

CS:433 Computer Networks

Assignment 1

Output & Observations

Q1. The raw socket is created and bound to the eth0 interface. The while loop parses the Ethernet and IP headers while capturing packets. The source IP, source port, destination IP, destination port, and protocol are extracted from an IPv4 and TCP packet, and a unique flow key is formed. If the flow is distinct, its details are written to a CSV file and included in the dictionary.

```
20 while True:
21     packet = s.recvfrom(65565)
22     raw_data,addr = packet
23
24     ethernet_header = raw_data[0:14]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Warning in send_packets.c:send_packets() line 489:
Unable to send packet: Error with PF_PACKET send() [16027]: Message too long (errno = 90)
Warning in send_packets.c:send_packets() line 489:
Warning in send_packets.c:send_packets() line 489:
Unable to send packet: Error with PF_PACKET send() [16031]: Message too long (errno = 90)
Warning in send_packets.c:send_packets() line 489:
Unable to send packet: Error with PF_PACKET send() [16033]: Message too long (errno = 90)
Actual: 13713 packets (3571601 bytes) sent in 5.71 seconds
Rated: 624942.3 Bps, 4.99 Mbps, 2399.43 pps
Flows: 1027 flows, 179.69 fps, 16204 unique flow packets, 2 unique non-flow packets
Statistics for network device: eth0
Successful packets: 13713
Failed packets: 2493
Truncated packets: 0
Retried packets (ENOBUFFS): 0
Retried packets (EAGAIN): 0

(kali@kali) ~/CN/Assignment 1
```

In the above image, you can see that many packets are failing as the default “mtu” is set to 1500, so it drops the packet with a size greater than 1500 bytes. We then increased the MTU to 16055 bytes as it was the limit of my device.

```
Source IP: 18.66.40.93, Destination IP: 192.168.122.197
Source Port: 443, Destination Port: 32832
Source MAC: 52:54:00:45:3f:66, Destination MAC: 52:54:00:8b:df:95, EtherType: 0x800
Source IP: 192.168.122.197, Destination IP: 18.66.40.93
Source Port: 32832, Destination Port: 443
Source MAC: 52:54:00:8b:df:95, Destination MAC: 52:54:00:45:3f:66, EtherType: 0x800
Source IP: 54.192.166.121, Destination IP: 192.168.122.197
Source Port: 443, Destination Port: 35402
Source MAC: 52:54:00:8b:df:95, Destination MAC: 52:54:00:45:3f:66, EtherType: 0x800
Source IP: 54.192.166.121, Destination IP: 192.168.122.197
Source Port: 443, Destination Port: 35402
Source MAC: 52:54:00:45:3f:66, Destination MAC: 52:54:00:8b:df:95, EtherType: 0x800
Source IP: 192.168.122.197, Destination IP: 142.250.183.195
Source Port: 60516, Destination Port: 80
Source MAC: 52:54:00:8b:df:95, Destination MAC: 52:54:00:45:3f:66, EtherType: 0x800
Source IP: 142.250.183.195, Destination IP: 192.168.122.197
Source Port: 80, Destination Port: 60516
Source MAC: 52:54:00:45:3f:66, Destination MAC: 52:54:00:8b:df:95, EtherType: 0x800
Source IP: 192.168.122.197, Destination IP: 54.192.166.121
Source Port: 35402, Destination Port: 443
Source MAC: 52:54:00:8b:df:95, Destination MAC: 52:54:00:45:3f:66, EtherType: 0x800
Source IP: 52.16.32.113, Destination IP: 192.168.122.197
Source Port: 443, Destination Port: 39376
Source MAC: 52:54:00:45:3f:66, Destination MAC: 52:54:00:8b:df:95, EtherType: 0x800
Source IP: 192.168.122.197, Destination IP: 52.16.32.113
Source Port: 39376, Destination Port: 443
AC
Packet capture stopped.

win 501, options [nop,nop,TS val 3053662763 ecr 4145644460], length 0
13:38:44.1692812324 IP 54.192.166.121.443 > 192.168.122.197.35402: Flags [.], ack 109
9, win 135, options [nop,nop,TS val 1036065892 ecr 3026877158], length 0
13:38:44.1692812324 IP 54.192.166.121.443 > 192.168.122.197.35402: Flags [P.], seq 65
67:6606, ack 1099, win 135, options [nop,nop,TS val 1036065892 ecr 3026877158], lengt
h 39
13:38:44.1692812324 IP 192.168.122.197.60516 > 142.250.183.195.80: Flags [.], ack 703
, win 501, options [nop,nop,TS val 658567548 ecr 2868637004], length 0
13:38:44.1692812324 IP 142.250.183.195.80 > 192.168.122.197.60516: Flags [.], ack 420
, win 4096, options [nop,nop,TS val 2868637045 ecr 0], length 0
13:38:44.1692812324 IP 192.168.122.197.35402 > 54.192.166.121.443: Flags [.], ack 660
6, win 501, options [nop,nop,TS val 3026877259 ecr 1036065892], length 0
13:38:45.1692812325 STP 802.1d, Config, Flags [none], bridge-id 8000.52:54:00:45:3f:6
6.8001, length 35
13:38:45.1692812325 IP 52.16.32.113.443 > 192.168.122.197.39376: Flags [P.], seq 1405
0:14087, ack 8808, win 3216, length 37
13:38:45.1692812325 IP 192.168.122.197.39376 > 52.16.32.113.443: Flags [.], ack 14087
, win 501, length 0
Actual: 16142 packets (13267518 bytes) sent in 10.80 seconds
Rated: 1227587.6 Bps, 9.82 Mbps, 1493.55 pps
Flows: 1027 flows, 95.02 fps, 16204 unique flow packets, 2 unique non-flow packets
Statistics for network device: eth0
Successful packets: 16142
Failed packets: 64
Truncated packets: 0
Retried packets (ENOBUFFS): 0
Retried packets (EAGAIN): 0
```

