

CS 331 Computer Networks

Assignment - 1

Github link: [CN_Assignment1](#)

Part1: Metrics and plots:

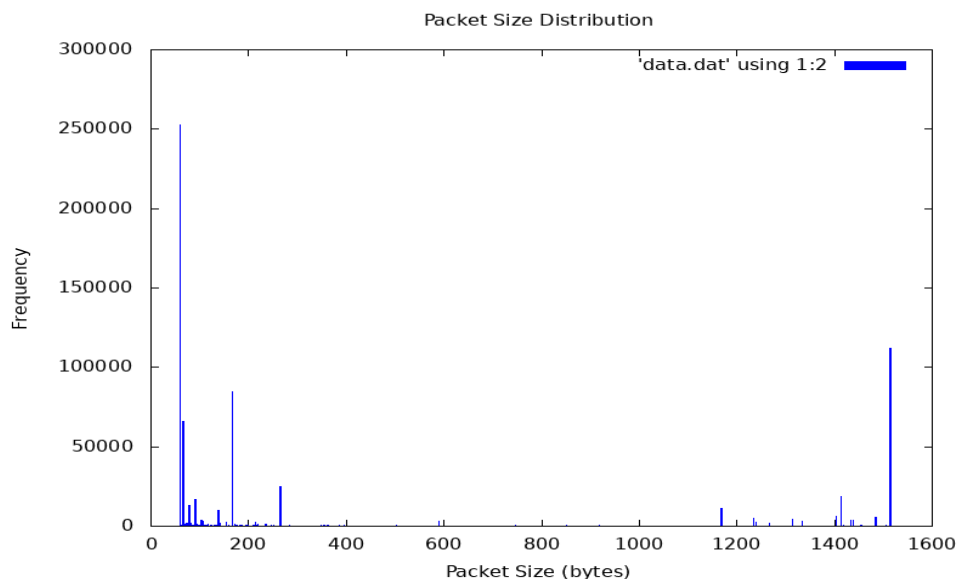
Q1. Find the total amount of data transferred (in bytes), the total number of packets transferred, and the minimum, maximum, and average packet sizes. Also, show the distribution of packet sizes (e.g., by plotting a histogram of packet sizes).

We have analyzed the packet capture file (PCAP) in C++ to calculate and visualize packet size statistics. We used the libpcap library to read the PCAP file. For each packet, it records the size, updating total bytes, packet count, minimum, and maximum sizes. It also calculates the average packet size. After processing all packets, we got the final output, that is the calculated metrics (total bytes, packets, min/max/average size). Finally, a histogram of the packet size distribution using gnuplot. This visualization helps understand the common packet sizes and their frequency within the captured network traffic.

OUTPUT:

```
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ g++ main.cpp -o main -lpcap
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ sudo ./main
Total Bytes: 355622618
Total Packets: 792151
Min Packet Size: 54
Max Packet Size: 1514
Average Packet Size: 448.933
Histogram saved as 'histogram.png'.
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ |
```

Histogram for the above output:

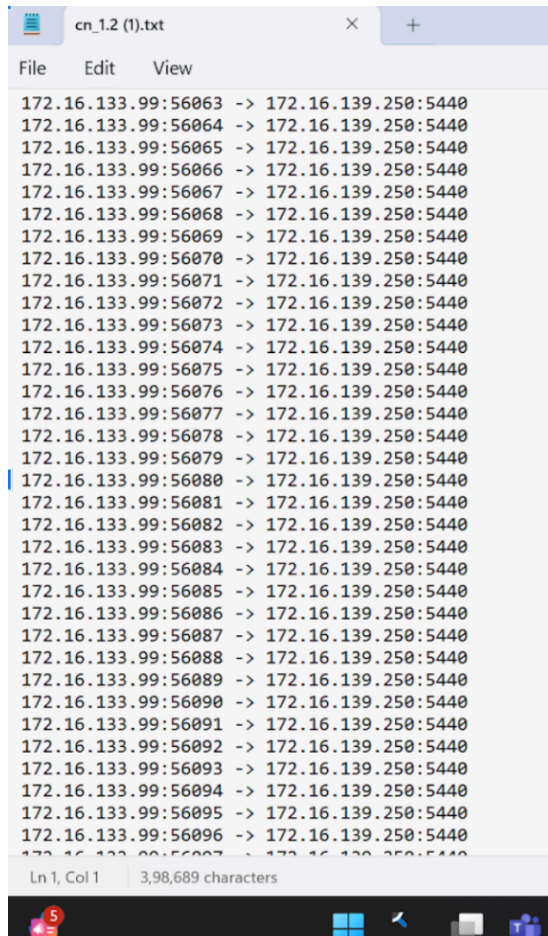


Q2. Find unique source-destination pairs (source IP:port and destination IP:port) in the captured data.

