Assignment 3

- Name: Abhiroop Mukherjee
- Roll No.: 510519109
- GSuite: 510519109.abhirup@students.iiests.ac.in

- Subject: Computer Networks Lab (CS 3272)

Question 1

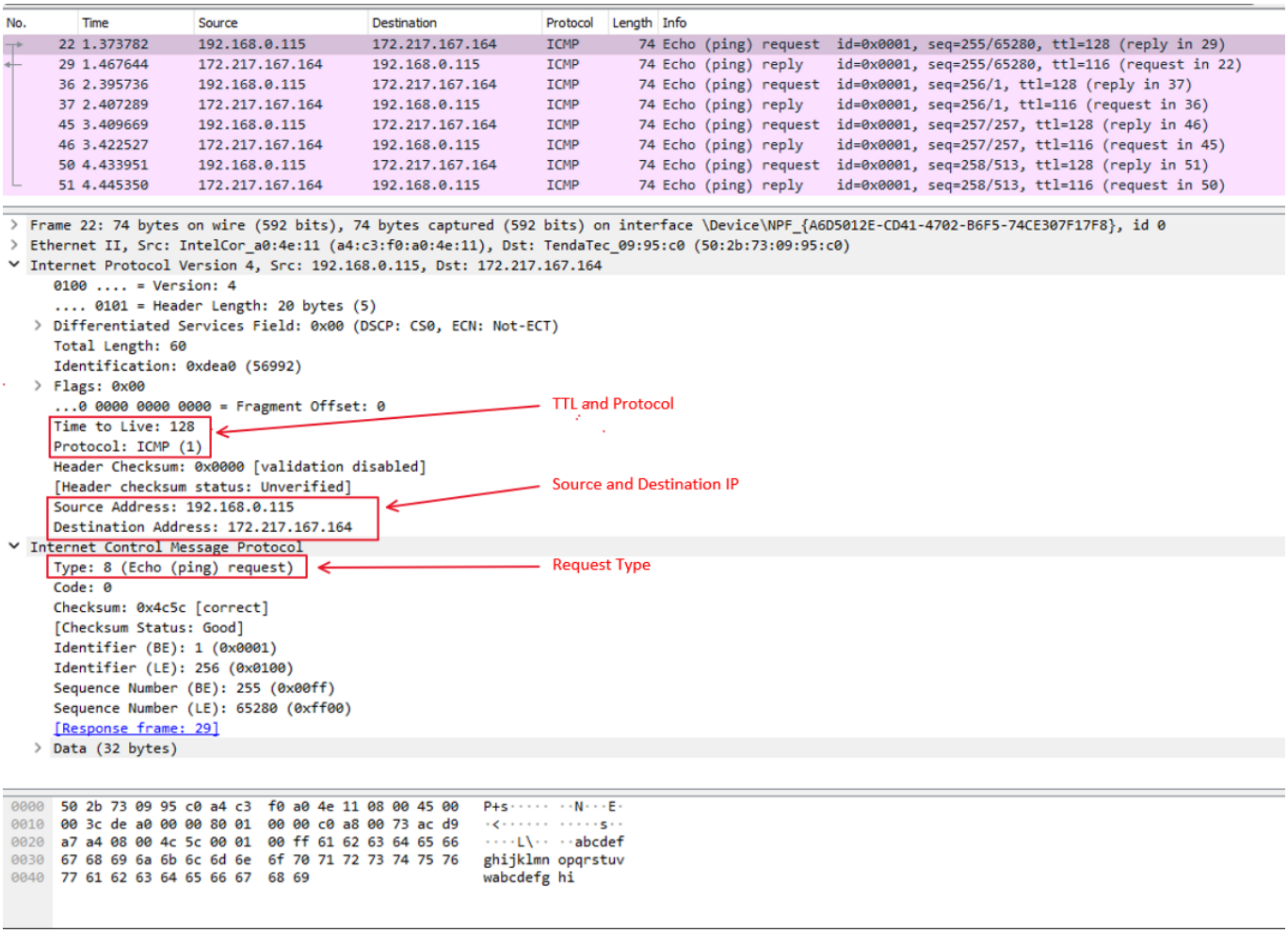
Analyse the packets (across all layers) exchanged with your computer while executing the following commands:

1. ping
2. traceroute
3. dig
4. arp
5. wget

1. ping

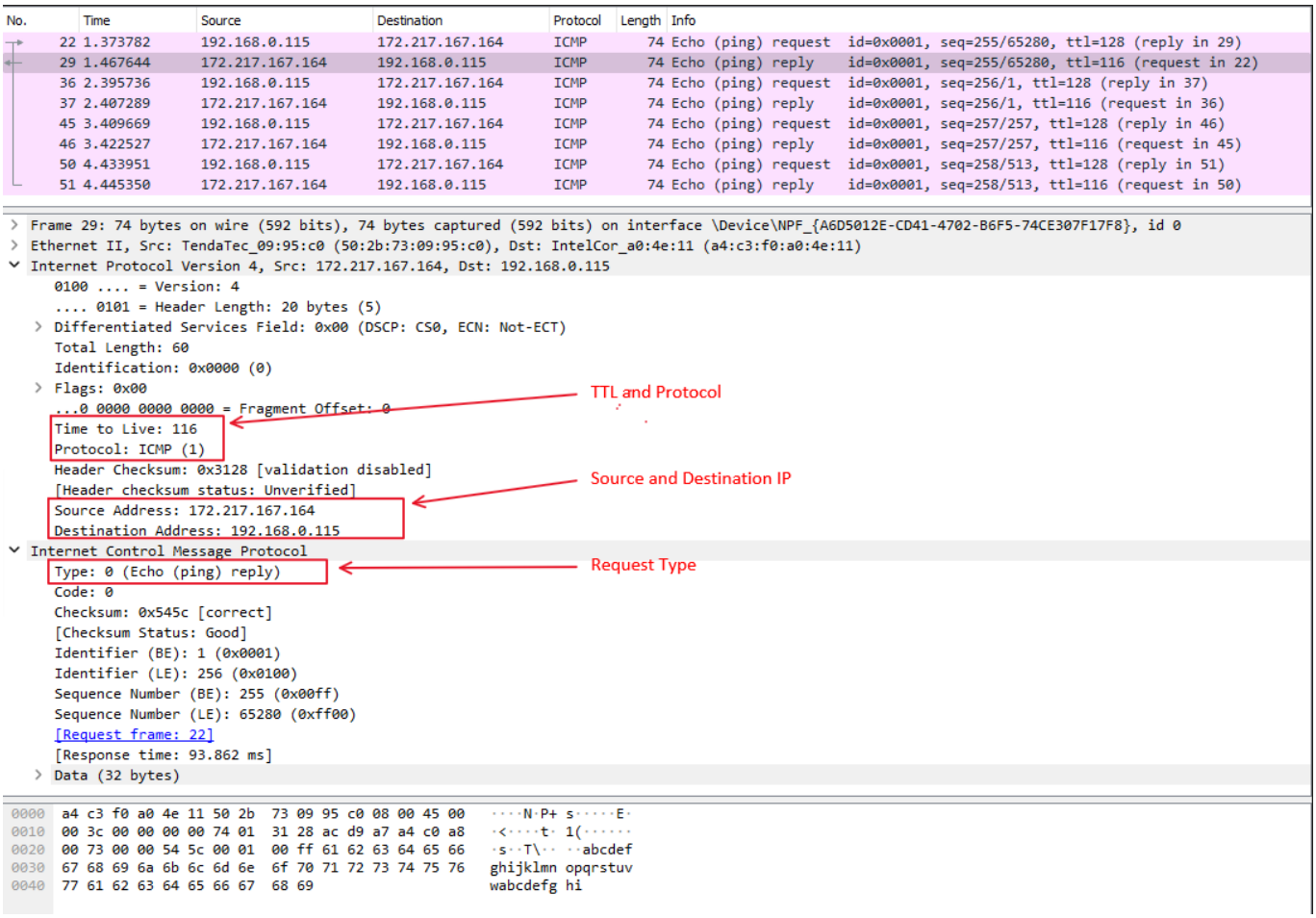
`ping` uses ICMP (Internet Control Message Protocol) to send messages to a host. Running `ping www.google.com` in terminal and then capturing the packets via wireshark, we observe the following:

Echo Request Packet



- Protocol: ICMP
- Type: 8 (Echo (ping) request)
- TTL: 128

Echo Reply Packet



- Protocol: ICMP
- Type: 0 (Echo (ping) reply)
- TTL: 116

Protocol Info

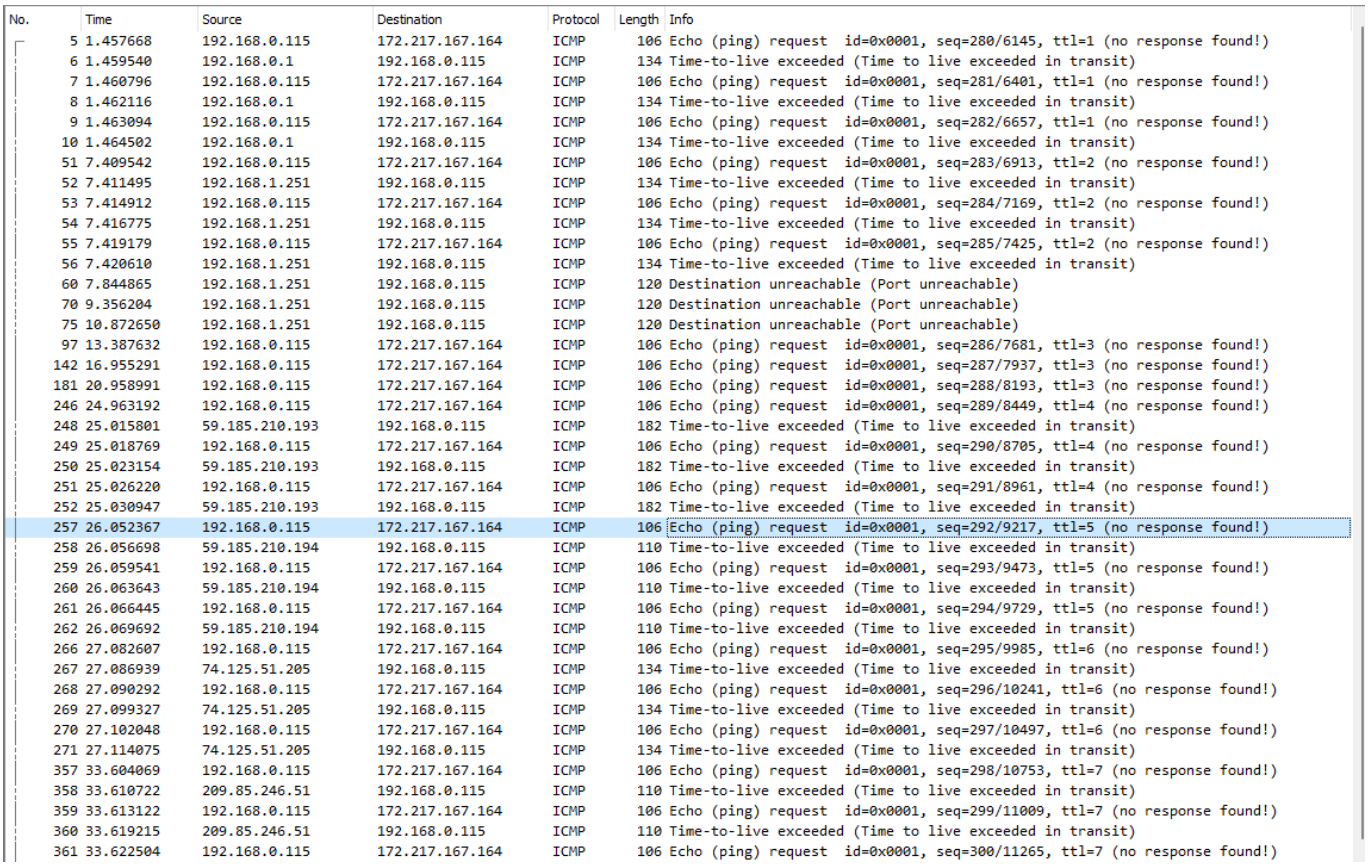
Layer	Protocol Used
Network	ICMP

NOTE: ICMP is not associated with a transport layer protocol like TCP or UDP.

2. traceroute

The way `traceroute` works is by sending packets with increasing TTL and rely on "TTL Exceeded" packets from the routers to trace the path to a host.