For part b, the “PartI_reversedns.csv” contains all the hostnames and IP addresses in the “0.pcap” file. From the CSV file, five hostnames and their IP addresses are provided in the table.

S.No	Hostname	IP Address
1	bom07s30-in-f14.1e100.net	142.250.183.14
2	a23-37-240-101.deploy.static.akamaitechnologies.com	23.37.240.101
3	cloudproxy10013.sucuri.net	192.124.249.13
4	ec2-54-246-192-162.eu-west-1.compute.amazonaws.com	54.246.192.162
5	sa.outbrain.com	66.225.223.95

Q2. The code provided by us is sufficient to get all the payloads in the form of a CSV file, and then you can easily search for each of the questions with their hints and references in that CSV file.

1. In this question, we filtered all the payload ASCII values of all the TCP packets and then screened the keyword “Flag” among them using the “grep” command.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Rated: 124842.5 Bps, 0.998 Mbps, 361.53 pps
Flows: 296 flows, 21.82 fps, 6860 unique flow packets, 13 unique non-flow packets
Statistics for network device: eth0
  Successful packets: 4904
  Failed packets: 1969
  Truncated packets: 0
  Retried packets (ENOBUFS): 0
  Retried packets (EAGAIN): 0

(kali@kali) - [~/CN/Assignment 1]
$ sudo python Part_2.py | grep Flag
here, this is not the Flag, skip this packet
..^....c?},...E..X....@...ggggiiii.g.q.....P. .i...Hi t
here, this is not the Flag, skip this packet
..^....c?},...E..X....@...jjjjllll.j.t.....P. .j...Hi t
here, this is not the Flag, skip this packet
..^....c?},...E..X....@...mmmmoooo.m.w.....P. .Q...Hi t
here, this is not the Flag, skip this packet
..^....c?},...E..3....@.../lclcefg...P. ....Flag
: Romeo
..^....c?},...E..X....@...Jp.....P. ....Hi t
here, this is not the Flag, skip this packet
..^....c?},...E..X....@...Fl.....P. ....Hi t
here, this is not the Flag, skip this packet
```

Ans: Flag: **Romeo**

2. In this question, we filtered all the payload ASCII values of all the TCP packets and then screened the keyword “secret” among them using the “grep” command.

```
(kali@kali) - [~/Desktop/Computer Networks]
$ sudo python Part_2.py | grep Secret
Connection: Secret: I find a way, not a excuse.