The C++ code analyzes the PCAP file to identify and count unique TCP and UDP connections. It uses the libpcap library to read the packet capture file. For each packet, it extracts the source and destination IP addresses and ports, handling both TCP and UDP protocols. It stores these connection details (source IP, source port, destination IP, destination port) in a `std::set` called `uniqueConnections`. The `std::set` automatically ensures that only unique connections are stored. After processing all packets, the code prints the list of unique source-destination IP address and port pairs found in the PCAP file. It effectively provides a summary of the distinct network conversations captured in the trace.

OUTPUT :

We stored the output in a txt file which contains all the unique pairs.

A screenshot of a text editor window titled 'cn_1.2 (1).txt'. The window displays a list of unique source-destination IP address and port pairs. Each line represents a unique connection, showing the source IP and port followed by an arrow and the destination IP and port. The list starts with '172.16.133.99:56063 -> 172.16.139.250:5440' and continues down to '172.16.133.99:56096 -> 172.16.139.250:5440'. The status bar at the bottom indicates 'Ln 1, Col 1' and '3,98,689 characters'.

(The output txt file is in the github repo)

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.

Output:

The C++ code captures and analyzes network packets from a pcap file, specifically focusing on TCP and UDP packets. The `analyzePacket` function processes each packet, extracting its source and destination IP addresses and port numbers, and tracks the flow count and data transferred per connection. It also updates the total packets and bytes processed. The `printPerformanceMetrics` function calculates and displays

performance metrics, such as packets per second (PPS) and megabits per second (Mbps). The main function opens the pcap file, processes each packet using the analyzePacket function, and prints the performance metrics, flow counts, and the connection with the most data transferred. It provides a detailed analysis of network traffic, helping identify high-traffic connections and overall network performance. The use of standard libraries, such as pcap.h and chrono, ensures efficient packet processing and time management.

A part of the output is here in a screenshot: (complete output in github repository)

```
Source IP Flows:
97.239.144.76 -> 1 flows
81.131.131.4 -> 1 flows
81.131.146.6 -> 1 flows
67.131.131.6 -> 1 flows
52.239.144.76 -> 1 flows
80.239.102.76 -> 1 flows
80.131.131.6 -> 1 flows
170.170.170.170 -> 1 flows
64.246.6.110 -> 1 flows
80.239.69.76 -> 1 flows
81.131.131.70 -> 1 flows
158.131.131.6 -> 1 flows
115.0.131.6 -> 1 flows
166.131.131.6 -> 1 flows
37.115.0.76 -> 1 flows
83.140.65.130 -> 5 flows
81.131.131.68 -> 1 flows
80.239.66.76 -> 1 flows
81.163.131.6 -> 1 flows
192.168.10.50 -> 6 flows
213.199.179.162 -> 1 flows
157.56.236.134 -> 13 flows
207.171.163.152 -> 3 flows
71.13.48.32 -> 3 flows
23.45.65.51 -> 13 flows
```

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.

Output:

By running the file “sniffer2.py”, we have generated a file named captured_traffic.pcap by capturing the packets while running it,

Output by finding the speed using tcpreplay using the command: `sudo tcpreplay -i enp03 --mbps=1500000 captured_traffic.pcap`

```
Actual: 792151 packets (355622618 bytes) sent in 12.38 seconds
Rated: 28702867.6 Bps, 229.62 Mbps, 63935.76 pps
Statistics for network device: enp0s3
    Successful packets:      792151
    Failed packets:         0
    Truncated packets:      0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN): 0
set-iiton-vm@set-iiton-vm:~/Downloads$
```

Now, I have done using my program. The output is stored in a text file named “Packet_sniffer.txt”. We have analysed the output using python code and this is the output:

```
### Packet Rate (PPS) ###  
Average PPS: 8.512406  
Max PPS: 684.724000  
Min PPS: 0.003667
```

```
### Bandwidth (Mbps) ###  
Average Mbps: 0.873162  
Max Mbps: 79.343600  
Min Mbps: 0.000003
```

```
PS C:\Users\VENU GOPALROA\Desktop\
```

PART 2 CATCH ME IF YOU CAN:

Q1.How Many Login Attempts Were Made? (Hint : filter packet with ip 192.168.10.50)

Output by running the file part2.1.cpp:

```
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ g++ part2.1.cpp -o part2.1 -lpcap  
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ sudo ./part2.1  
Number of login attempts: 4
```

Q2.What are the credentials in the successful login attempt? (Hint : My password is secure password)

Output: Credentials in the successful login attempt:

```
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ g++ part2_2.cpp -o part2_2 -lpcap  
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ sudo ./part2_2  
Found successful login attempt!  
Successful login credentials:  
Username: admin  
Password: securepassword  
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ |
```

Q3.What is the client's source Port number when making the login attempt successful?

