

Computer Networks

CS-331

Assignment-1

Team ID: 5

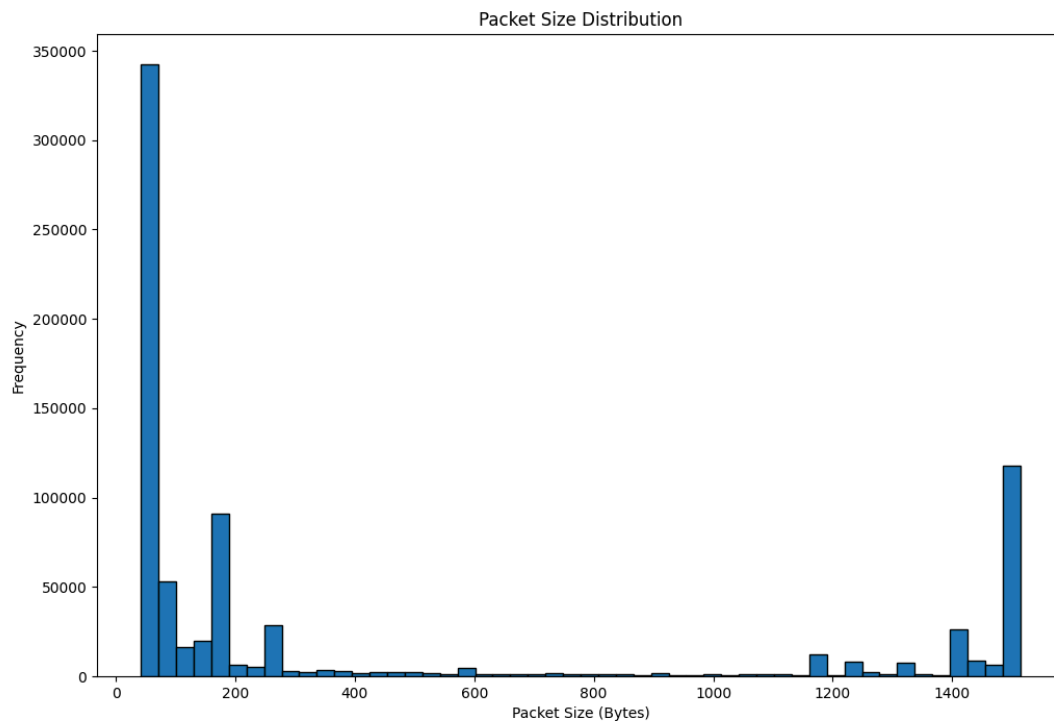
Daksh Jain(22110066), Harshit(22110095)

Q1.

a.

- Total Data Transferred: 364640811 bytes
- Total Packets Transferred: 805996 Packets
- Min Packet Size: 42 bytes
- Max Packet Size: 1514 bytes
- Average Packet Size: 452.41 bytes

Histogram of packet sizes:



b.

- Number of unique source-destination pairs: **41034**
- All unique source-destination pairs saved in part1_b.txt ([Uploaded in github](#))

c.

- All source IP flows saved in part1_c_source.txt ([Uploaded in github](#))
- All destination IP flows saved in part1_c_dest.txt ([Uploaded in github](#))
- Top Data Transferring Pair: **172.16.133.95:49358 → 157.56.240.102:443** transferred **17342229** bytes

The following screenshot confirms the stated values.

```
● dj@Endys cn_assign_1 % python3 -u "/Users/dj/Desktop/Acads/Sem 6/CN
/assignments/cn_assign_1/part1.py"
Total Data Transferred(Bytes): 364640811
Total Packets Transferred: 805996
Min Packet Size: 42 bytes
Max Packet Size: 1514 bytes
Average Packet Size: 452.41 bytes

Number of unique source-destination pairs: 41034

All unique source-destination pairs saved to part1_b.txt

All source IP flows saved to part1_c_source.txt

All destination IP flows saved to part1_c_dest.txt

Top Data Transferring Pair:
172.16.133.95:49358 → 157.56.240.102:443 transferred 17342229 bytes
```

Code: https://github.com/dkshjn/cn_assign_1/blob/main/part1.py

d.