09:47:13.1692971233 IP 10.7.52.103 > 172.217.21.4: ICMP echo request, id 46169, seq 270, length 64
09:47:14.1692971234 IP 172.217.21.4 > 10.7.52.103: ICMP echo reply, id 46169, seq 270, length 64
Actual: 6873 packets (12562745 bytes) sent in 10.06 seconds
Rated: 1248381.8 Bps, 9.98 Mbps, 682.98 pps
Flows: 296 flows, 29.41 fps, 6860 unique flow packets, 13 unique non-flow packets
Statistics for network device: eth0
  Successful packets: 6873
  Failed packets: 0
  Truncated packets: 0
  Retried packets (ENOBUFS): 0
  Retried packets (EAGAIN): 0
```

Ans: Secret: **I find a way, not a excuse.**

3. In this question, we printed all the payload ASCII values of the TCP packets whose checksum was equal to “0xf436”; we got the message that your password is somewhere in this stream, meaning the password packet must have the same IP and port then we searched for the same IP in our CSV file. We got the packet with the payload text as the password: “Berlin”. It was a dropped packet, as we had to update our code even to consider the dropped packet payload.

```
TCP Checksum:
Calculated Checksum: 0x5956

TCP Payload (Text):
GET /your-password-is-somewhere-in--this-stream HTTP/1.1

TCP Checksum:
Calculated Checksum: 0x0000

TCP Payload (Text):

46169, seq 269, length 64
09:47:13.1692971233 IP 172.217.21.4 > 10.7.52.103: ICMP echo reply, id 46169, seq 269, length 64
09:47:13.1692971233 IP 10.7.52.103 > 172.217.21.4: ICMP echo request, id 46169, seq 270, length 64
09:47:14.1692971234 IP 172.217.21.4 > 10.7.52.103: ICMP echo reply, id 46169, seq 270, length 64
Actual: 6873 packets (12562745 bytes) sent in 10.06 seconds
Rated: 1248279.6 Bps, 9.98 Mbps, 682.92 pps
Flows: 296 flows, 29.41 fps, 6860 unique flow packets, 13 unique non-flow packets
Statistics for network device: eth0
  Successful packets: 6873
  Failed packets: 0
  Truncated packets: 0
```

```

TCP Checksum:
Calculated Checksum: 0x5956

TCP Payload (Text):
GET / HTTP/1.1
Origin: www.cs433.com
User-Agent: PASSWORD-Berlin

46169, seq 270, length 64
09:47:14.1692971234 IP 172.217.21.4 > 10.7.52.103: ICMP echo reply, id 46
169, seq 270, length 64
Actual: 6873 packets (12562745 bytes) sent in 10.06 seconds
Rated: 1248279.6 Bps, 9.98 Mbps, 682.92 pps
Flows: 296 flows, 29.41 fps, 6860 unique flow packets, 13 unique non-flow
packets
Statistics for network device: eth0
Successful packets: 6873

```

Ans: Instructions:

1. **your -password-is-somewhere-in--this-stream HTTP/1.1**
2. **User-Agent: PASSWORD-Berlin**

4. In this question, we filtered all the payload ASCII values of all the TCP packets and then screened the IP “123.134.156.178” among them using the “grep” command. Then further, we added the value of both source and destination ports and used the grep command again to reach our resultant packet.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Rated: 124971.3 Bps, 0.999 Mbps, 361.91 pps
Flows: 296 flows, 21.84 fps, 6860 unique flow packets, 13 unique non-flow packets
Statistics for network device: eth0
Successful packets: 4904
Failed packets: 1969
Truncated packets: 0
Retried packets (ENOBUFS): 0
Retried packets (EAGAIN): 0

(kali@kali) - [~/CN/Assignment 1]
$ sudo python partII.py | grep -A 2 "10987"
Source IP:123.128.56.78 Source Port:10987 Destination IP:12.128.128.78 Destination port
:443 Length:107
b'\x00\x00*\x00\x01\xf6\xf8c?}\x1a\x08\x00\x00\x00\x00\x01\x00\x00\x00\x069\xfe(\x808N\
x8c\x80\x80N*\xeb\x01\xbb\x00\x00\x00\x00\x00\x00P\x02 \x00\x98N\x00\x00The per
son you are looking for is Rabindranath Tagore'

```

Ans: The person you are looking for is Rabindranath Tagore

5. In this question, we filtered all the payload ASCII values of all the TCP packets and then filtered the keyword “milkshake” among them using the “grep” command.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Actual: 4904 packets (1693402 bytes) sent in 13.55 seconds
Rated: 124961.7 Bps, 0.999 Mbps, 361.88 pps
Flows: 296 flows, 21.84 fps, 6860 unique flow packets, 13 unique non-flow packets
Statistics for network device: eth0
Successful packets: 4904
Failed packets: 1969
Truncated packets: 0
Retried packets (ENOBUFS): 0
Retried packets (EAGAIN): 0