The client source port number is: 12345

```
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ g++ part2_2.cpp -o part2_2 -lpcap  
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ sudo ./part2_2  
Found successful login attempt!  
Q2. Successful login credentials:  
Username: admin  
Password: securepassword  
Q3. Client's source port: 12345
```

Q4.What is the total content length of all login attempt payloads?

```
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ g++ part2_4.cpp -o part2_4 -lpcap  
sirig@SIRI:/mnt/c/Users/sirig/OneDrive/Desktop/CN_A1$ sudo ./part2_4  
Total content length of all login attempts: 17669 bytes
```

PART3 CAPTURE THE PACKETS:

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.

I. TLSv1 (Transport Layer Security):

TLS is used to secure communications over a computer network, ensuring privacy and data integrity. TLSv1 is an older version, often used for securing protocols like HTTP (HTTPS), FTP, and others.

Layer: Application Layer (though it operates between Transport and Application layers).

RFC: RFC 2246 (for TLS 1.0).

II. QUIC (Quick UDP Internet Connections)

QUIC is a transport-layer protocol designed to replace TCP+TLS, improving web performance by reducing latency and enabling faster, more secure connections. It is widely used in HTTP/3, streaming services, and online gaming.

Layer: Transport Layer

RFC: RFC 9000

III. ARP (Address Resolution Protocol)

ARP is used to map an IP address to a MAC address within a local network, enabling devices to communicate over Ethernet or Wi-Fi.

Layer: Link Layer (OSI Layer 2)

RFC: RFC 826

IV. AJP13 (Apache JServ Protocol 1.3)

AJP13 is a binary protocol used for communication between web servers (e.g., Apache) and application servers (e.g., Tomcat), providing faster request forwarding than HTTP.

Layer: Application Layer

RFC: No official RFC

V. NTP (Network Time Protocol)

NTP is used to synchronize the clock of computers and devices over a network to a precise time reference. It ensures accurate timekeeping for applications like logging, authentication, and distributed systems.

Layer: Application Layer

RFC: RFC 5905

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.

i) Canarabank:

Request method: GET

IP Address: 107.162.160.8:443

Protocol: HTTP/1.1

Connection: keep-alive for request header and close for response header (The connection is non-persistent connection)

The screenshot shows the Chrome DevTools Network tab. The left sidebar lists 145 requests, with 6.1 MB transferred. The main panel shows the details of a selected request to canarabank.com. The 'General' tab is active, displaying the following information:

- Request URL: https://canarabank.com/
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 107.162.160.8:443
- Referrer Policy: origin

The 'Response Headers' tab is also visible, showing the following headers:

- HTTP/1.1 200 OK
- Cache-Control: public, max-age=36000
- Content-Type: text/html; charset=utf-8
- X-Content-Type-Options: nosniff
- X-XSS-Protection: 1; mode=block
- X-Frame-Options: SAMEORIGIN
- Referrer-Policy: no-referrer-when-downgrade
- Content-Security-Policy: default-src data: http s;; img-src * 'self' data: https;; style-src 'sel

custom.css	Accept-Encoding: gzip, deflate, br, zstd
simple-slider.css	Accept-Language: en-US,en;q=0.9,fr;q=0.8,ko;q=0.7
logo.webp	Cache-Control: max-age=0
icon-13.svg	Connection: keep-alive
MSME.png	Cookie: NSC_10.14.241.15_TTM=ffffffff0906ef154552

