Assignment 1: Getting To Know Network Traffic

COL334

Laksh Goel (2023CS10848)

IIT Delhi

August 21, 2025

Contents

1 Measurement Tools **2**
- 1.1 Ping . . . 2
  - 1.1.1 For Google . . . 4
  - 1.1.2 For Cragslist . . . 6
- 1.2 Traceroute . . . 6

2 Traffic Capture and Analysis **10**
- 2.1 HTTP Site . . . 10
  - 2.1.1 DNS Analysis . . . 10
  - 2.1.2 HTTP Requests . . . 11
  - 2.1.3 TCP Connections . . . 13
- 2.2 HTTPS Site . . . 13
  - 2.2.1 Any HTTP packet? . . . 14
  - 2.2.2 HTML or JavaScript file without filters? . . . 14
  - 2.2.3 DNS traffic: . . . 14
  - 2.2.4 TCP Connections: . . . 14
  - 2.2.5 Explanation: . . . 15

3 Performance Analysis **15**
- 3.1 Python Script . . . 15
- 3.2 Analysis of packet captures . . . 15

4 Conclusion **19**

Goal

The goal of this assignment is to familiarize with network data collection, traffic analysis, and basic network measurement tools. We use `ping`, `traceroute`, and `Wireshark`, and also analyze packet captures using a Python script.

1 Measurement Tools

1.1 Ping

Screenshots:

```
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 10 www.google.com
PING www.google.com (142.250.182.164): 56 data bytes
64 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=26.698 ms
64 bytes from 142.250.182.164: icmp_seq=1 ttl=116 time=30.310 ms
64 bytes from 142.250.182.164: icmp_seq=2 ttl=116 time=32.189 ms
64 bytes from 142.250.182.164: icmp_seq=3 ttl=116 time=27.266 ms
64 bytes from 142.250.182.164: icmp_seq=4 ttl=116 time=26.770 ms
64 bytes from 142.250.182.164: icmp_seq=5 ttl=116 time=27.649 ms
64 bytes from 142.250.182.164: icmp_seq=6 ttl=116 time=30.472 ms
64 bytes from 142.250.182.164: icmp_seq=7 ttl=116 time=30.783 ms
64 bytes from 142.250.182.164: icmp_seq=8 ttl=116 time=30.639 ms
64 bytes from 142.250.182.164: icmp_seq=9 ttl=116 time=26.712 ms

--- www.google.com ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 26.698/28.949/32.189/2.006 ms
lakshgoel@Lakshs-MacBook-Air ~ %
```

Figure 1: ping to www.google.com

```
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 10 www.craigslist.com
PING www.craigslist.com (208.82.237.135): 56 data bytes
64 bytes from 208.82.237.135: icmp_seq=0 ttl=43 time=409.291 ms
64 bytes from 208.82.237.135: icmp_seq=1 ttl=43 time=327.844 ms
64 bytes from 208.82.237.135: icmp_seq=2 ttl=43 time=300.328 ms
64 bytes from 208.82.237.135: icmp_seq=3 ttl=43 time=368.537 ms
64 bytes from 208.82.237.135: icmp_seq=4 ttl=43 time=266.661 ms
64 bytes from 208.82.237.135: icmp_seq=5 ttl=43 time=308.598 ms
64 bytes from 208.82.237.135: icmp_seq=6 ttl=43 time=327.513 ms
64 bytes from 208.82.237.135: icmp_seq=7 ttl=43 time=262.074 ms
64 bytes from 208.82.237.135: icmp_seq=8 ttl=43 time=262.208 ms
64 bytes from 208.82.237.135: icmp_seq=9 ttl=43 time=283.892 ms

--- www.craigslist.com ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 262.074/311.695/409.291/45.954 ms
lakshgoel@Lakshs-MacBook-Air ~ %
```

Figure 2: ping to craigslist.com

A. Protocols Used ICMP (Internet Control Message Protocol) is used by ping to test connectivity between two hosts. System sends an ICMP Echo Request message to the target host. If the target host is reachable, it replies with an ICMP Echo Reply message. By measuring the time between sending the request and receiving the reply, ping calculates the round-trip time (RTT) and checks packet loss.

ICMP is part of the network layer.

B. Latency Comparison

$$\text{Craigslist (www.craigslist.com):} \quad \text{Average ping} \approx 312\,\text{ms}$$
$$\text{Google (www.google.com):} \quad \text{Average ping} \approx 29\,\text{ms}$$

- This difference mainly arises from the server infrastructure and geographical distribution of the two websites. Google operates a vast network of globally distributed data centers, with servers located close to most users, including in India. As a result, packets sent to Google travel shorter distances and pass through fewer intermediate networks, reducing delay.
- In contrast, Craigslist has fewer servers, primarily concentrated in the United States. This means packets from the network must travel across longer distances and multiple hops to reach the Craigslist servers, leading to a significantly higher average round-trip time.
- Even for the same website, the latency across multiple pings is not constant. Variations occur due to several factors. Network congestion can cause queuing delays at routers, while routing paths may change dynamically depending on traffic conditions, slightly altering the round-trip time.

C. IPv6 Ping To force ping to use IPv6, I used the ping6 command (equivalent to ping -6) on macOS.

```
lakshgoel@Lakshs-MacBook-Air ~ % ping6 -c 5 www.google.com
PING6(56=40+8+8 bytes) 2001:df4:e000:3fc1:9434:ae1c:1288:d77d --> 2404:6800:4002:815::2004
16 bytes from 2404:6800:4002:815::2004, icmp_seq=0 hlim=116 time=6.051 ms
16 bytes from 2404:6800:4002:815::2004, icmp_seq=1 hlim=116 time=10.459 ms
16 bytes from 2404:6800:4002:815::2004, icmp_seq=2 hlim=116 time=10.209 ms
16 bytes from 2404:6800:4002:815::2004, icmp_seq=3 hlim=116 time=6.491 ms
16 bytes from 2404:6800:4002:815::2004, icmp_seq=4 hlim=116 time=5.969 ms

--- www.google.com ping6 statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 5.969/7.836/10.459/2.049 ms
```

Figure 3: ping6 to google.com

- For Google, IPv6 ping runs successfully. The DNS for Google resolves to an IPv6 address (2404:6800:4002:815::2004 in this case), and the replies are received with very low round-trip times (around 6–10 ms). This shows that Google has full IPv6 support and nearby servers capable of responding to IPv6 traffic.

