is persistent. This is evident from the observed packet exchanges in both the TCP and TLS layers. After the initial connection setup using the SYN, ACK packets in a round-robin fashion, and the ClientHello and ServerHello messages for TLSv1.3, the connection remains open for subsequent communication. There is no indication that the connection is closed after the handshake. Instead, data is continuously exchanged via the established connection, confirming that the connection is persistent throughout the session.

3. Canarabank.com:-





Potentially Non-Persistent at First:- In the typical TLS handshake process, after the initial SYN and ACK packets for TCP connection setup, the ClientHello and ServerHello messages exchange keys to establish the secure connection. However, in this case, Canara Bank's server seems to respond with multiple ServerHello packets, likely indicating some negotiation or retries. Additionally, the presence of several encrypted handshake messages suggests that the initial connection setup is a bit more complicated, possibly due to the system trying to establish a secure TLS connection with multiple attempts.
Becomes Persistent:- While the initial connection might not be persistent due to the retries in the handshake process, once the TLS handshake and encryption setup are successful, the connection likely becomes persistent, with continuous data exchange occurring without further handshakes. This behavior aligns with how many modern web services establish connections that may have some overhead during the handshake but maintain a persistent connection afterward.

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.

For https://github.com/ website:-

Request Headers:-

1. **Accept:-** This header tells the server what kind of media types (content types) the client is willing to accept in the response.
2. **Accept-Encoding:-** This header specifies the types of content encodings (compression algorithms) that the client can understand. It helps the server decide how to encode the response body.
3. **Accept-Language:-** This header indicates the preferred languages for the response. The server can use this to provide the content in the language preferred by the client (if available).
4. **User-Agent:-** The user-agent header in an HTTP request is used to identify the client software making the request to the server. It typically contains information about the browser or application, the operating system, and sometimes additional details like the version number of the software.

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
User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36

Response Header:-

1. **Content-Type:-** Indicates the media type (MIME type) of the response body. It tells the client what kind of data is being sent, such as HTML, JSON, or an image.
2. **Content-Encoding:-** Specifies the encoding (compression) applied to the response body. This allows the server to compress the content to save bandwidth, and the client knows how to decompress it.
3. **Referrer-Policy:-** Controls how much information is sent in the referrer header when making requests from the current site. It defines the level of privacy for the referrer information.
4. **ETag:-** Provides a unique identifier (usually a hash) for the resource version. It helps with caching and conditional requests. When a client makes a subsequent request, it can send the ETag value to check if the resource has changed (via the If-None-Match header).

Content-Type:	text/html; charset=utf-8
Date:	Sat, 01 Feb 2025 13:59:47 GMT
Etag:	W/"c3eb17c8779adfcf100ee320663d8f95"
Referrer-Policy:	origin-when-cross-origin, strict-origin-when-cross-origin
Cache-Control:	max-age=0, private, must-revalidate
Content-Encoding:	gzip

Error code 401:-

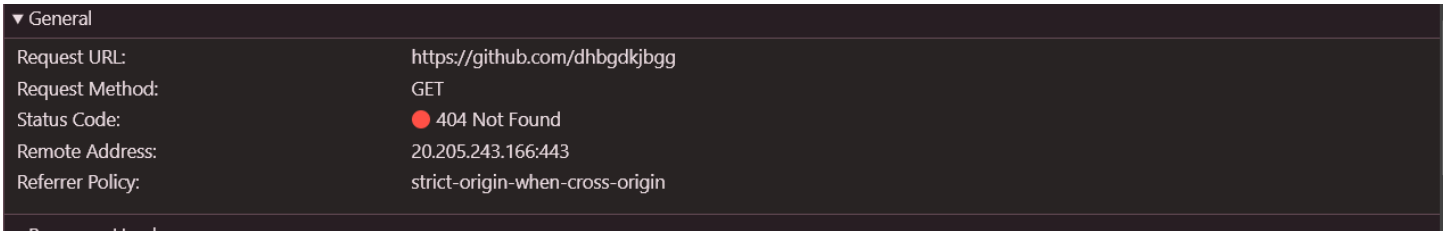
- Unauthorized indicates that the request made to the server requires authentication, but the client has either not provided valid credentials or failed to authenticate properly. It typically occurs when the server expects login details (like a username and password or an API key), but they are missing, incorrect, or insufficient to gain access to the requested resource.

▼ General

Request URL:	https://api.github.com/user/repos
Request Method:	GET
Status Code:	401 Unauthorized
Remote Address:	20.207.73.85:443
Referrer Policy:	strict-origin-when-cross-origin

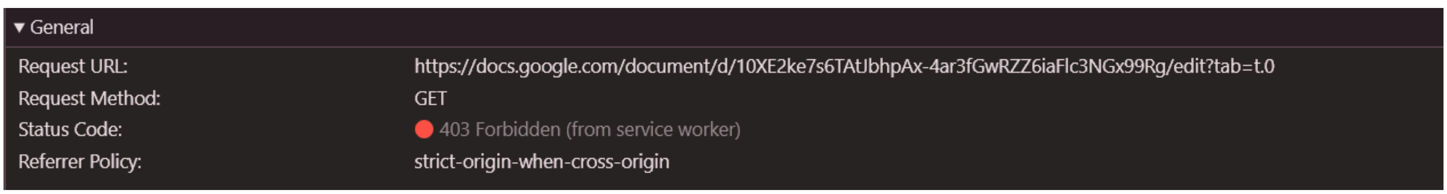
Error code 404:-

- Not Found occurs when the server cannot find the requested resource. This could be due to a wrong URL, a deleted or moved webpage, or an incorrect path to the resource. It indicates that the server is reachable, but it doesn't have the specific resource the client is asking for.