1. On single device:

The maximum speed at which the program successfully captures packets without data loss when both tcpreplay and the capture program run on the same machine is:

- **1300 packets per second (pps)**
- **4.7 megabits per second (Mbps)**

The following screenshots validate these results:

1. **First Screenshot:** Shows total packets processed (**805996**), data transferred (364640870 bytes), and capture rate (**1300.00 pps** and **4.70 Mbps**).

Given image is output of the command:

sudo tcpreplay -i bridge100 --pps=1300 /Users/dj/Downloads/5.pcap

```
Actual: 805996 packets (364640870 bytes) sent in 619.99 seconds
Rated: 588133.3 Bps, 4.70 Mbps, 1300.00 pps
Flows: 41680 flows, 67.22 fps, 805297 unique flow packets, 454 unique non-flow p
ackets
Statistics for network device: bridge100
    Successful packets:      805996
    Failed packets:         0
    Truncated packets:      0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN): 0
```

2. **Second Screenshot:** Confirms consistent capture performance with a slight variation in Mbps (4.69 Mbps) and pps (1298.59 pps).

Given image is output of the command:

sudo tcpreplay -i bridge100 --mbps=4.7 /Users/dj/Downloads/5.pcap

```
only 60 available
Actual: 805996 packets (364640870 bytes) sent in 620.66 seconds
Rated: 587499.8 Bps, 4.69 Mbps, 1298.59 pps
Flows: 41680 flows, 67.15 fps, 805297 unique flow packets, 454 unique non-flow p
ackets
Statistics for network device: bridge100
    Successful packets:      805996
    Failed packets:         0
    Truncated packets:      0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN): 0
```

3. **Third Screenshot:** Displays the final captured packet logs, ensuring no packet loss the captured packets match the total packets in the pcap file.

```
Captured Packet #805989: Ether / IP / TCP 172.16.255.1:10655 > 128.241.90.211:http A
Captured Packet #805990: Ether / IP / TCP 204.14.234.85:pcsync_https > 192.168.3.131:57243 A / Padding
Captured Packet #805991: Ether / IP / TCP 65.54.95.68:http > 192.168.3.131:56434 A / Raw
Captured Packet #805992: Ether / IP / TCP 65.54.95.75:http > 192.168.3.131:56427 A / Raw
Captured Packet #805993: Ether / IP / TCP 192.168.3.131:57246 > 204.14.234.85:pcsync_https A
Captured Packet #805994: Ether / IP / TCP 207.46.96.145:5480 > 10.0.2.15:utsftp PA / Raw
Captured Packet #805995: Ether / IP / TCP 65.54.95.75:http > 192.168.3.131:56427 A / Padding
Captured Packet #805996: Ether / IP / TCP 192.168.3.131:56133 > 65.54.95.216:http A

Capture complete.
```

Sniffer program code: https://github.com/dkshjn/cn_assign_1/blob/main/sniff.py

2. Two devices (Windows-macOS)

The maximum speed at which the program successfully captures packets without data loss when both tcpreplay and the capture program run on the same machine is:

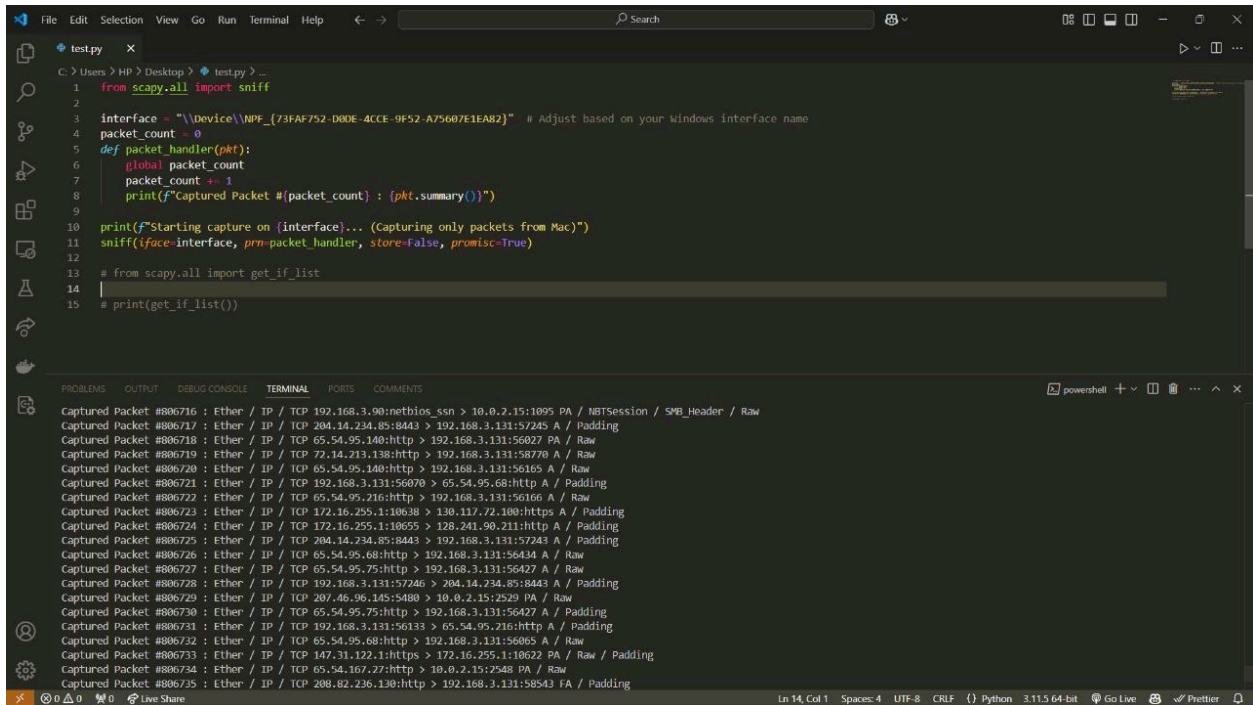
- **800 packets per second (pps)**
- **2.9 megabits per second (Mbps)**

The following screenshot validate the results:

1. macOS(Sender)

```
Actual: 805996 packets (364640870 bytes) sent in 1007.49 seconds
Rated: 361928.6 Bps, 2.89 Mbps, 800.00 pps
Flows: 41680 flows, 41.36 fps, 805297 unique flow packets, 454 unique non-flow packets
Statistics for network device: en9
    Successful packets:      805996
    Failed packets:         0
    Truncated packets:      0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN): 0
```

2. Receiver(Windows)



```
File Edit Selection View Go Run Terminal Help
test.py
C:\Users\HP\Desktop> test.py
1 from scapy.all import sniff
2
3 interface = "\\Device\\NPF_{73FAF752-D80E-4CCE-9F52-A75607E1EA82}" # Adjust based on your windows interface name
4 packet_count = 0
5 def packet_handler(pkt):
6     global packet_count
7     packet_count += 1
8     print(f"Captured Packet #{packet_count} : {pkt.summary()}")
9
10 print(f"Starting capture on {interface}... (Capturing only packets from Mac)")
11 sniff(iface=interface, prn=packet_handler, store=False, promisc=True)
12
13 # from scapy.all import get_if_list
14
15 # print(get_if_list())
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
Captured Packet #806716 : Ether / IP / TCP 192.168.3.90:netbios_ssn > 10.0.2.15:1095 PA / NBTSessio / SMB_Header / Raw
Captured Packet #806717 : Ether / IP / TCP 204.14.234.85:8443 > 192.168.3.131:57245 A / Padding
Captured Packet #806718 : Ether / IP / TCP 65.54.95.140:http > 192.168.3.131:56027 PA / Raw
Captured Packet #806719 : Ether / IP / TCP 72.14.213.138:http > 192.168.3.131:58770 A / Raw
Captured Packet #806720 : Ether / IP / TCP 65.54.95.140:http > 192.168.3.131:56165 A / Raw
Captured Packet #806721 : Ether / IP / TCP 192.168.3.131:56070 > 65.54.95.68:http A / Padding
Captured Packet #806722 : Ether / IP / TCP 65.54.95.216:http > 192.168.3.131:56166 A / Raw
Captured Packet #806723 : Ether / IP / TCP 172.16.255.1:10638 > 130.117.72.100:https A / Padding
Captured Packet #806724 : Ether / IP / TCP 172.16.255.1:10655 > 128.241.90.211:http A / Padding
Captured Packet #806725 : Ether / IP / TCP 204.14.234.85:8443 > 192.168.3.131:57243 A / Padding
Captured Packet #806726 : Ether / IP / TCP 65.54.95.68:http > 192.168.3.131:56434 A / Raw
Captured Packet #806727 : Ether / IP / TCP 65.54.95.75:http > 192.168.3.131:56427 A / Raw
Captured Packet #806728 : Ether / IP / TCP 192.168.3.131:57246 > 204.14.234.85:8443 A / Padding
Captured Packet #806729 : Ether / IP / TCP 207.46.96.145:5080 > 10.0.2.15:2529 PA / Raw
Captured Packet #806730 : Ether / IP / TCP 65.54.95.75:http > 192.168.3.131:56427 A / Padding
Captured Packet #806731 : Ether / IP / TCP 192.168.3.131:56133 > 65.54.95.216:http A / Padding
Captured Packet #806732 : Ether / IP / TCP 65.54.95.68:http > 192.168.3.131:56065 A / Raw
Captured Packet #806733 : Ether / IP / TCP 147.31.122.1:https > 172.16.255.1:10622 PA / Raw / Padding
Captured Packet #806734 : Ether / IP / TCP 65.54.167.27:http > 10.0.2.15:2548 PA / Raw
Captured Packet #806735 : Ether / IP / TCP 208.82.236.130:http > 192.168.3.131:58543 FA / Padding
```