- In contrast, when we try ping6 -c 5 www.craigslist.com, the command fails with the error "nodename nor servname provided, or not known". This happens because Craigslist does not publish an IPv6 address in its DNS. Without an IPv6 address, the system cannot establish an IPv6 connection, and therefore no pings are possible over IPv6. Craigslist can only be reached using IPv4 in this case.

```
round-trip min/avg/max/std-dev = 5.969/7.836/10.459/2.049 ms
lakshgoel@Lakshs-MacBook-Air ~ % ping6 -c 5 www.craigslist.com
ping6: getaddrinfo -- nodename nor servname provided, or not known
lakshgoel@Lakshs-MacBook-Air ~ %
```

Figure 4: ping6 to craigslist.com

D. Maximum Packet Size

To determine the maximum size that can be transmitted successfully, I performed a binary search approach.

1.1.1 For Google

```
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 56 www.google.com
PING www.google.com (142.250.182.164): 56 data bytes
64 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=27.430 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 27.430/27.430/27.430/0.000 ms
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 128 www.google.com
PING www.google.com (142.250.182.164): 128 data bytes
136 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=31.487 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 31.487/31.487/31.487/nan ms
lakshgoel@Lakshs-MacBook-Air ~ %
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 256 www.google.com
PING www.google.com (142.250.182.164): 256 data bytes
264 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=26.530 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 26.530/26.530/26.530/nan ms
lakshgoel@Lakshs-MacBook-Air ~ %
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 512 www.google.com
PING www.google.com (142.250.182.164): 512 data bytes
520 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=27.078 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 27.078/27.078/27.078/0.000 ms
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1024 www.google.com
PING www.google.com (142.250.182.164): 1024 data bytes
1032 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=39.337 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 39.337/39.337/39.337/nan ms
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 2048 www.google.com
PING www.google.com (142.250.182.164): 2048 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
lakshgoel@Lakshs-MacBook-Air ~ %
```

Figure 5: Finding maximum packet size for google.com till 1024 bytes

```
lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1536 www.google.com
PING www.google.com (142.250.182.164): 1536 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
lakshgoel@Lakshs-MacBook-Air ~ %
```

Figure 6: Packet Loss at 1536 Bytes

```
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1280 www.google.com
PING www.google.com (142.250.182.164): 1280 data bytes
1288 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=35.023 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 35.023/35.023/35.023/0.000 ms
[lakshgoel@Lakshs-MacBook-Air ~ %
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1408 www.google.com
PING www.google.com (142.250.182.164): 1408 data bytes
1416 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=31.247 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 31.247/31.247/31.247/nan ms
[lakshgoel@Lakshs-MacBook-Air ~ %
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1472 www.google.com
PING www.google.com (142.250.182.164): 1472 data bytes
1480 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=35.933 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 35.933/35.933/35.933/0.000 ms
```

Figure 7: No packet loss till 1472 Bytes

```
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1504 www.google.com
PING www.google.com (142.250.182.164): 1504 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1488 www.google.com
PING www.google.com (142.250.182.164): 1488 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1480 www.google.com
PING www.google.com (142.250.182.164): 1480 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1476 www.google.com
PING www.google.com (142.250.182.164): 1476 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1474 www.google.com
PING www.google.com (142.250.182.164): 1474 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1473 www.google.com
PING www.google.com (142.250.182.164): 1473 data bytes

--- www.google.com ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
[lakshgoel@Lakshs-MacBook-Air ~ % ping -c 1 -s 1472 www.google.com
PING www.google.com (142.250.182.164): 1472 data bytes
1480 bytes from 142.250.182.164: icmp_seq=0 ttl=116 time=26.757 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 26.757/26.757/26.757/0.000 ms
```

Figure 8: Packet loss from 1504 to 1473 Bytes, No packet loss at 1472 Bytes

Packets up to 1472 bytes consistently reached the server with zero packet loss, while any larger size (from 1473 bytes onward) resulted in 100% packet loss. Thus, for Google, the largest ICMP packet that can be reliably sent without loss is 1472 bytes—a limitation likely set by the network's maximum transmission unit along the path.

Including the ICMP header (8 bytes) and the IP header (20 bytes), the total maximum packet size was 1500 bytes, which matches the standard Ethernet Maximum Transmission Unit.

1.1.2 For Cragslist

On doing similar analysis on Cragslist, I found that packets of size greater than 283 Bytes size had 100% packet loss. So the maximum total packet size is 311 bytes (283 + 28).

This indicates that Craigslist's network path enforces a much smaller effective MTU or explicitly restricts ICMP packet sizes. Such restrictions are common where servers, often located overseas—may be configured to block larger ICMP packets to reduce load, prevent fragmentation, and mitigate potential security risks.

1.2 Traceroute

Screenshots:

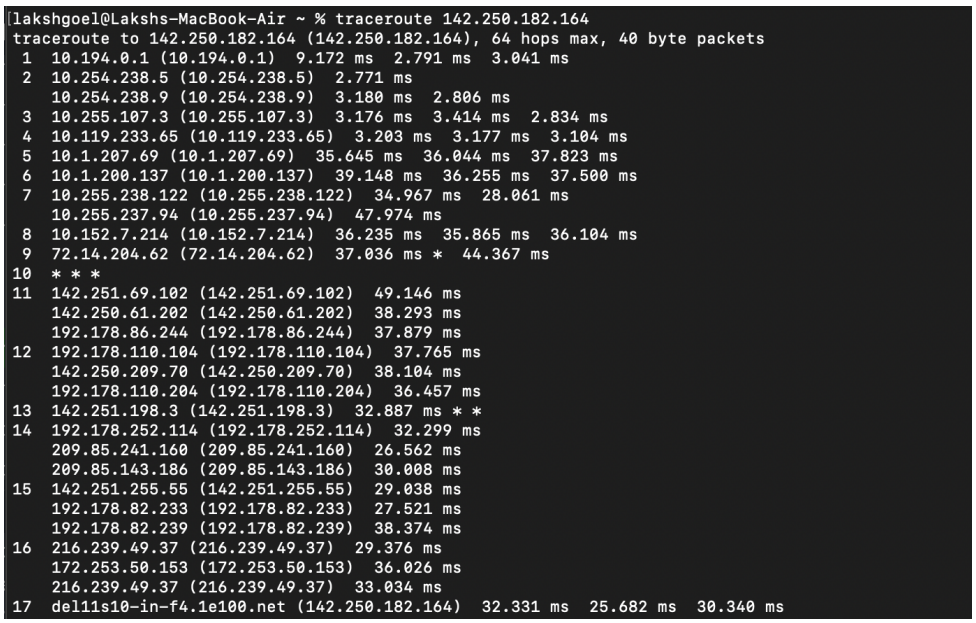

Figure 9: Google Traceroute

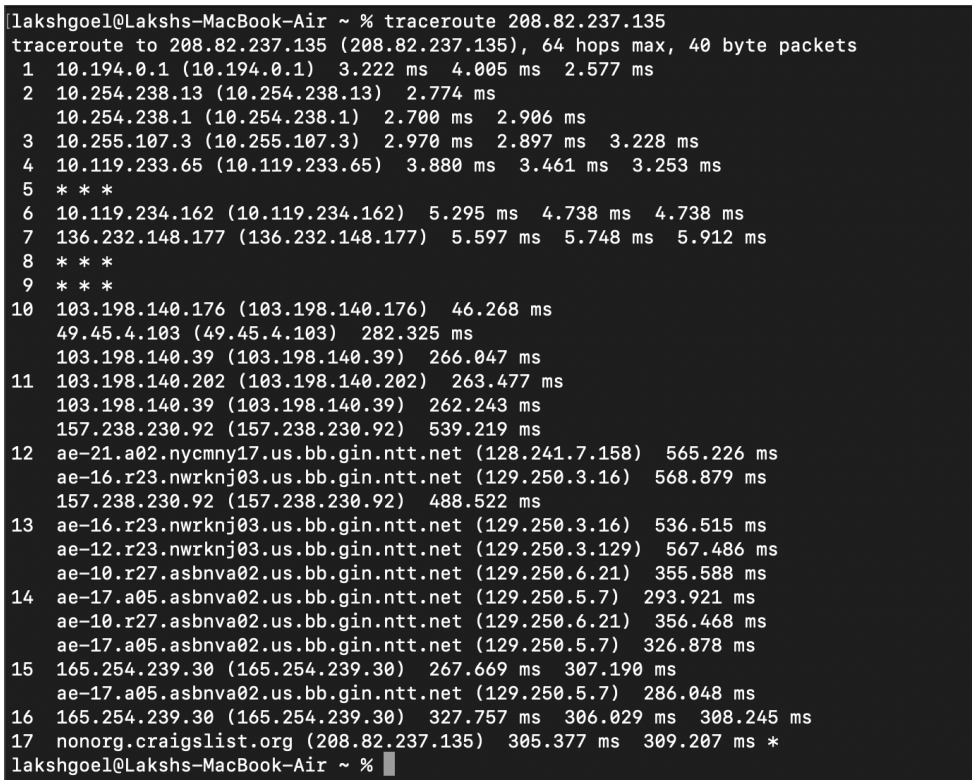

Figure 10: Criageslist traceroute

A. Hop Counts and ASNs - Google: 17 hops (142.250.182.164) - Craigslist: 17 hops (208.82.237.135)
Website used to find ASN number: iplocation.com

Traceroute to Google (142.250.182.164)

Table 1: Per-hop Details (Google)

Hop	Responder (IP/Hostname)	ASN	Location
1	10.194.0.1	Private IP	Not Available
2	10.254.238.5	Private IP	Not Available
3	10.254.238.9	Private IP	Not Available
4	10.119.233.65	Private IP	Not Available
5	10.1.207.69	Private IP	Not Available
6	10.1.200.137	Private IP	Not Available
7	10.255.238.122	Private IP	Not Available
8	10.255.237.94	Private IP	Not Available
9	10.152.7.214	Private IP	Not Available
10	72.14.204.62	AS15169	Mumbai, India
11	* * *	-	-
12	142.251.69.102	AS15169	US, North America
	142.250.61.202	AS15169	US, North America
	192.178.86.244	AS15169	US, North America

Continued on next page

Table 1 – continued from previous page

Hop	Responder (IP/Hostname)	ASN	Location
13	192.178.110.104	AS15169	US, North America
	142.250.209.70	AS15169	US, North America
	192.178.110.204	AS15169	US, North America
14	142.251.198.3	AS15169	US, North America
	192.178.252.114	AS15169	US, North America
	209.85.241.160	AS15169	US, North America
	209.85.143.186	AS15169	US, North America
15	142.251.255.55	AS15169	US, North America
	192.178.82.233	AS15169	US, North America
	192.178.82.239	AS15169	US, North America
16	216.239.49.37	AS15169	Georgia, US, North America
	172.253.50.153	AS15169	US, North America
	216.239.49.37	AS15169	Georgia, US, North America
17	del11s10-in-f4.1e100.net (142.250.182.164)	AS15169	US, North America

Traceroute to Craigslist (208.82.237.135)

Table 2: Per-hop Details (Craigslist)

Hop	Responder (IP/Hostname)	ASN	Location
1	10.194.0.1	Private IP	Not Available
2	10.254.238.13	Private IP	Not Available
	10.254.238.1	Private IP	Not Available
3	10.255.107.3	Private IP	Not Available
4	10.119.233.65	Private IP	Not Available
5	* * *	-	-
6	10.119.234.162	Private IP	Not Available
7	136.232.148.177	AS55836	Delhi, India
8	* * *	-	-
9	* * *	-	-
10	103.198.140.176	AS64049	Singapore, Asia
	49.45.4.103	AS64049	Mumbai, Maharashtra
	103.198.140.39	AS64049	Singapore, Asia
11	103.198.140.202	AS64049	Singapore
	103.198.140.39	AS64049	Singapore
	157.238.230.92	AS2914	Malayshia, Asia
12	ae-21.a02.nycmny17.us.bb.gin.ntt.net (128.241.7.158)	AS2914	Culver City, California, USA
	ae-16.r23.nwrknj03.us.bb.gin.ntt.net (129.250.3.16)	AS2914	New York, USA
	157.238.230.92	AS2914	New York, USA
13	ae-16.r23.nwrknj03.us.bb.gin.ntt.net (129.250.3.16)	AS2914	New York, USA
	ae-12.r23.nwrknj03.us.bb.gin.ntt.net (129.250.3.129)	AS2914	New York, USA

Continued on next page

Table 2 – continued from previous page

Hop	Responder (IP/Hostname)	ASN	Location
	ae-10.r27.asbnva02.us.bb.gin.ntt.net (129.250.6.21)	AS2914	New York, USA
14	ae-17.a05.asbnva02.us.bb.gin.ntt.net (129.250.5.7)	AS2914	New York, USA
	ae-10.r27.asbnva02.us.bb.gin.ntt.net (129.250.6.21)	AS2914	New York, USA
	ae-17.a05.asbnva02.us.bb.gin.ntt.net (129.250.5.7)	AS2914	New York, USA
15	165.254.239.30	AS2914	Los Angeles, California, USA
	ae-17.a05.asbnva02.us.bb.gin.ntt.net (129.250.5.7)	AS2914	New York, USA
16	165.254.239.30	AS2914	Los Angeles, California, USA
17	nonorg.craigslist.org (208.82.237.135)	AS22414	San Francisco, California, USA

Any IPs in range

- 10.0.0.0 – 10.255.255.255
- 172.16.0.0 – 172.31.255.255
- 192.168.0.0 – 192.168.255.255

are internal/private IPs, not assigned to any organization on piblic internet.