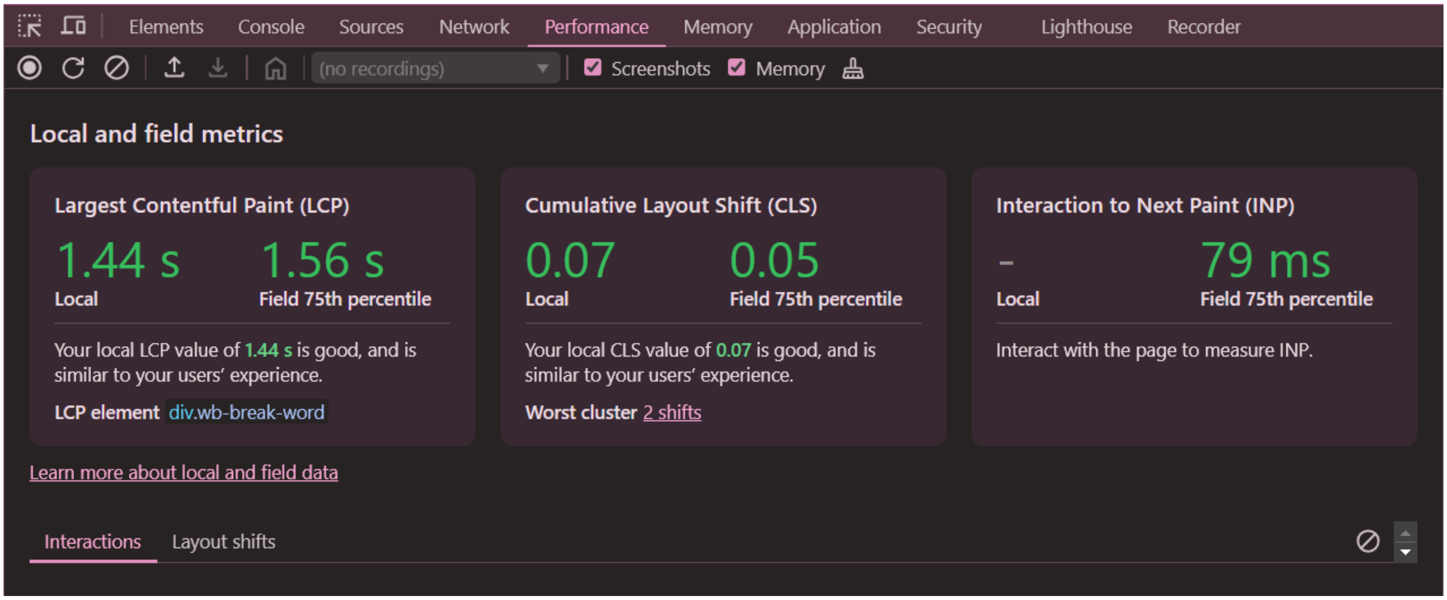
Error code 403:-

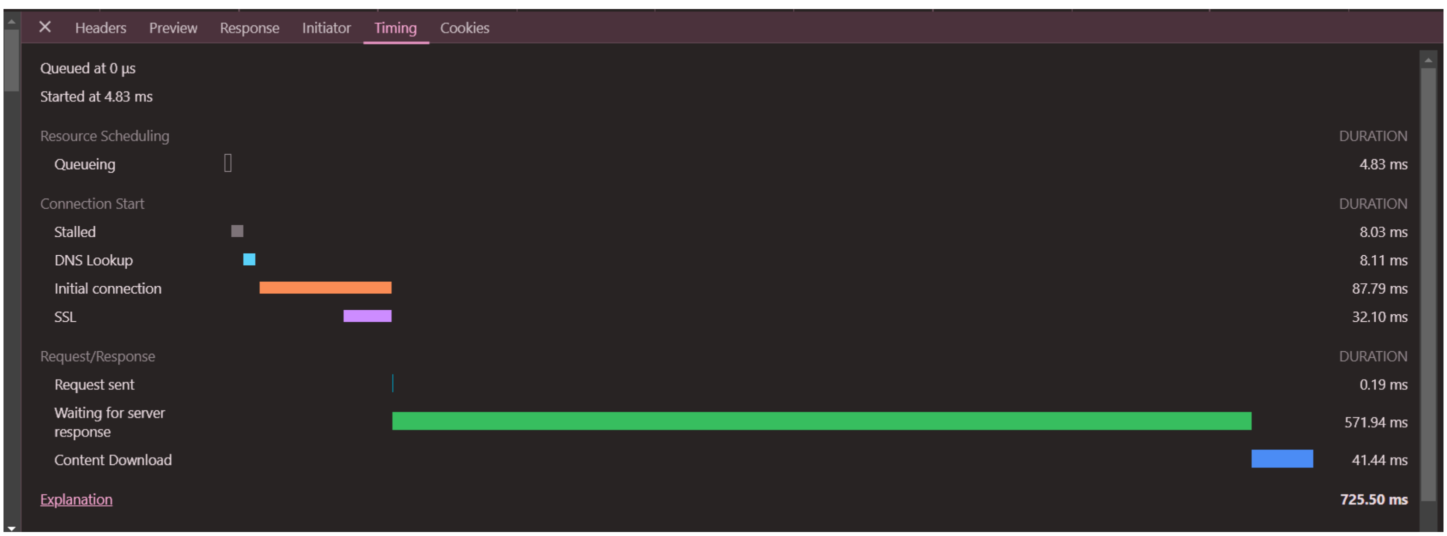
- Forbidden indicates that the server understands the request but refuses to authorize it. This typically happens when the client does not have permission to access the requested resource, even if they are authenticated. It could be due to restricted access, lack of proper privileges, or server settings preventing certain actions, such as accessing certain files or directories.