Methodology:

To ensure accurate packet replay and capture between two devices, we established a direct Ethernet connection and configured both systems accordingly.

1. Setting Up the Ethernet Connection

A LAN cable was used to connect the **Mac (sender)** and **Windows (receiver)**.

Wi-Fi was disabled on both devices to ensure that all traffic was routed through the Ethernet connection.

2. Assigning Static IP Addresses

To establish direct communication, static IP addresses were manually configured on both systems:

1. On Mac (sender):

Run the following command on terminal to configure the ip:

```
sudo ifconfig en9 192.168.50.1 netmask 255.255.255.0 up
```

2. On Windows (receiver):

- a. Open **Network and Internet Settings**.
- b. Go to **Ethernet > Change Adapter Settings**.
- c. Right-click **Ethernet > Properties**.
- d. Set **IP Address: 192.168.50.2, Subnet Mask: 255.255.255.0**.
- e. Click **OK** to apply settings.

3. Testing Connectivity

To verify the connection, a **ping test** was conducted:

1. From Mac:

```
ping 192.168.50.2
```

2. From Windows:

```
ping 192.168.50.1
```

If both devices responded, the connection was working.

4. Running tcpreplay on Mac to Send Packets

Once the connection was established, **packet replay** was initiated from the Mac using tcpreplay over Ethernet.

To test packet transfer at a fixed rate (PPS):

- `sudo tcpreplay -i en9 --pps=800 /Users/dj/Downloads/5.pcap`

To test packet transfer at a fixed rate (Mbps):

- `sudo tcpreplay -i en9 --mbps=2.9 /Users/dj/Downloads/5.pcap`

5. Capturing Packets on Windows

With packet replay in progress, a sniffer program was executed on the Windows machine to capture and analyze the incoming packets. The total number of packets received matched the number of packets sent, confirming successful transmission without packet loss.

Code: https://github.com/dkshjn/cn_assign_1/blob/main/sniff_two_device.py

Q2.

1.

- a. File Name Found: **networking_Questions.pdf**
- b. TCP Checksum: **35409**
- c. Source IP: **10.20.30.200**

2. Total Packets from 10.20.30.200: **30**

3. Phone Company Found: **Samsung**

- a. Port Used by Localhost: **1001**
- b. Total Packets from Localhost: **30**

```
● dj@Endys cn_assign_1 % python3 -u "/Users/dj/Desktop/Acads/Sem 6/CN/assignments/cn_assign_1/part2.py"
File Name Found: networking_Questions.pdf
TCP Checksum: 35409
Source IP: 10.20.30.200

Total Packets from 10.20.30.200: 30

Phone Company Found: Samsung
Port Used by Localhost: 1001

Total Packets from Localhost: 30
```

Code: https://github.com/dkshjn/cn_assign_1/blob/main/part2.py

Q3.