Running `traceroute -I www.google.com` in terminal and then capturing the packets via wireshark, we observe the following:



No.	Time	Source	Destination	Protocol	Length	Info
5	1.457668	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=280/6145, ttl=1 (no response found!)
6	1.459540	192.168.0.1	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
7	1.460796	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=281/6401, ttl=1 (no response found!)
8	1.462116	192.168.0.1	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
9	1.463094	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=282/6657, ttl=1 (no response found!)
10	1.464502	192.168.0.1	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
51	7.409542	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=283/6913, ttl=2 (no response found!)
52	7.411495	192.168.1.251	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
53	7.414912	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=284/7169, ttl=2 (no response found!)
54	7.416775	192.168.1.251	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
55	7.419179	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=285/7425, ttl=2 (no response found!)
56	7.420610	192.168.1.251	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
60	7.844865	192.168.1.251	192.168.0.115	ICMP	120	Destination unreachable (Port unreachable)
70	9.356204	192.168.1.251	192.168.0.115	ICMP	120	Destination unreachable (Port unreachable)
75	10.872650	192.168.1.251	192.168.0.115	ICMP	120	Destination unreachable (Port unreachable)
97	13.387632	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=286/7681, ttl=3 (no response found!)
142	16.955291	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=287/7937, ttl=3 (no response found!)
181	20.958991	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=288/8193, ttl=3 (no response found!)
246	24.963192	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=289/8449, ttl=4 (no response found!)
248	25.015801	59.185.210.193	192.168.0.115	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
249	25.018769	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=290/8705, ttl=4 (no response found!)
250	25.023154	59.185.210.193	192.168.0.115	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
251	25.026220	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=291/8961, ttl=4 (no response found!)
252	25.030947	59.185.210.193	192.168.0.115	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
257	26.052367	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=292/9217, ttl=5 (no response found!)
258	26.056698	59.185.210.194	192.168.0.115	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
259	26.059541	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=293/9473, ttl=5 (no response found!)
260	26.063643	59.185.210.194	192.168.0.115	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
261	26.066445	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=294/9729, ttl=5 (no response found!)
262	26.069692	59.185.210.194	192.168.0.115	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
266	27.082607	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=295/9985, ttl=6 (no response found!)
267	27.086939	74.125.51.205	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
268	27.090292	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=296/10241, ttl=6 (no response found!)
269	27.099327	74.125.51.205	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
270	27.102048	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=297/10497, ttl=6 (no response found!)
271	27.114075	74.125.51.205	192.168.0.115	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
357	33.604069	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=298/10753, ttl=7 (no response found!)
358	33.610722	209.85.246.51	192.168.0.115	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
359	33.613122	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=299/11009, ttl=7 (no response found!)
360	33.619215	209.85.246.51	192.168.0.115	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
361	33.622504	192.168.0.115	172.217.167.164	ICMP	106	Echo (ping) request id=0x0001, seq=300/11265, ttl=7 (no response found!)

Here we can observe that my host is sending ICMP echo request packets with increasing TTL and the routers are responding with "TTL Exceeded" packets.

We also see that all packets sent from my IP (192.168.0.115) have destination IP address (the IP address found by DNS lookup of www.google.com).

Protocol Info

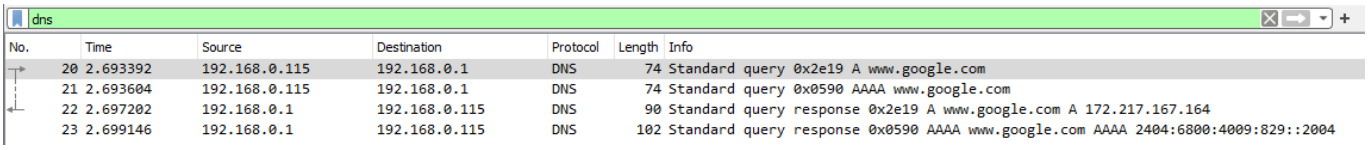
Since we used the `-I` flag, the protocol used is ICMP.

Layer	Protocol Used
Network	ICMP

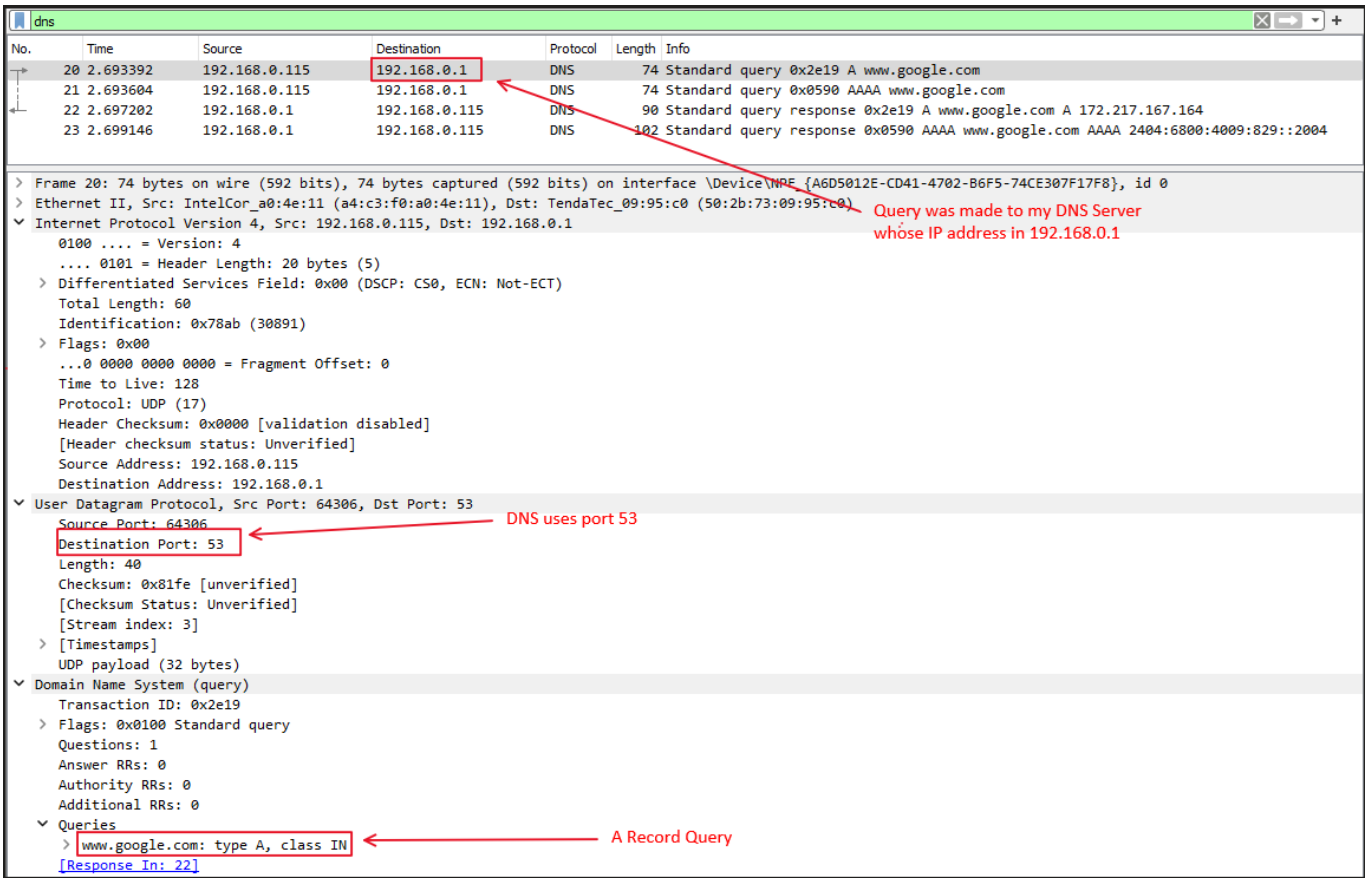
3. dig

We know that `dig` uses DNS protocol, so we can add "dns" filter in wireshark.