(kali@kali) - [~/CN/Assignment 1]
$ sudo python partII.py | grep --color=auto -A 2 "123.134.156.178"
zsh: suspended sudo python partII.py | grep --color=auto -A 2 "123.134.156.178"

(kali@kali) - [~/CN/Assignment 1]
$ sudo python partII.py | grep -A 2 "127.0.0.1"
Source IP:127.0.0.1 Source Port:1111 Destination IP:122.44
.56.78 Destination port:9876 Length:133
..^....c?},...E..w....@.I....z,8N.W&.....P. .$.o..GET
/milkshake HTTP/1.1..Cookie: user:customer..Referer: flavo
r- Strawberry....
Destination Mac Address: F8:63:3F:7D:2C:1A, Source Mac_Add
ress: 88:1D:FC:6C:2D:7F

```

Ans: Flavor-Strawberry

Q3. In this part, we simply ran the same code present in partIII but only for 30-second intervals. After which, a prompt is created which takes in the port number and, in return, provides the PID. In case there is no activity on the port, the prompt returns “No process found using port ___”

```

Destination Mac_Address: 08:00:27:FF:A1:42, Source Mac_Address: 52:54:00:12:35:02
Source IP:54.192.153.239 Source Port:443 Destination IP:10.0.2.15 Destination port:54044 Length:1917
Destination Mac_Address: 52:54:00:12:35:02, Source Mac_Address: 08:00:27:FF:A1:42
Source IP:10.0.2.15 Source Port:54044 Destination IP:54.192.153.239 Destination port:443 Length:54
Destination Mac_Address: 52:54:00:12:35:02, Source Mac_Address: 08:00:27:FF:A1:42
Source IP:10.0.2.15 Source Port:49085 Destination IP:23.200.154.78 Destination port:443 Length:75
Destination Mac_Address: 08:00:27:FF:A1:42, Source Mac_Address: 52:54:00:12:35:02
Source IP:23.200.154.78 Source Port:443 Destination IP:10.0.2.15 Destination port:49085 Length:70
Destination Mac_Address: 52:54:00:12:35:02, Source Mac_Address: 08:00:27:FF:A1:42
Source IP:10.0.2.15 Source Port:54044 Destination IP:54.192.153.239 Destination port:443 Length:85
Destination Mac_Address: 08:00:27:FF:A1:42, Source Mac_Address: 52:54:00:12:35:02
Source IP:54.192.153.239 Source Port:443 Destination IP:10.0.2.15 Destination port:54044 Length:60
Destination Mac_Address: 52:54:00:12:35:02, Source Mac_Address: 08:00:27:FF:A1:42
Source IP:10.0.2.15 Source Port:57510 Destination IP:142.250.192.110 Destination port:443 Length:93

Finished sniffing packets for 30 seconds.