B. Meaning of '*': Yes, the "*" in the traceroute output indicates that the probe packet sent at that hop did not receive a reply within the timeout period. This could happen due to packet loss, network congestion, or because the router at that hop is configured to block or deprioritize ICMP Time Exceeded messages. It does not necessarily mean the path is broken, only that the specific hop chose not to respond.

C. Multiple IPs at Same Hop Yes, in some cases you may observe multiple IP addresses for the same hop count in a traceroute output. This usually happens because routers often use load balancing or equal-cost multi-path (ECMP) routing. When multiple paths exist that are considered equally good by the routing protocol, the router may distribute packets across these paths to balance network traffic. As a result, each probe that traceroute sends can take a slightly different route, leading to multiple IP addresses being reported at the same hop.

What is load balancing?

Load balancing or equal-cost multi-path (ECMP) routing means that if a router has two or more equally good routes to reach the destination, it doesn't always pick just one. Instead, it splits the traffic across these routes to use the network more efficiently and avoid congestion.

D. Geolocation of IPs

For the Google traceroute, the initial hops are private IP addresses within the local network, showing very low RTTs (2–9 ms). Hop 10 (72.14.204.62) is geolocated to Mumbai, India, and the latency remains low at around 37 ms, suggesting a nearby server. Although subsequent hops are listed in the US, the final RTT to the destination server in Delhi is only about 30 ms. This indicates that the geolocation data for the intermediate hops might be misleading, and the connection was likely routed through Google's infrastructure within India, which is consistent with the very low overall latency.

For the Craigslist traceroute,

- Local Hops (1-7): The first few hops are local private IPs and an ISP in Delhi (AS55836), with RTTs under 6 ms.
- Jump to Asia (Hop 10): The RTT jumps significantly to 46-282 ms as traffic is routed to Singapore and Mumbai (AS64049).

- Intercontinental Jump (Hops 11-12): There's a massive increase in latency, soaring to over 500 ms. This corresponds with the geolocation data showing the path moving from Asia (Malaysia, AS2914) to the United States (New York, AS2914). This jump is the primary contributor to the high total RTT and logically represents the transoceanic delay.
- Within the US (Hops 12-17): The RTTs remain high but relatively stable (around 300 ms) as the packets traverse the US from New York to the final destination in San Francisco.

E. Internet Structure

The path to Craigslist clearly illustrates the traditional 3-tiered Internet architecture:

1. Access Providers: Hops 1-6 show private IP addresses like 10.194.0.1 and 10.254.238.13. These are internal to my local network. At hop 7, your data leaves your local network and hits the first public internet provider in Delhi (AS55836).
2. Transit Providers: The traffic then moves through several regional and international transit providers. It goes through AS64049 (in Singapore/Mumbai) and then to AS2914, which carries the traffic from Asia to the United States.
3. Destination ISP: Finally, the traffic arrives at AS22414, the network hosting Craigslist in San Francisco.

The path to Google does not follow this observable tiered structure. After leaving the local network, the traffic enters Google's own massive global network (AS15169) at hop 10 in Mumbai. From that point on, every subsequent hop until the final destination is within AS15169.

This is happening because Google operates a vast **Content Delivery Network (CDN)**. Instead of paying transit providers to carry its traffic across the globe, Google has built its own infrastructure. It peers directly with access ISPs worldwide, allowing it to manage traffic routing internally. This keeps user traffic on its own high-speed, optimized network for as long as possible, which is why the connection is much faster and doesn't show hops across multiple different Autonomous Systems.

2 Traffic Capture and Analysis

2.1 HTTP Site

2.1.1 DNS Analysis

Website: http://www.httpvshttps.com

I used a Disply *dns.qry.name == "www.httpvshttps.com"* filter in Wireshark to filter out DNS queries and responses of the site. Then I used time reference feature in wireshark to set a refernce to a query packet.

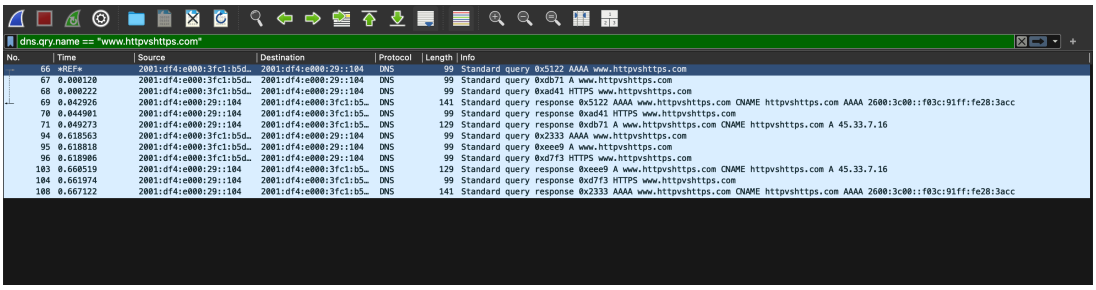

Figure 11: Wireshark capture showing multiple DNS requests.

Result:

- Query (Packets 66 & 69): $0.042926s - 0.000000s = 0.042926s = 42.926ms$
- Query (Packets 67 & 71): $0.049273s - 0.000220s = 0.049053s = 49.053ms$
- Query (Packets 68 & 70): $0.044301s - 0.000222s = 0.044079s = 44.079ms$
- Query (Packets 94 & 106): $0.667122s - 0.618563s = 0.048559s = 48.559ms$
- Query (Packets 95 & 103): $0.660519s - 0.618818s = 0.041701s = 41.701ms$
- Query (Packets 96 & 104): $0.661974s - 0.618986s = 0.042988s = 42.988ms$

The average time for a DNS response across these six requests is:

$$\frac{42.926 + 49.053 + 44.079 + 48.559 + 41.701 + 42.988}{6} = \frac{269.306}{6} \approx 44.884ms$$

2.1.2 HTTP Requests

I applied a *http* filter to filter out http requests. Number of packets are shown at bottom right of wireshark.