1.

1. SNMP (Simple Network Management Protocol)

- **Usage:** SNMP is used for managing and monitoring network devices such as routers, switches, and servers. It allows network administrators to collect information, configure devices, and detect faults.
- **Layer:** Application Layer
- **RFC:** RFC 1157

2. NTP (Network Time Protocol)

- **Usage:** NTP is used to synchronize the clocks of computers over a network. It ensures accurate timekeeping, which is critical for logging, authentication, and other time-sensitive operations.
- **Layer:** Application Layer
- **RFC:** RFC 5905

3. SIP (Session Initiation Protocol)

- **Usage:** SIP is used for initiating, maintaining, modifying, and terminating real-time sessions such as voice and video calls over IP networks. It is a key protocol in VoIP (Voice over IP) systems.
- **Layer:** Application Layer
- **RFC:** RFC 3261

4. DHCP (Dynamic Host Configuration Protocol)

- **Usage:** DHCP is used to automatically assign IP addresses and other network configuration parameters (like subnet mask and default gateway) to devices on a network.
- **Layer:** Application Layer
- **RFC:** RFC 2131

5. LDAP (Lightweight Directory Access Protocol)

- **Usage:** LDAP is used to access and maintain distributed directory information services over an IP network. It is commonly used in directory services like Microsoft Active Directory.
- **Layer:** Application Layer
- **RFC:** RFC 4511


2.

a. canarabank.com


Request Line: GET / HTTP/1.1

HTTP Version: HTTP/1.1

Destination IP: 107.162.160.8

▼ General		
Request URL:	https://canarabank.com/	
Request Method:	GET	
Status Code:	● 200 OK	
Remote Address:	107.162.160.8:443	
Name	Status	Protocol
 canarabank.com	200	http/1.1

Connection: The connection is non-persistent (Close).

Name	Status	Protocol	Connection
 canarabank.com	200	http/1.1	close


b. github.com

Request Line: GET / HTTP/2


HTTP Version: HTTP/2

Destination IP: 20.207.73.8

▼ General	
Request URL:	https://github.com/
Request Method:	GET
Status Code:	● 200 OK
Remote Address:	20.207.73.82:443
Referrer Policy:	strict-origin-when-cross-origin

Name	Status	Protocol
 github.com	200	h2

Connection: For HTTP/2, the Connection field is empty(as shown in the image), and the connection is persistent by default, keeping it open unless explicitly closed.


Name	Status	Protocol	Connection
 github.com	200	h2	


c. netflix.com

Request Line: GET / HTTP/2


HTTP Version: HTTP/2

Destination IP: 54.246.79.9

▼ General	
Request URL:	https://www.netflix.com/in/
Request Method:	GET
Status Code:	 200 OK
Remote Address:	52.214.181.141:443
Referrer Policy:	strict-origin-when-cross-origin

Name	Status	Protocol
 in/	200	h2

Connection: For HTTP/2, the Connection field is empty(as shown in the image), and the connection is persistent by default, keeping it open unless explicitly closed.