Enter the port number: 443
Process ID for port 443: 29769
Enter the port number: 57510
No process found using port 57510.
Enter the port number: 54044
Process ID for port 54044: 29769
Enter the port number:

```

Q.4>

Q.1> Part-a>

SSDP allows devices such as printers, modems, and surveillance cameras to be discovered on a network quickly and easily. It does this by broadcasting a message to the network, which other devices can respond to.

The RFC for SSDP is RFC 2660. It is titled "Simple Service Discovery Protocol."

22...	49.2085...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
22...	49.2089...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
22...	50.2207...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
22...	50.2215...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
23...	51.2239...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
23...	51.2239...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
25...	52.2338...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
25...	52.2338...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
13...	110.011...	10.7.0.238	239.255.255.250	SSDP	167 M-SEARCH * HTTP/1.1
17...	150.679...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
17...	150.698...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
17...	151.690...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
17...	151.710...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
17...	152.695...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
17...	152.710...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1

ICMP stands for Internet Control Message Protocol. It is a network protocol IP hosts and routers use to send error messages and status information.

The RFC for ICMP is RFC 792. It is titled "Internet Control Message Protocol."

12...	icmpv0	10.7.0.238	142.250.67.238	ICMP	74 Echo (ping) request	id=0x0001, seq=4897/8467, ttl=128 (reply in 12346)
12...	87.6060...	142.250.67.238	10.7.0.238	ICMP	74 Echo (ping) reply	id=0x0001, seq=4897/8467, ttl=115 (request in 12345)
12...	88.5952...	10.7.0.238	142.250.67.238	ICMP	74 Echo (ping) request	id=0x0001, seq=4898/8723, ttl=128 (reply in 12360)
12...	88.6117...	142.250.67.238	10.7.0.238	ICMP	74 Echo (ping) reply	id=0x0001, seq=4898/8723, ttl=115 (request in 12358)
13...	89.6082...	10.7.0.238	142.250.67.238	ICMP	74 Echo (ping) request	id=0x0001, seq=4899/8979, ttl=128 (reply in 13143)
13...	89.6231...	142.250.67.238	10.7.0.238	ICMP	74 Echo (ping) reply	id=0x0001, seq=4899/8979, ttl=115 (request in 13141)
13...	90.6227...	10.7.0.238	142.250.67.238	ICMP	74 Echo (ping) request	id=0x0001, seq=4900/9235, ttl=128 (reply in 13161)
13...	90.6359...	142.250.67.238	10.7.0.238	ICMP	74 Echo (ping) reply	id=0x0001, seq=4900/9235, ttl=115 (request in 13160)

The Address Resolution Protocol (**ARP**) is a network protocol that maps IP addresses to MAC addresses. It is used in local area networks (LANs) to determine the physical address of a device based on its IP address.

RFC 826

316	10.5902...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
319	12.6354...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
52...	63.2242...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
66...	65.9866...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
11...	82.7807...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
14...	116.062...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)
14...	116.164...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)
14...	116.471...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)
15...	127.837...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
16...	128.454...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
16...	128.861...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
16...	129.373...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 0.0.0.0 (Request)
16...	129.783...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)
16...	129.885...	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)

NBNS is a broadcast protocol, which means that it sends messages to all devices on the network.

RFC 1001

17...	151.174...	10.7.0.238	224.0.0.252	LLMNR	64 Standard query 0x434e A wpad
17...	151.580...	10.7.0.238	10.7.63.255	NBNS	92 Name query NB WPAD<00>
17...	151.581...	10.7.0.238	10.7.63.255	NBNS	92 Name query NB WPAD<00>
17...	151.594...	fe80::5f93:9c38:d483:c29d	ff02::1:3	LLMNR	84 Standard query 0xcea8 A wpad
17...	151.594...	fe80::5f93:9c38:d483:c29d	ff02::1:3	LLMNR	84 Standard query 0x434e A wpad
17...	151.594...	10.7.0.238	224.0.0.252	LLMNR	64 Standard query 0x434e A wpad
17...	151.594...	10.7.0.238	224.0.0.252	LLMNR	64 Standard query 0xcea8 A wpad