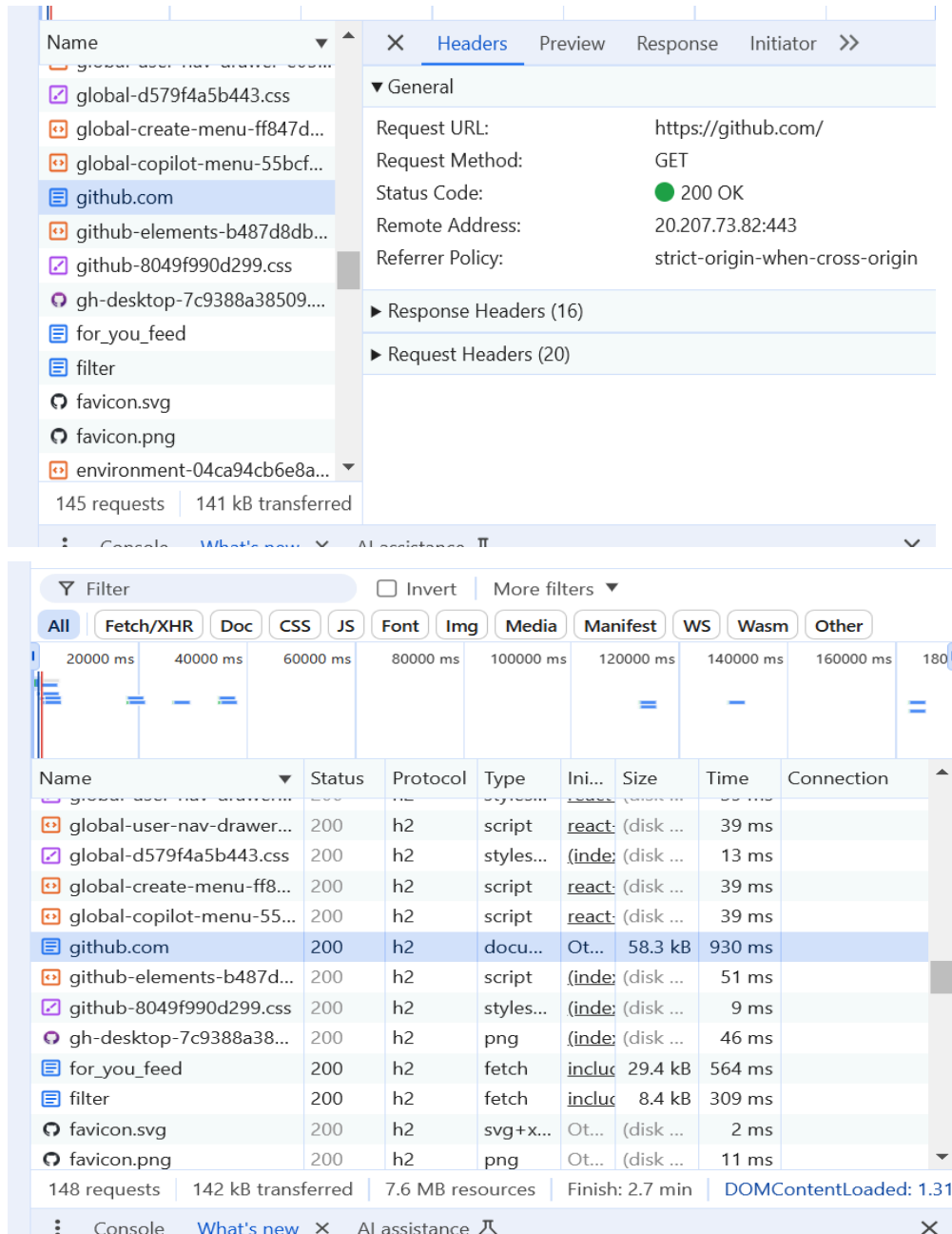
ii) Github:

Request method: GET

IP Address: 20.207.73.82:443

Protocol: HTTP/2

Connection: blank (for HTTP 2, the connection is persistent by default)