Running `dig www.google.com` in terminal and then capturing the packets via wireshark, we observe the following:

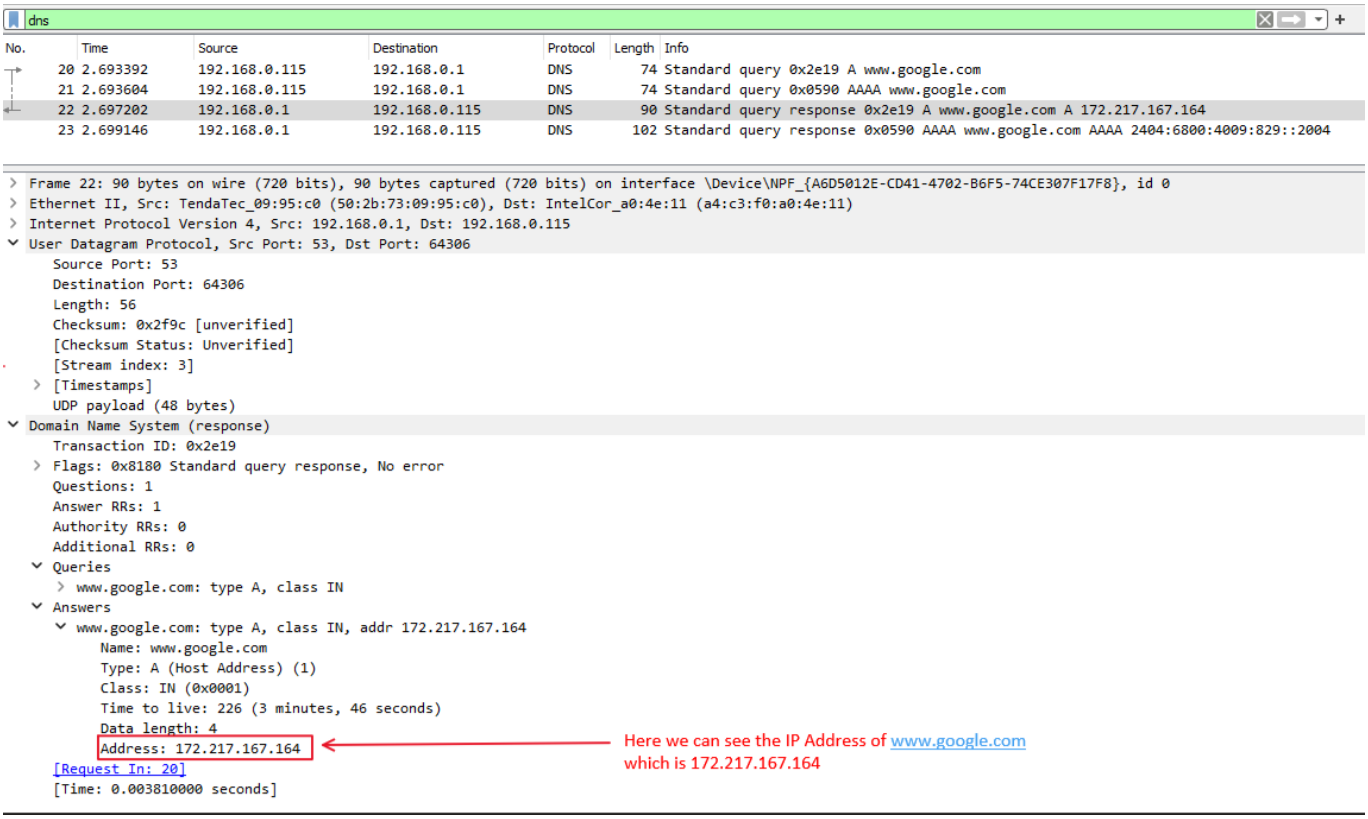


DNS request:



- Query made from DNS Server 192.168.0.1
- Query: "www.google.com: type A, class IN"

DNS response:



- We received an A record for `www.google.com` with address `172.217.167.164`

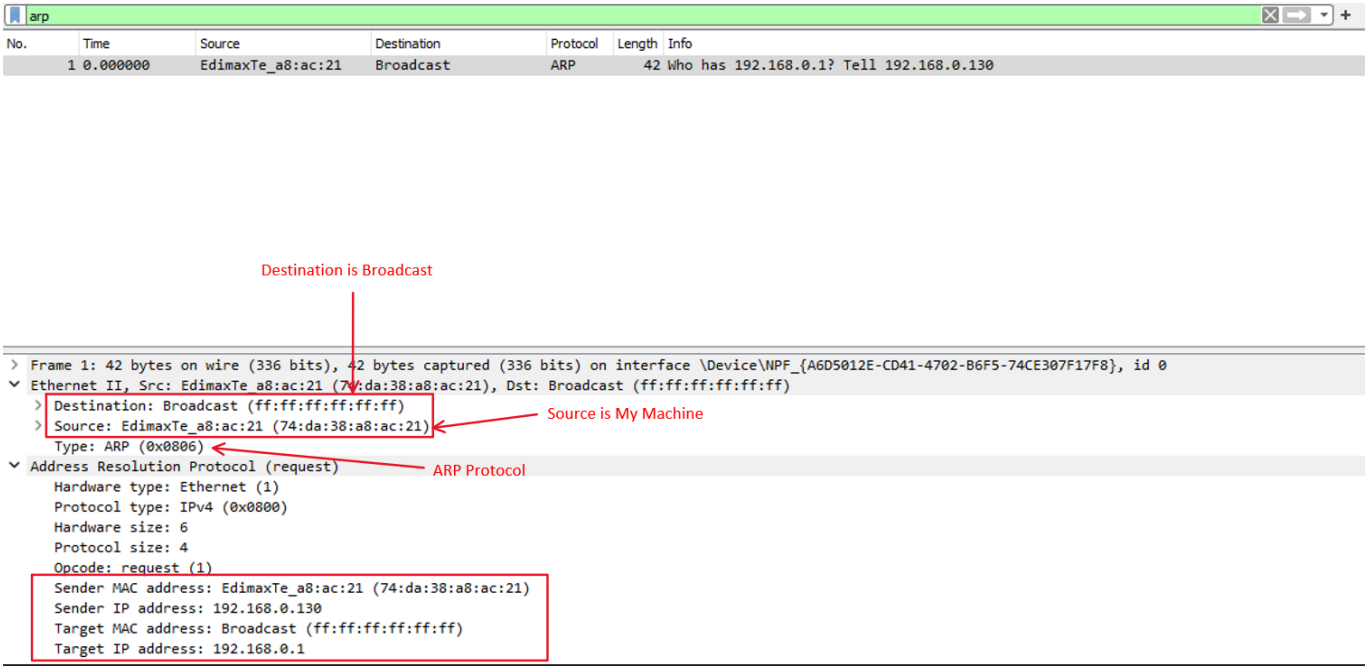
Protocol Info

- Protocol used here is DNS

Layer	Protocol Used
Application	DNS
Transport	UDP
Network	IP

4. arp

`arp` maintains a table of mac addresses and corresponding IP addresses. So when we run `arp` command, it sends a broadcast packet to all the hosts in the network.



- Destination: Broadcast (`ff:ff:ff:ff:ff:ff`)

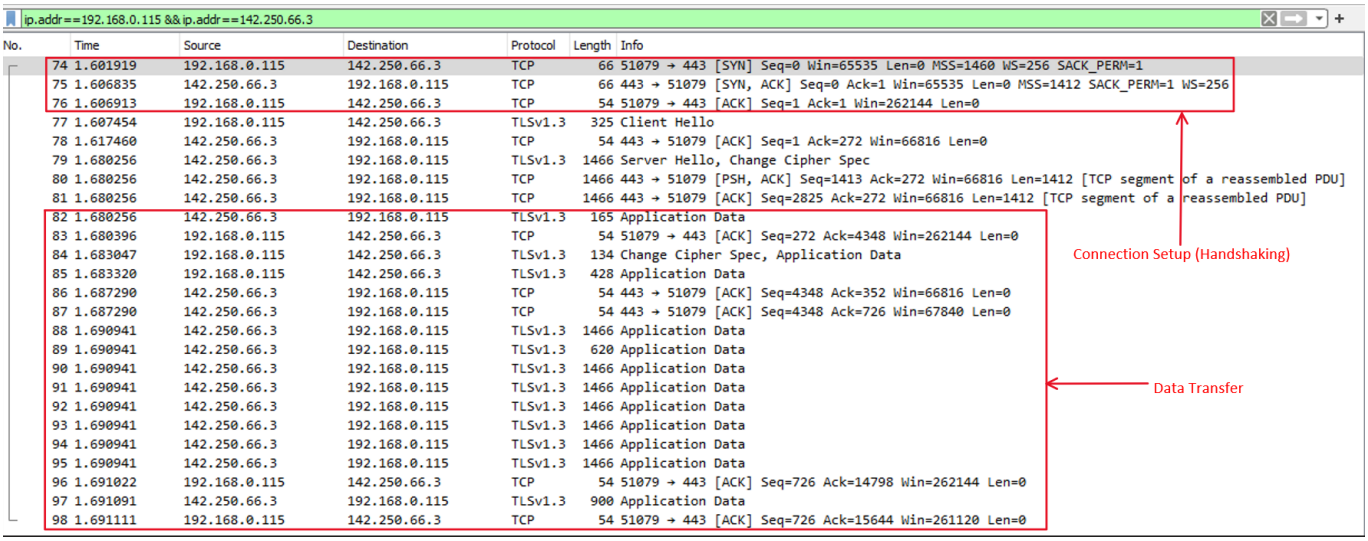
Protocol Info

- Protocol: ARP

Layer	Protocol Used
Network	ARP

5. wget

`wget` makes a GET HTTP(S) request to a server and downloads the file. Running `wget www.google.com` in terminal and then capturing the packets via wireshark, we observe the following:



No.	Time	Source	Destination	Protocol	Length	Info
74	1.601919	192.168.0.115	142.250.66.3	TCP	66	51079 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
75	1.606835	142.250.66.3	192.168.0.115	TCP	66	443 → 51079 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM=1 WS=256
76	1.606913	192.168.0.115	142.250.66.3	TCP	54	51079 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
77	1.607454	192.168.0.115	142.250.66.3	TLSv1.3	325	Client Hello
78	1.617460	142.250.66.3	192.168.0.115	TCP	54	443 → 51079 [ACK] Seq=1 Ack=272 Win=66816 Len=0
79	1.680256	142.250.66.3	192.168.0.115	TLSv1.3	1466	Server Hello, Change Cipher Spec
80	1.680256	142.250.66.3	192.168.0.115	TCP	1466	443 → 51079 [PSH, ACK] Seq=1413 Ack=272 Win=66816 Len=1412 [TCP segment of a reassembled PDU]
81	1.680256	142.250.66.3	192.168.0.115	TCP	1466	443 → 51079 [ACK] Seq=2825 Ack=272 Win=66816 Len=1412 [TCP segment of a reassembled PDU]
82	1.680256	142.250.66.3	192.168.0.115	TLSv1.3	165	Application Data
83	1.680396	192.168.0.115	142.250.66.3	TCP	54	51079 → 443 [ACK] Seq=272 Ack=4348 Win=262144 Len=0
84	1.683047	192.168.0.115	142.250.66.3	TLSv1.3	134	Change Cipher Spec, Application Data
85	1.683320	192.168.0.115	142.250.66.3	TLSv1.3	428	Application Data
86	1.687290	142.250.66.3	192.168.0.115	TCP	54	443 → 51079 [ACK] Seq=4348 Ack=352 Win=66816 Len=0
87	1.687290	142.250.66.3	192.168.0.115	TCP	54	443 → 51079 [ACK] Seq=4348 Ack=726 Win=67840 Len=0
88	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
89	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	620	Application Data
90	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
91	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
92	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
93	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
94	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
95	1.690941	142.250.66.3	192.168.0.115	TLSv1.3	1466	Application Data
96	1.691022	192.168.0.115	142.250.66.3	TCP	54	51079 → 443 [ACK] Seq=726 Ack=14798 Win=262144 Len=0
97	1.691091	142.250.66.3	192.168.0.115	TLSv1.3	900	Application Data
98	1.691111	192.168.0.115	142.250.66.3	TCP	54	51079 → 443 [ACK] Seq=726 Ack=15644 Win=261120 Len=0