17...	151.690...	10.7.0.238	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
17...	151.710...	10.7.0.238	239.255.255.250	SSDP	212 M-SEARCH * HTTP/1.1
17...	152.171...	10.7.0.238	224.0.0.251	MDNS	70 Standard query 0x0000 A wpad.local, "QM" question
17...	152.172...	10.7.0.238	224.0.0.251	MDNS	70 Standard query 0x0000 A wpad.local, "QM" question
17...	152.172...	fe80::5f93:9c38:d483:c29d	ff02::fb	MDNS	90 Standard query 0x0000 A wpad.local, "QM" question
17...	152.173...	fe80::5f93:9c38:d483:c29d	ff02::fb	MDNS	90 Standard query 0x0000 A wpad.local, "QM" question
17...	152.337...	10.7.0.238	10.7.63.255	NBNS	92 Name query NB WPAD<00>
17...	152.337...	10.7.0.238	10.7.63.255	NBNS	92 Name query NB WPAD<00>

QUIC stands for Quick UDP Internet Connections. It is a new transport layer protocol designed to improve the performance of web browsing and other internet applications.

RFC 9000

98...	77.6220...	10.7.0.238	35.186.224.25	QUIC	1292 Initial, DCID=e00a86f5887bdc98, PKN: 1, CRYPTO, PING, PING, CRYPTO, CRY
98...	77.6655...	35.186.224.25	10.7.0.238	QUIC	1292 Initial, SCID=e00a86f5887bdc98, PKN: 1, ACK, PADDING
98...	77.7018...	35.186.224.25	10.7.0.238	QUIC	1292 Protected Payload (KP0)
98...	77.7085...	10.7.0.238	35.186.224.25	QUIC	1292 Handshake, DCID=e00a86f5887bdc98
98...	77.7091...	10.7.0.238	35.186.224.25	QUIC	200 Protected Payload (KP0), DCID=e00a86f5887bdc98
98...	77.7100...	10.7.0.238	35.186.224.25	QUIC	1288 Protected Payload (KP0), DCID=e00a86f5887bdc98
98...	77.7101...	10.7.0.238	35.186.224.25	QUIC	706 Protected Payload (KP0), DCID=e00a86f5887bdc98
98...	77.7247...	35.186.224.25	10.7.0.238	QUIC	1292 Protected Payload (KP0)
98...	77.7247...	35.186.224.25	10.7.0.238	QUIC	162 Protected Payload (KP0)
98...	77.7247...	35.186.224.25	10.7.0.238	QUIC	69 Protected Payload (KP0)
98...	77.7250...	10.7.0.238	35.186.224.25	QUIC	74 Protected Payload (KP0), DCID=e00a86f5887bdc98
99...	77.7498...	35.186.224.25	10.7.0.238	QUIC	66 Protected Payload (KP0)
99...	77.7659...	10.7.0.238	35.186.224.25	QUIC	74 Protected Payload (KP0), DCID=e00a86f5887bdc98
99...	77.9299...	35.186.224.25	10.7.0.238	QUIC	309 Protected Pavload (KP0)

Part-b>

Connection = TCP

Time of sending packet = 0.012294405

Time of response = 0.029015437
 RTT = 0.029015437 - 0.012294405
 => 0.016721032 sec = 16.721032 ms

2 0.011340975	10.0.136.7	10.0.2.15	DNS	542 Standard query response 0xc031 A contile-images.services.mozilla.com A 34.120.115.102 NS 1.100
3 0.012294405	10.0.2.15	34.120.115.102	TCP	74 40464 -> 443 [SYN] Seq=0 Win=64000 Len=0 MSS=16015 SACK_PERM TSval=1509736514 TSecr=0 WS=128
4 0.029015437	34.120.115.102	10.0.2.15	TCP	68 443 -> 40464 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460

Q.2>

GitHub: Hypertext Transfer Protocol (HTTP) version 2 (HTTP/2)

Netflix: Hypertext Transfer Protocol (HTTP) version 1.1

Google: Hypertext Transfer Protocol (HTTP) version 2 (HTTP/2)

Difference between HTTP/2 and HTTP/1.1

- HTTP/2 uses a binary framing format for messages, while HTTP/1.1 uses a text-based framing format. This makes HTTP/2 more efficient and reliable.
- HTTP/2 supports multiplexing, which allows multiple requests to be sent over the same connection.
- HTTP/2 uses encryption by default, while HTTP/1.1 does not. This makes HTTP/2 more secure.

Q.3>

The screenshot shows a web browser window with the address bar displaying 'eoffice.iitgn.ac.in/SOComponent/gate.php'. The 'Application' tab is active, and the 'Cookies' section is expanded. The following table represents the cookies listed in the browser:

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Partitioned	Prioritized
AIT	e152d224d12c34ac0c317a152104f7bf	eoffice...	/	2023-...	35	✓	✓			Medium
PHPSESSID	jabr7q6hj3rktslqgo63k56t4	eoffice...	/	Session	35					Medium
_ga_9IPLGQPD3	GS1.1.1690633013.1.1.1690633350.40.0.0	.iitgn.a...	/	2024-...	52					Medium
_fbp	fb2.1690633015007.821817296	.iitgn.a...	/	2023-...	32			Lax		Medium
_ga	GA1.1.502845086.1690633014	.iitgn.a...	/	2024-...	29					Medium

Below the table, there is a message: 'Select a cookie to preview its value'.

- **_ga**: This cookie is used by Google Analytics to track your visits to the website. It is a persistent cookie, meaning it expires after two years.
- **PHPSESSID**: The PHPSESSID cookie is a first-party cookie, which means it is set by the website you visit. It is a session cookie that expires when you close your browser.
- **_fbp**: It tracks users across different websites and serves them with targeted advertising. The _fbp cookie is persistent, meaning it expires after three months.

References:-

- https://www.uv.mx/personal/angelperez/files/2018/10/sniffers_texto.pdf
- <https://github.com/jshreyans/sniffer>