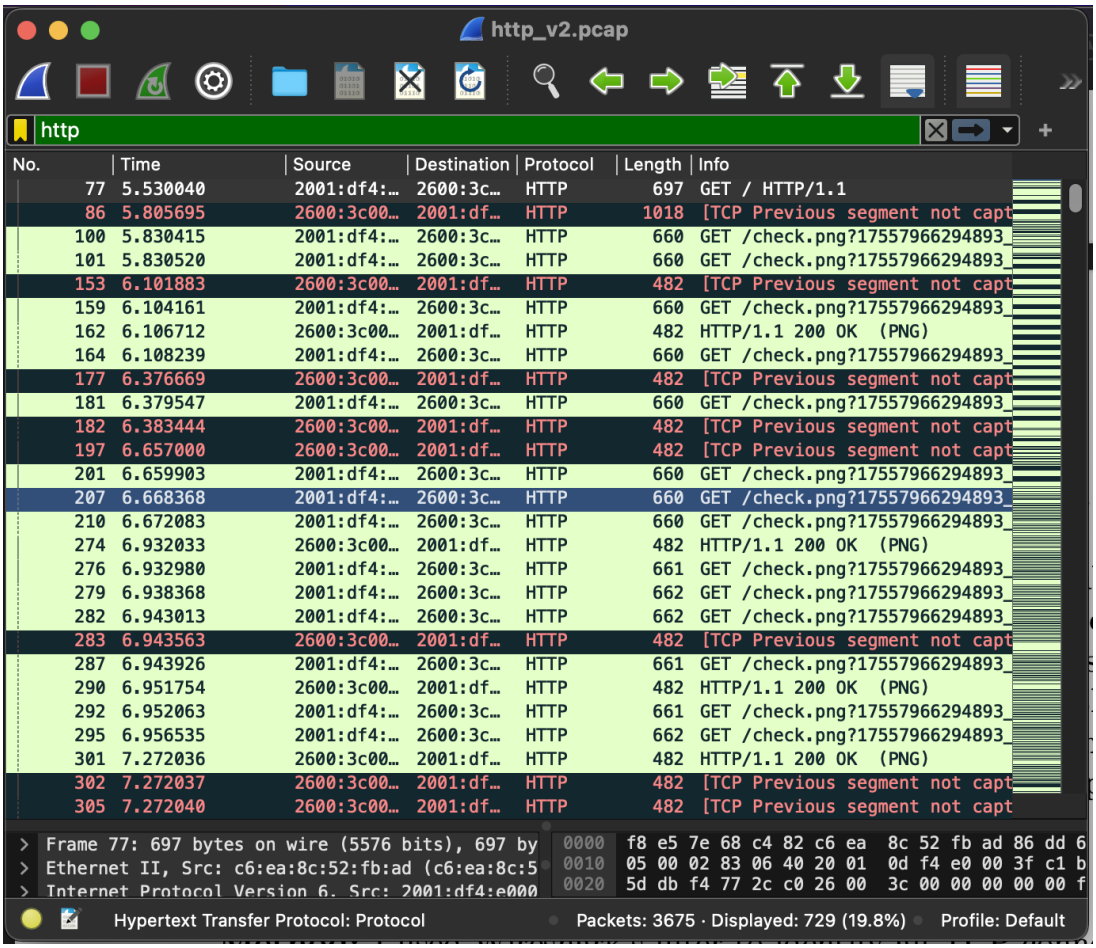

Figure 12: Wireshark capture showing HTTP requests.

Result: Approximately 729 HTTP requests.
Analysis:

1. The capture shows approximately 729 HTTP requests were made for loading a single webpage. This indicates that the website is composed of numerous individual resources such as images, scripts, and style sheets. A high request count generally increases page load time due to repeated request–response cycles.

2. Multiple requests are initiated before the corresponding responses are received. This reflects how browsers attempt to optimize performance by opening parallel connections

3. The initial GET request fetches the HTML document, which in turn references other embedded objects. The subsequent requests for images, JavaScript files, and style sheets show that the website follows a dependency chain: the browser must first parse the HTML before issuing additional resource requests. This hierarchical loading increases latency.

4. Observed that each request takes around $\sim$0.3s

5. Each object (PNG, SVG, JavaScript file) is retrieved through an independent HTTP request.

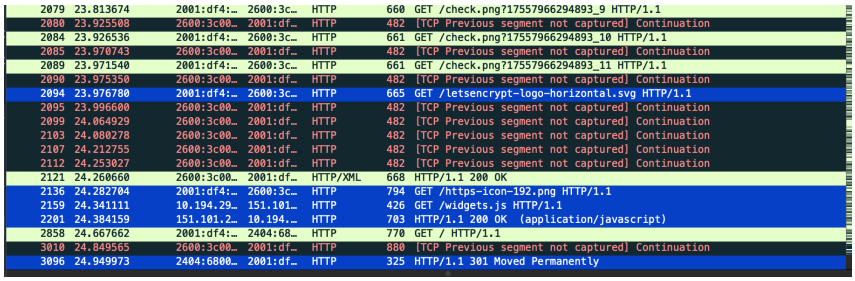

Figure 13: Multiple objects fetched for one page.

2.1.3 TCP Connections

I filtered out packets that marked the start of new TCP connection using filter:
`tcp.flags.syn==1 && tcp.flags.ack==0 && (ip.addr == 45.33.7.16 or ipv6.addr == 2600:3c00::f03c:91ff:fe28:3acc)`
Count: 9