- First Packet was sent with `SYN` flag set
- Second Packet was sent with `SYN-ACK` flag set
- Finally, Third Packet was sent with `ACK` flag set
- Now the handshaking is done, data transfer starts

Looking at a packet of Data Transfer we see:

```
····N·P+s ··· E ···   z··)··B··· s···· ·k·4·4IP·· ··  ·········S·y·=Xf·····
·1···ae·$n· VHnd}·1···LH·Zh·||i·c···<·8·@·· ··s·········· ·····@!··|·*·}b····@}
lH·;····x·····@+·M ·`t·%|O*·4·_··S-·····3···········:p·V···,c k·4·[·········
·^ZY·5G·····>c··_·\···2k ·^G·····~··g····E· ··FCP·C····F···~Z v···`8·^·i·4·3KY·Z
··N···^ZF···Y*a·-G··m····*··· ·f·····i·m8C····2·····)·m··8···*····%·wD··0·0·$g·
·T·h·[··D·y··a···ho·····3@#·5·o··q··Y·····rI···nY···n·P··.·=·d·z····/
··········A·@·····n&··ny··d·8·J*·S·····IU·~·r····"\···\z····Y··Ds····I·0··J,·····O6··
· *6 ·o ·· ··*6··xjqE)···;·G%···f`@l····U··t·j·*·····A·i·>·_v){·G+·f·Fc·····7··a··|·,
5····&·]··*B·x··0 ··vG·d·w$· ····/·<·f··5·····8/·····)4ot····}xW···!
K[·ck·3>···(·*···I"··c··8·]··UM··dC·?··n·········B·#·t·· j··/···d··)···|·· ··Y·3···
·v·x· ·j9·L)R·q[w···$·-·X·/x·
···*·vca·;··Q·C·a···en9.w···\·%··gD·:·····8···········?·h·G····
····Q··r·i·g···z;·lD···q··l·············{·u·····Mf··q···|··n····h··o····B&
H··g·····,·.\·f·M·<+···v][··z>8O·K1···9···=g··Z·7af
"··J···)···H····9····]··9·····%)··>·····Dl···f·
Yq8···$?··+···K v\6·@·8·y··)·;9··1·(···YD···7·z4~··l···}··<ko·/
`··6··Yq····p··Zt&D····0YO08·g··&····:v··o·K·····?·8·· ···Y·a····J·C··J1JT/
·B·_···x··6·Ovb··*{j·B··Xt··]··%·Y·i>·· ··>>f··
·;·2x··smKI·+.···6·(·[··H··T[··<·p··········NG?z|
q·y··t·········g·····."·u····So·y·····W···i}····· ·K···G=·········&··g···gY··v|
·3··>··+lBO)5H··h···f·5·····_··· ·F·o.A·U····q···Z···"···)t[···C·····m·····WC·4
```

- here all data is gibberish, because it was end-to-end encrypted. The request was made through HTTPS protocol which makes it impossible for sniffers like wireshark to decode the data.

Protocol Info

- Application Layer Protocol: HTTPS
- Transport Layer Protocol: TCP
- Network Layer Protocol: IP

Layer	Protocol Used
Application	HTTPS
Transport	TCP
Network	IP

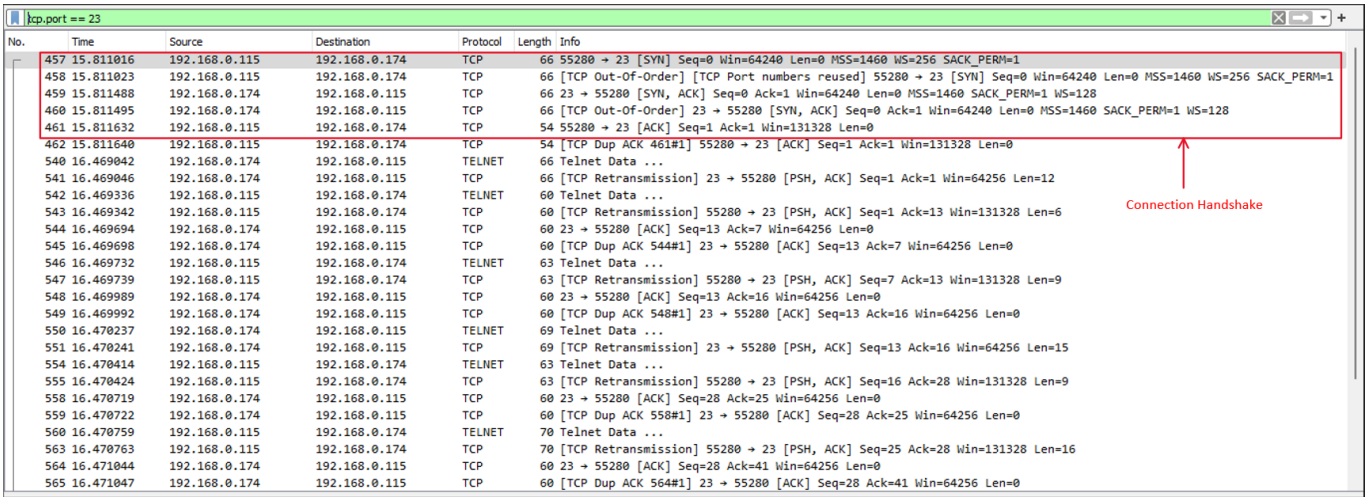
Question 2

Capture the packets while sending/receiving `telnet` request/response between your computer and a custom server running the `telnet` daemon. What is your observation while analyzing the application layer data?

Packet Capture

I have set up a telnet daemon in a VM which is in Bridged Mode. The VM's IP is `192.168.0.174`. I have installed telnetd which is actively listening on port 23.