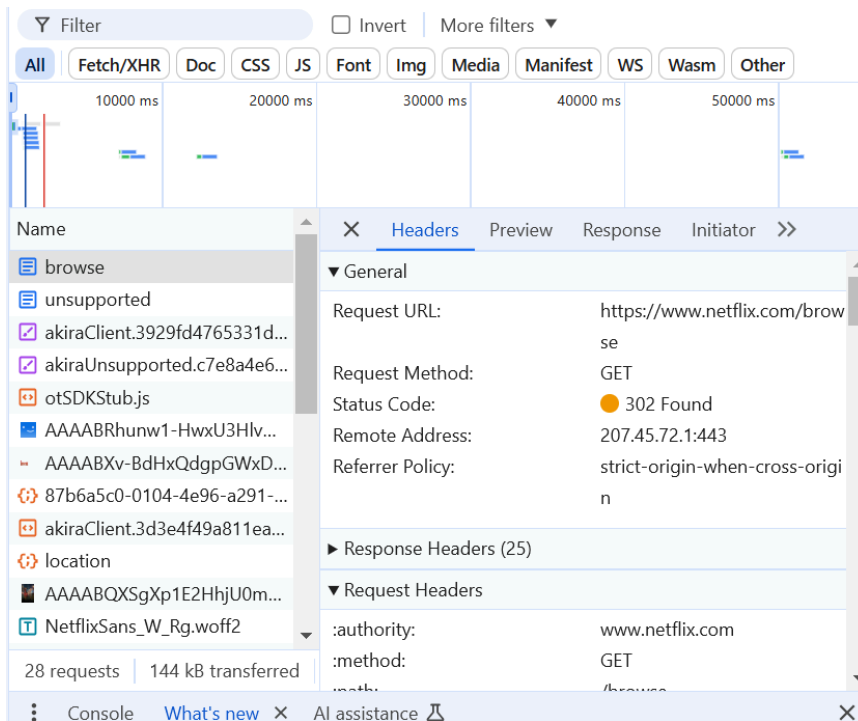
iii) Netflix:

Request method: GET

IP Address: 207.45.72.1:443

Protocol: HTTP/2

Connection: blank (for HTTP 2, the connection is persistent by default)



Name	Status	Protocol	Type	Ini...	Size	Time	Connection
browse	302	h2	docu...	Ot...	299 B	274 ms	
unsupported	200	h2	docu...	brow...	46.8 kB	563 ms	
akiraClient.3929fd476531d...	200	http/1.1	styles...	unsu...	(disk ...	8 ms	
akiraUnsupported.c7e8a4e6...	200	http/1.1	styles...	unsu...	466 B	32 ms	keep-alive
otSDKStub.js	200	http/1.1	script	unsu...	(disk ...	4 ms	
AAAABRhunw1-HwxU3Hlv...	200	http/1.1	png	unsu...	27.7 kB	53 ms	keep-alive
AAAABXv-BdHxQdgpGWxD...	200	http/1.1	webp	unsu...	4.8 kB	56 ms	keep-alive
87b6a5c0-0104-4e96-a...	200	http/1.1	xhr	otSDI...	(disk ...	2 ms	
akiraClient.3d3e4f49a811ea...	200	http/1.1	script	unsu...	(disk ...	123 ms	
location	200	h2	xhr	otSDI...	163 B	70 ms	
AAAABQXSgXp1E2HhjU0m...	200	http/1.1	webp	unsu...	58.5 kB	58 ms	keep-alive
NetflixSans_W_Rg.woff2	200	http/1.1	font	akira...	(disk ...	3 ms	

28 requests | 144 kB transferred | 15.7 MB resources | Finish: 50.37 s | DOMContentLoaded: 684

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 github.com,

In response headers,

Header field name	Value
-------------------	-------

cache-control:	max-age=0, private, must-revalidate
content-encoding:	gzip
content-type:	text/html; charset=utf-8
server:	GitHub.com

In request headers,

Header field name	Value
cache-control:	max-age=0
authority:	github.com
accept-encoding:	gzip, deflate, br, zstd
scheme:	https

HTTP error codes:

We didn't get any error codes while loading.

These are general error codes possible:

404 Not Found → Accessing

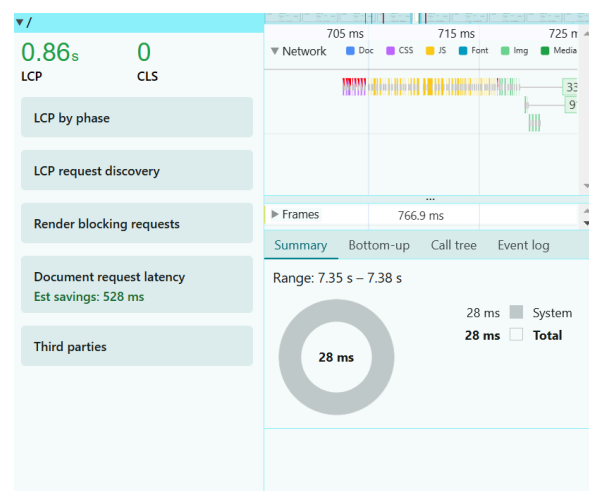
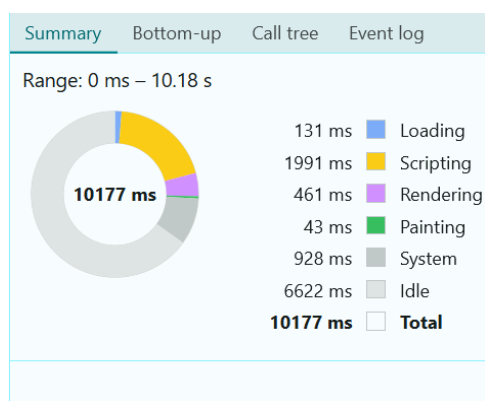
403 Forbidden → Trying to access restricted content without login