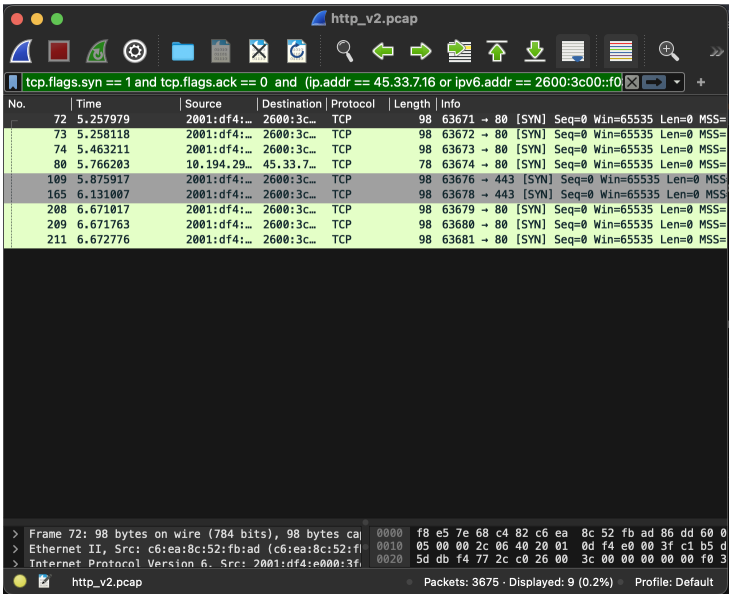

Figure 14: Count of TCP packets

A. TCP vs HTTP connection: Number of TCP connections are significantly less than HTTP connections. With HTTP/1.1, the default behavior is to keep a single TCP connection open and reuse it for multiple HTTP requests/responses.

B. Content objects fetched over the same TCP connection: Yes, Multiple content objects fetched over same TCP Connection.

2.2 HTTPS Site

https://www.httpvshttps.com

I performed a packet capture on above mentioned site.

2.2.1 Any HTTP packet?

I applied an *http* filter. There was no http traffic visible. This is expected because HTTPS does not transmit HTTP data in plaintext. Instead, HTTP requests and responses are encrypted inside TLS records, making them unreadable to packet capture tools

2.2.2 HTML or JavaScript file without filters?

On inspecting the traffic without filters, no HTML or JavaScript files were visible. Instead, the capture showed only encrypted packets of TLSv1.2 and TLSv1.3. These packets encapsulate the website's resources (HTML, CSS, JavaScript, images) but in an encrypted format. This demonstrates the core advantage of HTTPS: confidentiality of application-layer content.

2.2.3 DNS traffic:

I used a Disply *dns.qry.name == "www.httpvshttps.com"* filter in Wireshark to filter out DNS queries and responses of the site.

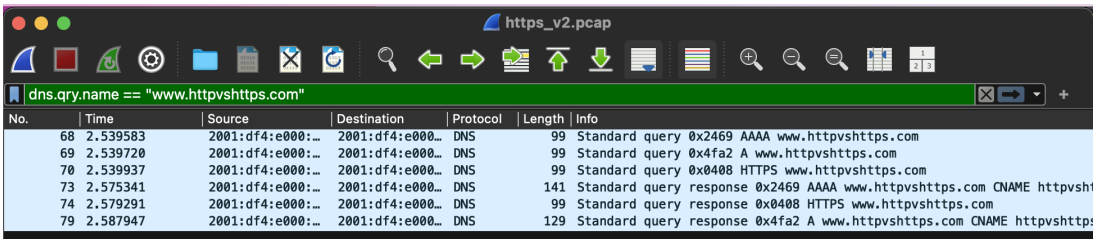

Figure 15: Wireshark capture showing multiple DNS requests for HTTPS.

Result: Yes, There was DNS traffic. 3 query packets were observed, as shown in figure. Unlike HTTP content, DNS queries (in this case) are transmitted in plaintext.

2.2.4 TCP Connections:

I filtered out packets that marked the start of new TCP connection using filter:
`tcp.flags.syn==1 && tcp.flags.ack==0 && (ip.addr == 45.33.7.16 or ipv6.addr == 2600:3c00::f03c:91ff:fe28:3acc)`
Count: 4

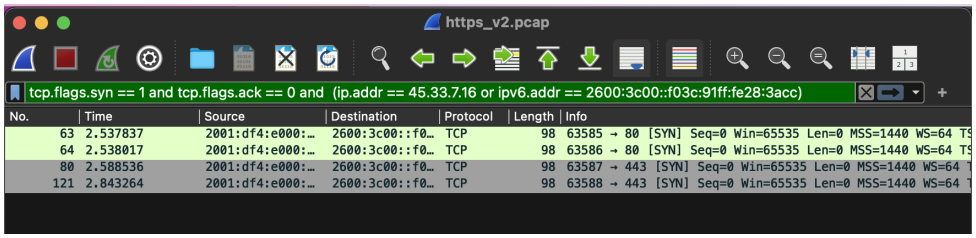

Figure 16: Count of TCP packets for HTTPS

In HTTP connection, there were 9 TCP connections. These numbers were expected to be similar, as HTTPS just adds a layer of encryption on top of underlying TCP transport protocol.

The relatively small number of connections might be due to the fact that duration of capture of data in HTTPS is quite small. HTTPS generally also uses persistent connections.

2.2.5 Explanation:

The key difference between HTTP and HTTPS traffic lies in the use of TLS encryption. When a browser connects via HTTPS, the communication first goes through a TLS handshake where cryptographic parameters are negotiated, certificates are validated, and session keys are established. After this handshake, all application-layer data (HTTP requests, responses, and web objects) are encrypted.

Because of this encryption:

1. No HTTP packets are visible — the payload is hidden inside TLS records.
2. HTML/JS files are not directly observable — only encrypted application data flows can be seen.
3. Fewer TCP connections are required — because persistent TLS sessions can carry many requests.

In contrast, with plain HTTP, every request and response was visible in Wireshark, and many TCP connections were required to fetch all resources. The differences arise because HTTPS is designed to provide confidentiality, integrity, and efficiency, while HTTP only focuses on functional delivery without security.

3 Performance Analysis

3.1 Python Script

Methodology: No file was mentioned in the command given in the assignment document, so I wrote a script that could capture traffic between client and server IP addresses. I used *tshark* for capturing data and saving it in a file. Later, a Piazza thread clarified that a PCAP file would be provided for analysis. I have used the *dpkt* library for packet analysis and *matplotlib* for visualization. I iterated over all the packets and filtered out IP packets containing TCP data.

1. Throughput Analysis: For throughput analysis at discrete unit time intervals, I created a dictionary mapping discrete time to data transferred or received in each interval, and plotted it.

2. Round Trip Time (RTT) Analysis: For Round Trip Time calculation, I matched the acknowledgement (A) with sent packets such that $A = S + L$, where S is the sequence number and L is the size of the packet. I then stored each timestamp with its RTT value and plotted them.