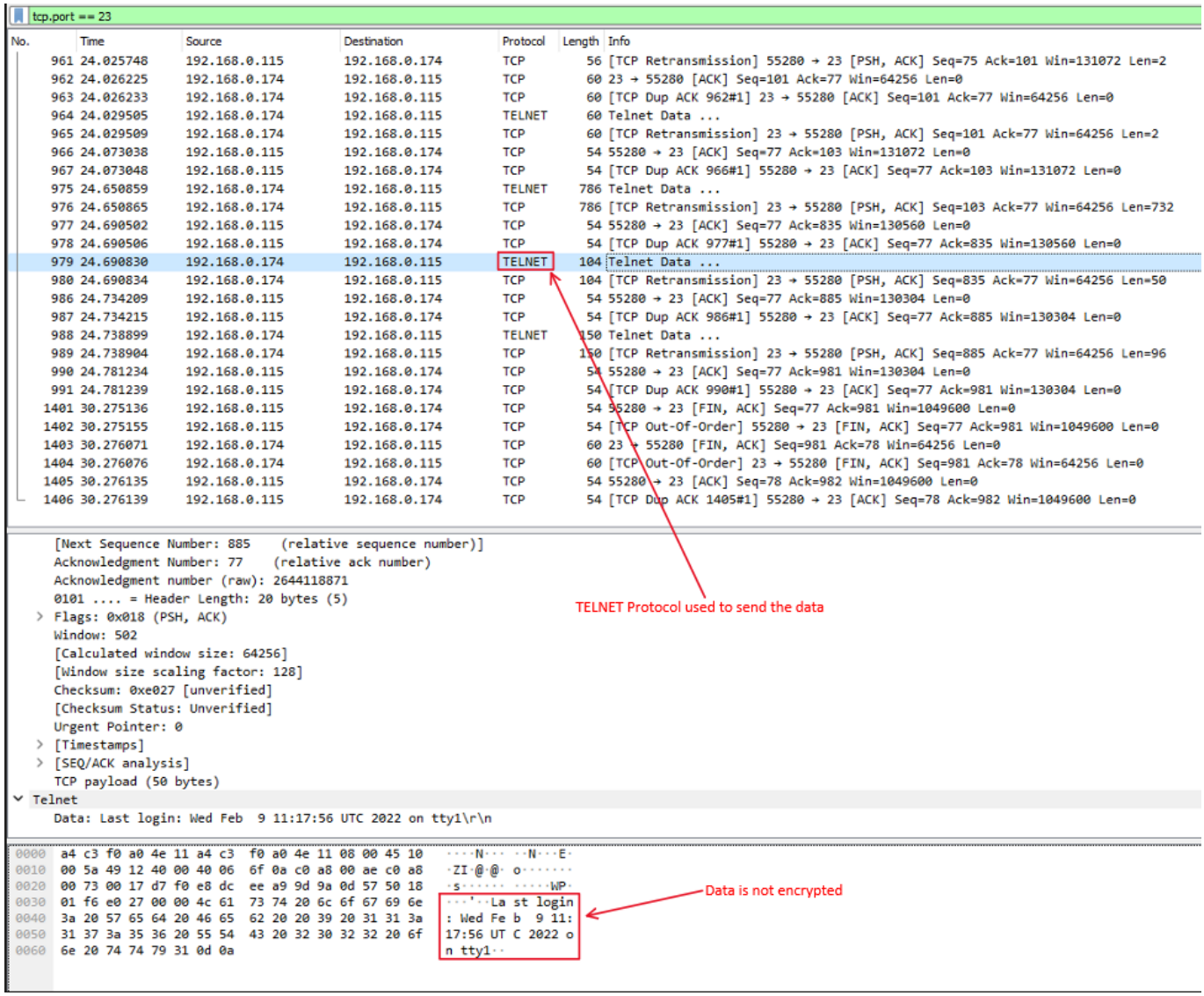
Connecting to VM via `telnet 192.168.0.174` and then capturing the packets via wireshark, we observe the following:



We observe that same three-way TCP handshaking is done between the telnet client and the telnet daemon.

After that every data is passed though the `TELNET` Protocol.

Observing the TELNET packets



- NOTE: Data Transfer is not encrypted and can be easily seen by sniffing softwares like wireshark.

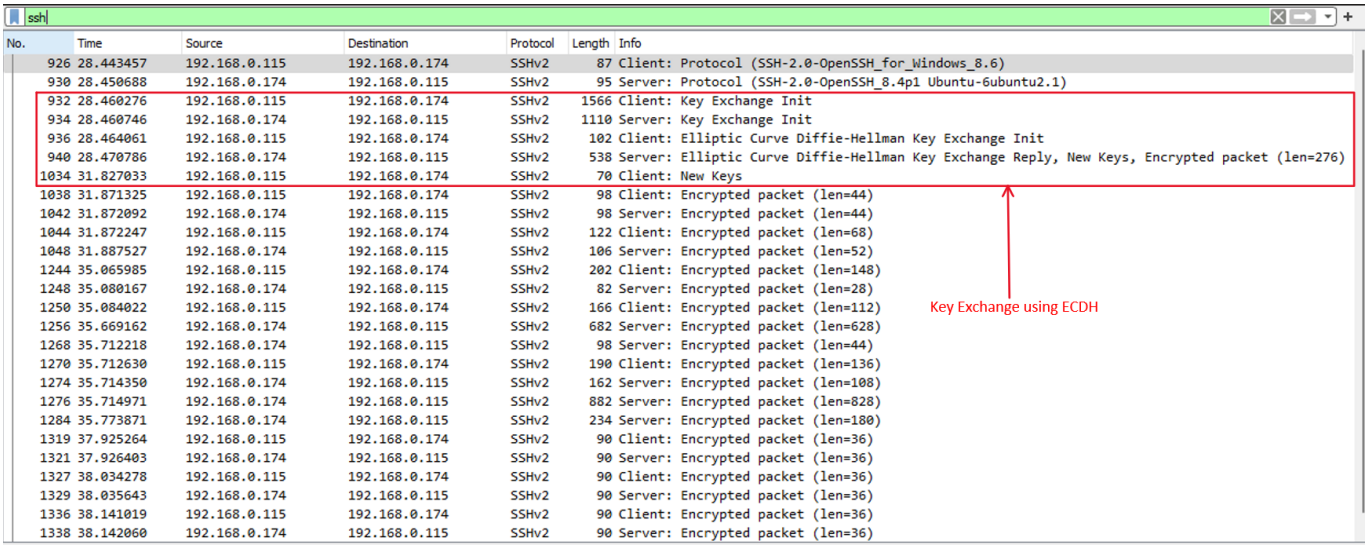
Question 3

Capture the packets while sending/receiving `ssh` request/response between your computer and one of the department servers. What is your observation while analyzing the application layer data?