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.

Performance metrics:-





Cookies used and associated flags:-

Headers Preview Response Initiator Timing Cookies

Request Cookies show filtered out request cookies

Name	Value	Domain	Path	Expires /...	Size	HttpOnly	Secure	SameSite	Partition...	Cross Site	Priority
__Host-user_session_same_site	X1wAXT02svN1NQBtfk2b5s6coyDY4x2d_1xOxq8dv4GyJ...	github.c...	/	2025-02...	77	✓	✓	Strict			Medium
_device_id	8d67c96298b4b7a7c9b016d8168908ab	github.c...	/	2026-01...	42	✓	✓	Lax			Medium
_gh_sess	v6YcIr9%2Fc8avJjvPaVyL9LvfTbkF3HNljZTwfTp3iziQJ33X...	github.c...	/	Session	462	✓	✓	Lax			Medium
_octo	GH1.1.1363154166.1736355376	.github.c...	/	2026-01...	32		✓	Lax			Medium
color_mode	%7B%22color_mode%22%3A%22auto%22%2C%22light...	.github.c...	/	Session	214		✓	Lax			Medium
cpu_bucket	xlg	.github.c...	/	Session	13		✓	Lax			Medium
dotcom_user	Zeenu03	.github.c...	/	2026-01...	18	✓	✓	Lax			Medium
logged_in	yes	.github.c...	/	2026-01...	12	✓	✓	Lax			Medium
preferred_color_mode	dark	.github.c...	/	Session	24		✓	Lax			Medium
saved_user_sessions	145782990%3AX1wAXT02svN1NQBtfk2b5s6coyDY4x2d...	github.c...	/	2025-04...	79	✓	✓	Lax			Medium
tz	Asia%2FCalcutta	.github.c...	/	Session	17		✓	Lax			Medium
user_session	X1wAXT02svN1NQBtfk2b5s6coyDY4x2d_1xOxq8dv4GyJ...	github.c...	/	2025-02...	60	✓	✓	Lax			Medium

Response Cookies

Name	Value	Domain	Path	Expires /...	Size	HttpOnly	Secure	SameSite	Partition...	Cross Site	Priority
_gh_sess	zvJqhiY6IzH%2FPz7SzgT4vfnoK6yUtYqdkyTSZ4iwq3LUY...	github.c...	/	Session	501	✓	✓	Lax			Medium

Browser Name: Google Chrome
Performance Metrics:

- **Largest Contentful Paint (LCP)**:- LCP measures how quickly the largest content element (like an image or text) becomes visible on the page. The 75th percentile in field data represents the value at which 75% of users experienced this performance or better.
- **Cumulative Layout Shift (CLS)**:- CLS measures the visual stability of the page. It tracks unexpected shifts in the layout, which can disrupt the user's experience. A value below 0.1 is considered good, with lower values being better.
- **Interaction to Next Paint (INP)**:- INP measures the time it takes for the page to respond to user interactions. A lower value (like 79 ms) indicates a fast interaction response time, which is a good sign for user experience.

Timing Metrics:-

- **Queuing**:- The time the request spends in the browser's queue before it is sent.
- **Stalled:-** Time spent waiting for the request to start due to network conditions or resource contention.
- **DNS lookup:-** Time taken to resolve the domain name to an IP address.
- **Initial Connection:-** Time taken to establish the connection with the server (including the TCP handshake).
- **SSL:-** Time spent in setting up a secure SSL/TLS connection.
- **Request Sent:-** Time it takes for the browser to send the HTTP request to the server.
- **Waiting for server response:-** Time spent waiting for the server to respond after the request was sent.
- **Content Download:-** Time taken to download the content from the server.