3.2 Analysis of packet captures

Download Throughput:

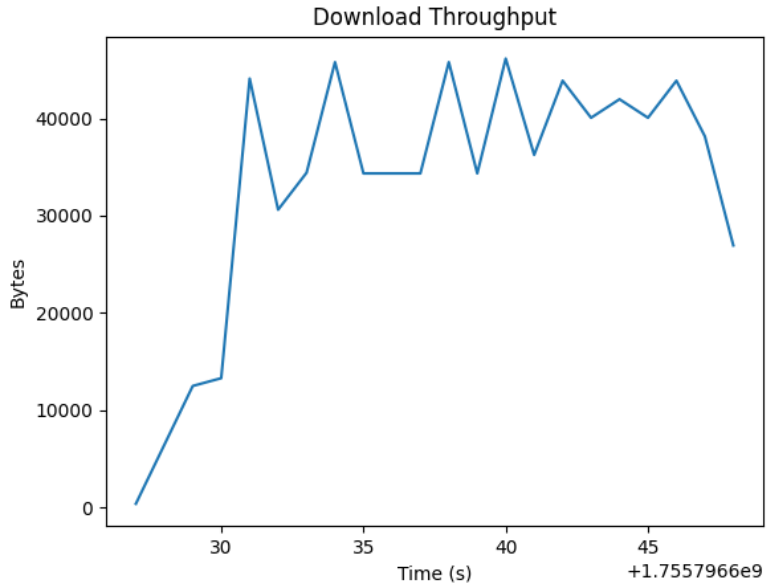

Figure 17: Download Throughput plot for HTTP

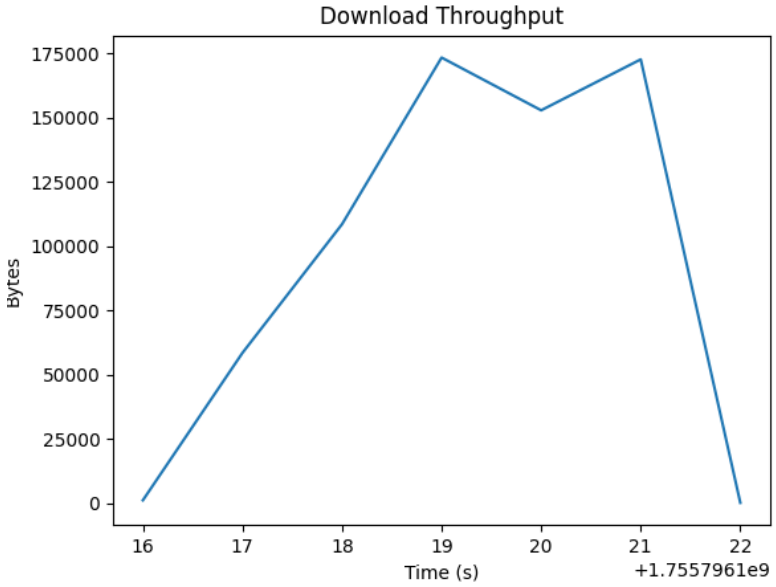

Figure 18: Download Throughput plot for HTTPS

Upload Throughput:

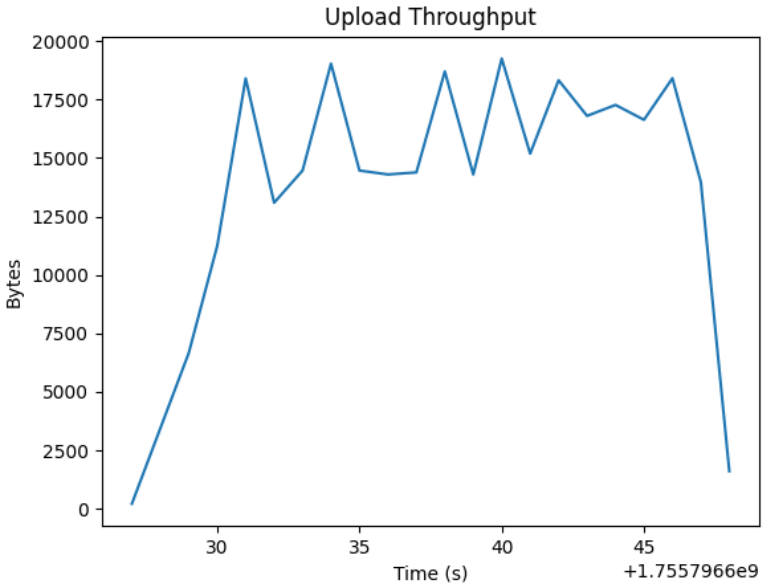

Figure 19: Upload Throughput plot for HTTP

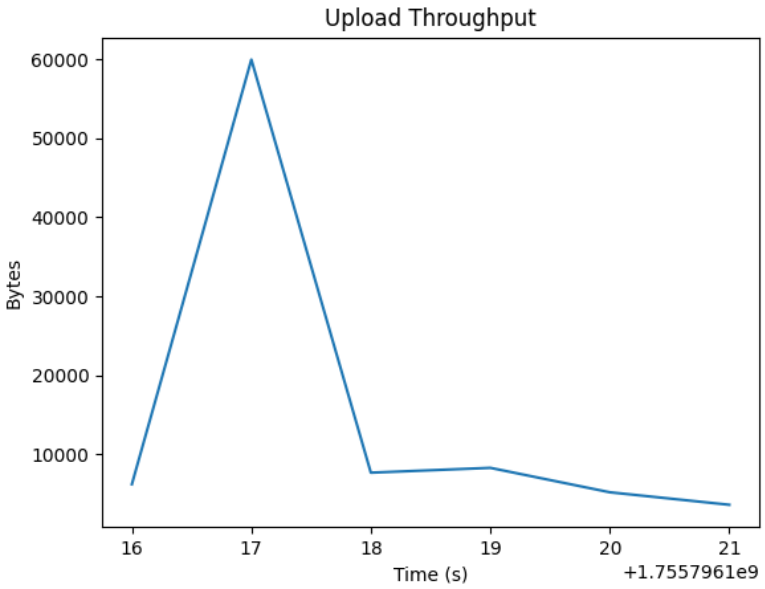

Figure 20: Upload Throughput plot for HTTPS

Round Trip Time (RTT):