500 Internal Server Error → Sometimes occurs when the server has a temporary issue.

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: Chrome

Github.com



Response cookie: set-cookie:set-cookie:

_gh_sess=%2FcAAhe13R%2F5or5rFg43ls78ukGzCwztAlQNUAF9NgFh%2Bde29Z7DUojyxAPXZ6g6enhTB3Scjfxg7%2BBNRzgAx02qykG6OH2JkRO9Zjn%2FnRtGSW6S4o9Xh3Hf8m8GqrXTRsq5vtc9C4sRG6g4JkFiKuvSKaH3onF0iO%2BUuhrIzbSztKmRq1ALMZ53HMzCYUQUnhH%2Fc%2FD1MJo1Lyq2CY12aNkSMbwBQC11tw%2BZVBj0wAuJQLj7Yj6F4j0mjzIQT%2F6XEE NJhY8VhpcFfqFNBU%2FsRFlz2hqxlM5BIUJL4WUf%2FBGfdEFOdWD1k2kdRrM8wa%2FP6pnl%2Bwe%2FcbvNfxhts1%2BX0mBWKdim4OIG36ZJJA7TpulD%2BG6iXclvWnz2OXsa8Pyju--xBacQXUMboYgekYb--XVTcwM0Ji9Afv7LOm3c48w%3D%3D;

Flags: path=/; secure; HttpOnly; SameSite=Lax

Request Cookies:

_device_id=305f017b9af2277b853cf41b6d3a2902;
_octo=GH1.1.1736610246.1731553295;
saved_user_sessions=126078300%3AVgkPBf3q2uKwTxKHYpmru6RCkF6gcdrZZ92D72GMyCuu9Xc;
user_session=VgkPBf3q2uKwTxKHYpmru6RCkF6gcdrZZ92D72GMyCuu9Xcm;
__Host-user_session_same_site=VgkPBf3q2uKwTxKHYpmru6RCkF6gcdrZZ92D72GMyCuu9Xcm;
logged_in=yes;
dotcom_user=srijahn;
color_mode=%7B%22color_mode%22%3A%22auto%22%2C%22light_theme%22%3A%7B%22name%22%3A%22light%22%2C%22color_mode%22%3A%22light%22%7D%2C%22dark_theme%22%3A%7B%22name%22%3A%22dark%22%2C%22color_mode%22%3A%22dark%22%7D%7D%7D;
cpu_bucket=lg;
preferred_color_mode=light;
tz=Asia%2FCalcutta;
GHCC=Required:1-Analytics:1-SocialMedia:1-Advertising:1;
MicrosoftApplicationsTelemetryDeviceId=0bc1a5c8-8d35-42c4-9f20-08402e524620;
MSFPC=GUID=f794d0c2cc4e4d43a6dfe92331b53d62&HASH=f794&LV=202411&V=4&LU=1731218015878;
_gh_sess=nQRounRHHYizZ%2FXJM6HGPNYQtGdTC4WiMdSOTpBRqpusJcraBy6BJrbdp2u5O2Un2mSmRnpPB13apNqMi3m%2BSn1P%2Bo%2BuISxa8TpX4wvuRI9I%2FJlxxu5I35vuG8OdUojaZzVlbdH5kkOSQ4veZG0SvYUg9KYqk%2FYsqBXDUUI9iHqp2nVm%2F4W7qgb6gbBCPoPAbn8PXrqw3j0qUoavKp9GrolHOS1Rfi8mXhNM%2Bvwq6kNMS4q0jjnhxr2U64LDJkwve219YN3kuAC71hYmv9WrDernyfEBPVGbJEpkpKRT%2BohM4Ba7GCzqt%2FcgoiWLIY6fJUyPAx6u06JOzNo1qCvjLB7tvdwsPXKAqL18GjBiI%2F7Na5cevUICHLnQfionhoEI--EoOnQCnfJqM3JjrJ--AI8Zqe9J2r%2BqoxpUgCuXDQ%3D%3D