Name	Status	Protocol	Connection
 in/	200	h2	

b. **Browser Used** : Google Chrome

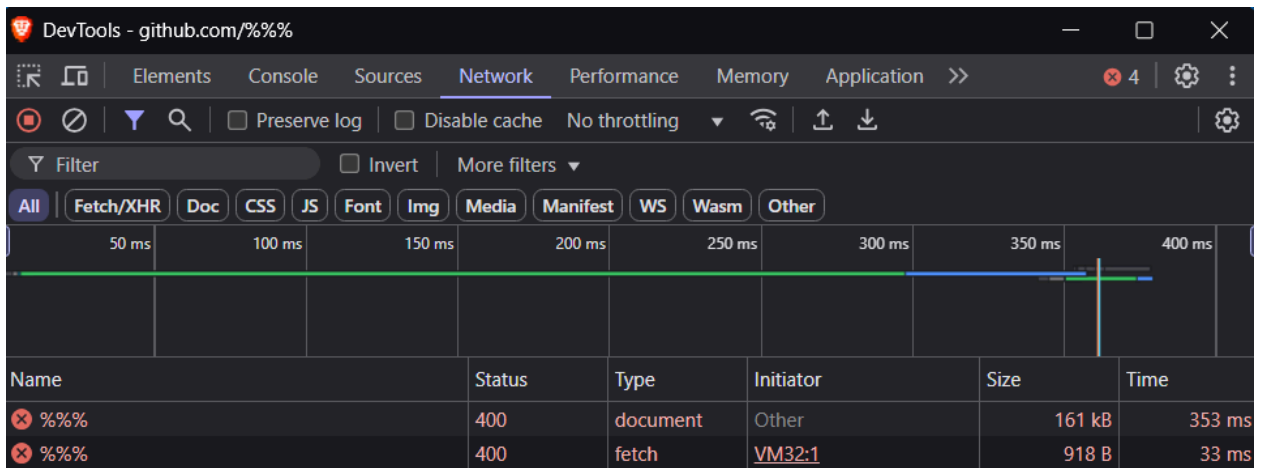
Response Headers :

- accept-ch: Sec-CH-UA-Platform-Version,Sec-CH-UA-Model
- content-encoding: gzip
- cache-control: no-cache, no-store, must-revalidate

Request Headers :

- :authority: www.netflix.com
- :method: GET
- :scheme: https

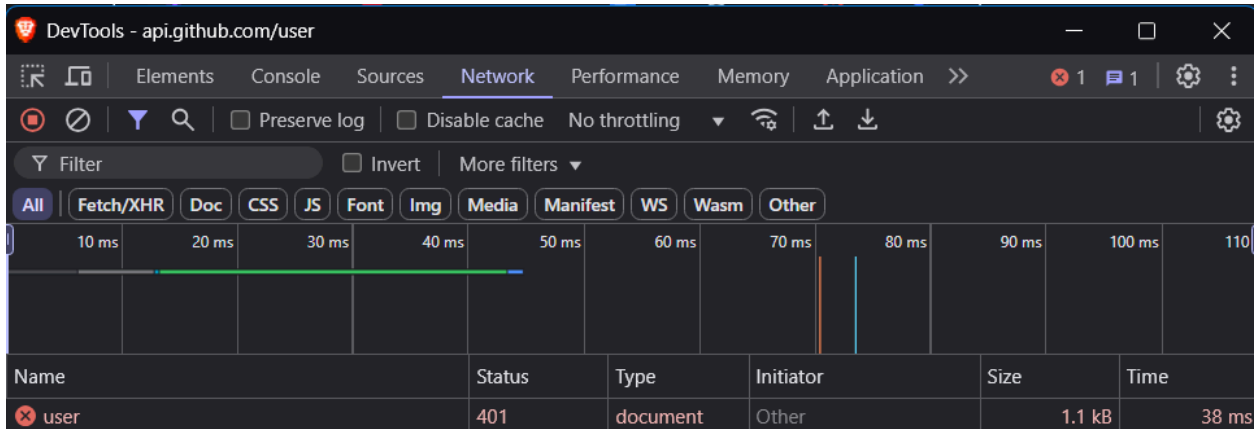
400 Bad Request: This error occurs when the server cannot understand the request due to malformed syntax.