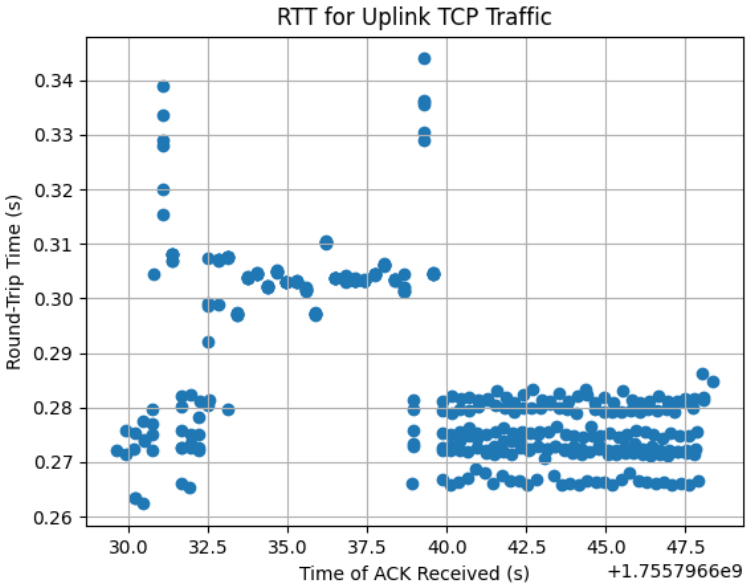

Figure 21: RTT plot for HTTP

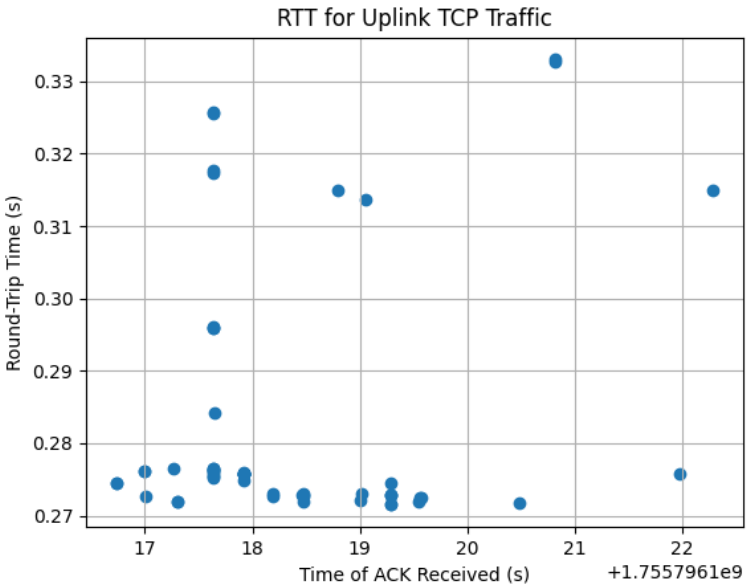

Figure 22: RTT plot for HTTPS

Analysis:

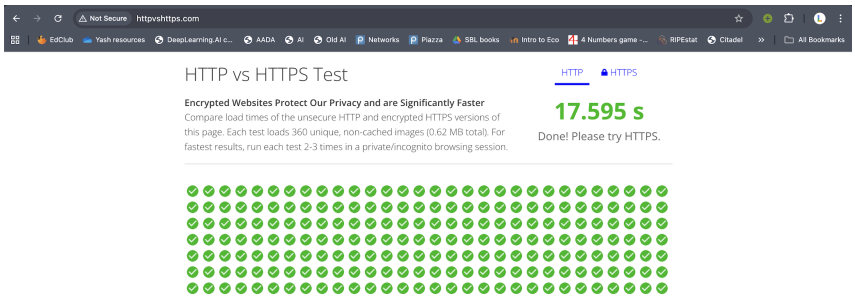

Figure 23

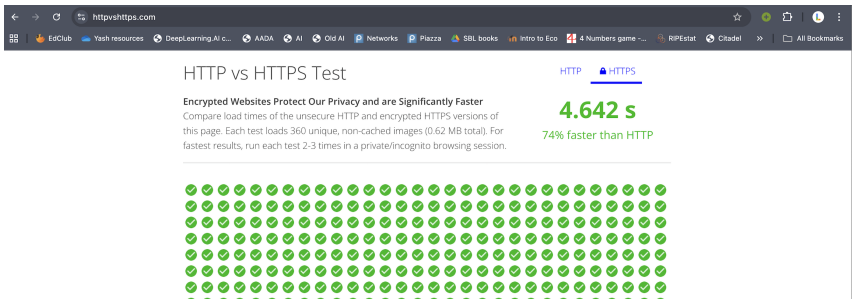

Figure 24

From the packet captures, it is observed that the total time taken to load the HTTP site was significantly higher than that for the HTTPS site. The primary reason for this lies in the difference in download throughput.

The download throughput for HTTPS was much higher compared to HTTP. A higher throughput implies that a larger amount of data could be transferred per unit time, thereby reducing the overall page load time. In contrast, HTTP had a much lower throughput, leading to slower transfers and consequently longer completion time.

$$\text{Download throughput in HTTP: } 44\ Megabytes$$
$$\text{Time displayed on HTTP webpage: } 17.6\ Seconds$$
$$\text{Download throughput in HTTPS: } 17.5\ Megabytes$$
$$\text{Time displayed on HTTPS webpage: } 4.6\ Seconds$$

Compairing the plots with the time shown on webpage,

- In figure 17, Duration of positive throughtput is 48-29 = 19s, which is nearly equal to 17.6s, shown on webpage.
- In figure 18, Duration of positive throughtput is 21-16 = 5s, which is nearly equal to 4.6s, shown on webpage.

4 Conclusion

This assignment provided a comprehensive analysis of network traffic and performance using fundamental measurement tools. The investigation revealed key insights into how network architecture,

protocols, and security measures impact the end-user experience.

The use of ping and traceroute demonstrated the significant real-world performance advantage of modern, distributed network architectures. The low latency connection to Google's Content Delivery Network (CDN) was a stark contrast to the high latency path to Craigslist's centralized servers, which involved a transcontinental route. Furthermore, the analysis highlighted differences in modern protocol adoption, with Google supporting IPv6 seamlessly while Craigslist did not.

Traffic capture and analysis with Wireshark provided a granular view of web protocols. Accessing the HTTP site revealed the high multiplicity of requests—over 700—needed to render a single webpage, managed through a smaller number of persistent TCP connections. The HTTPS capture concretely demonstrated the primary benefit of TLS: all application-layer data, including HTML and JavaScript files, was fully encrypted and uninspectable.

Finally, the performance analysis conducted with a Python script quantitatively confirmed that the HTTPS connection was not only more secure but also significantly more efficient. The HTTPS site achieved a substantially higher download throughput, resulting in a page load time nearly four times faster (4.6 seconds) than its unencrypted HTTP counterpart (17.6 seconds).