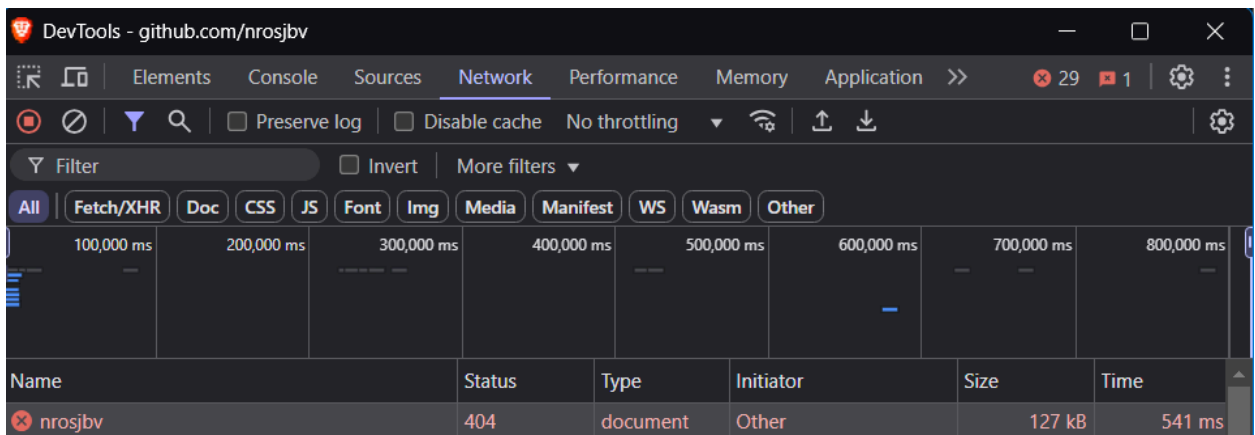
The screenshot shows the Chrome DevTools Network tab for the URL 'github.com/%%'. It displays two failed requests. The first request is a 'document' type with a status of 400, a size of 161 kB, and a time of 353 ms. The second request is a 'fetch' type with a status of 400, a size of 918 B, and a time of 33 ms. The 'Initiator' for the second request is 'VM32:1'. The 'Filter' dropdown is set to 'All', and the 'Preserve log' checkbox is checked.

Name	Status	Type	Initiator	Size	Time
%%%	400	document	Other	161 kB	353 ms
%%%	400	fetch	VM32:1	918 B	33 ms

401 Unauthorized: This error occurs when authentication is required but missing or incorrect.

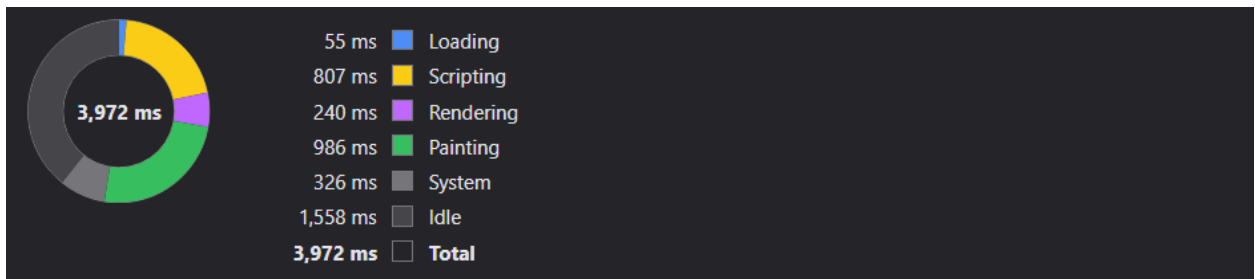


404 Not Found: This error means that the server cannot find the requested resource.



c. **Browser Used :** Brave

Website : github.com



Total Request : 135

Transferred : 61.2kB

Resources : 9.5 MB

Cookies :

Request Headers :

Cookie Name	HttpOnly	Secure	SameSite
__Host-user_session_same_site	✓	✓	Strict
_device_id	✓	✓	Lax
_gh_sess	✓	✓	Lax
_octo		✓	Lax
color_mode		✓	Lax
cpu_bucket		✓	Lax
dotcom_user	✓	✓	Lax
logged_in	✓	✓	Lax
preffered_color_mode		✓	Lax
saved_user_sessions	✓	✓	Lax
tz		✓	Lax
tz	✓	✓	Lax
user_session	✓	✓	Lax

Response Headers :

Cookie Name	HttpOnly	Secure	SameSite
_gh_sess	✓	✓	Lax

HttpOnly:

- If checked (✓), the cookie is inaccessible to JavaScript (e.g., via document.cookie).
- Helps protect against cross-site scripting (XSS) attacks.

Secure:

- If checked (✓), the cookie is only sent over HTTPS connections.
- Ensures the cookie is not transmitted over insecure HTTP.

SameSite:

- Determines how the cookie is sent with cross-site requests.
- Strict: Cookie is only sent in a first-party context.
- Lax: Cookie is sent with top-level navigations (e.g., clicking a link) but not with cross-site requests (e.g., images or scripts).