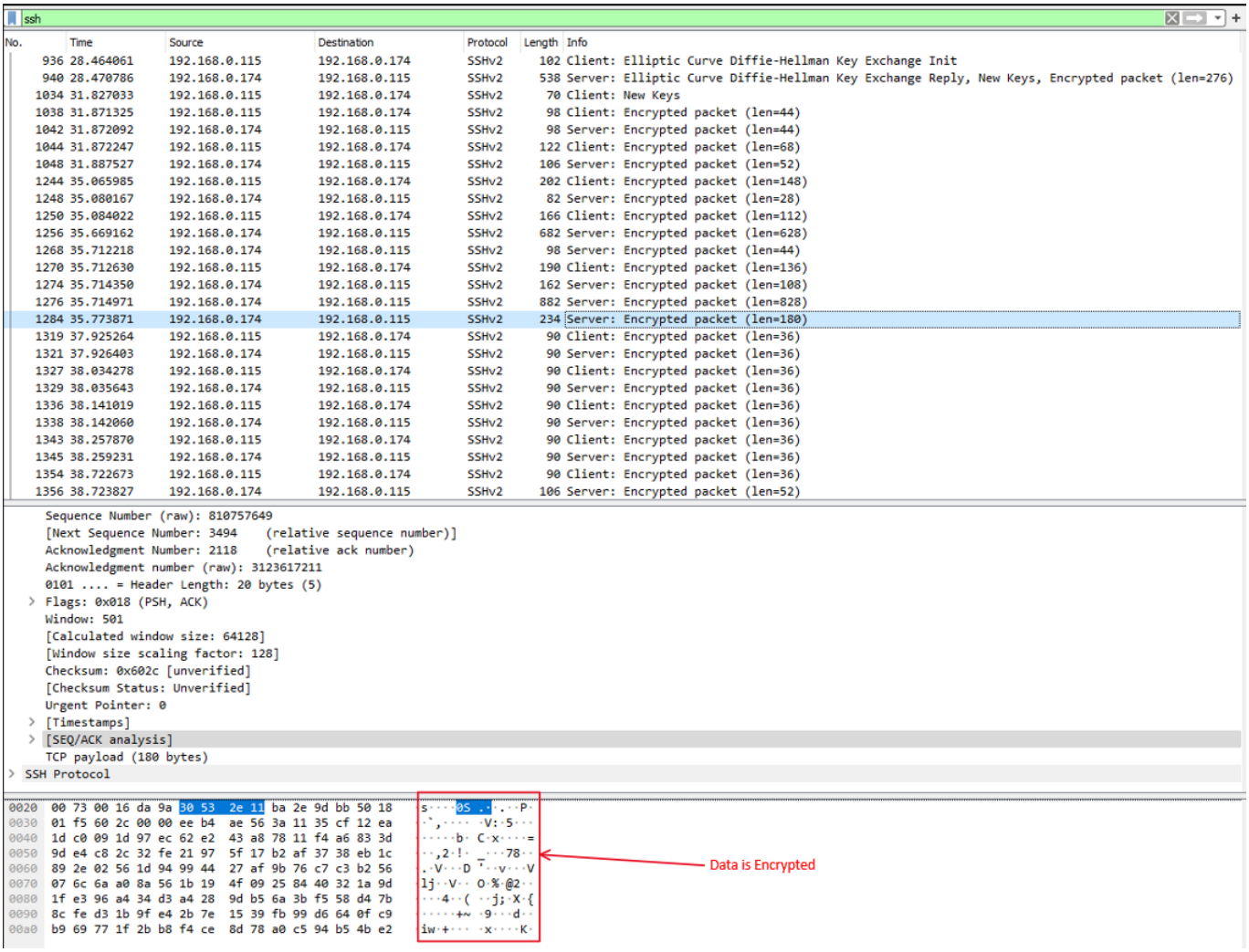
Packet Capture

SSH uses asymmetric cryptography to establish a shared secret key then uses the symmetric cryptography with the shared key for the remote session.

So, the first step of ssh will be to establish a shared secret key. Here we can see the server and the client initiating the key exchange via Elliptic Curve Diffie-Hellman (ECDH) key exchange algorithm.



After this the symmetric key is agreed upon the client and the server and all the remote session data is encrypted using the shared key.



Question 4

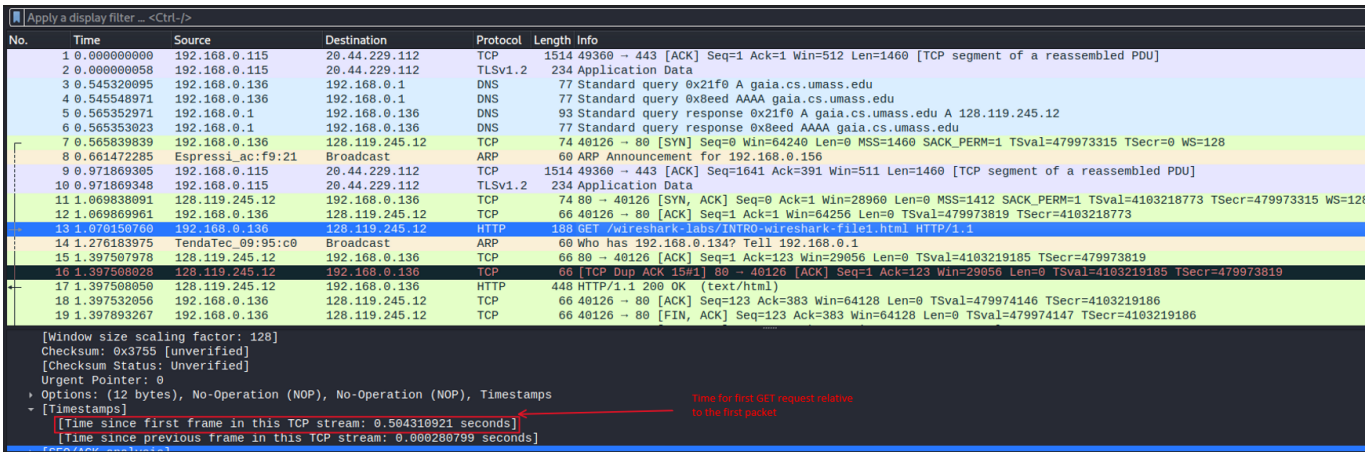
Enter the URL: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html and capture packets using `Wireshark`. After your browser has displayed the INTRO-wireshark-file1.html`page (it is a simple one line of congratulations), stop`Wireshark` packet capture. Answer the following from the packets captured:

1. How long did it take from when the `HTTP GET` message was sent until the `HTTP OK` reply was received?
2. What is the Internet address of the `gaia.cs.umass.edu`? What is the Internet address of your computer? Support your answer with an appropriate screenshot from your computer.

1. How long did it take from when the `HTTP GET` message was sent until the `HTTP OK` reply was received?

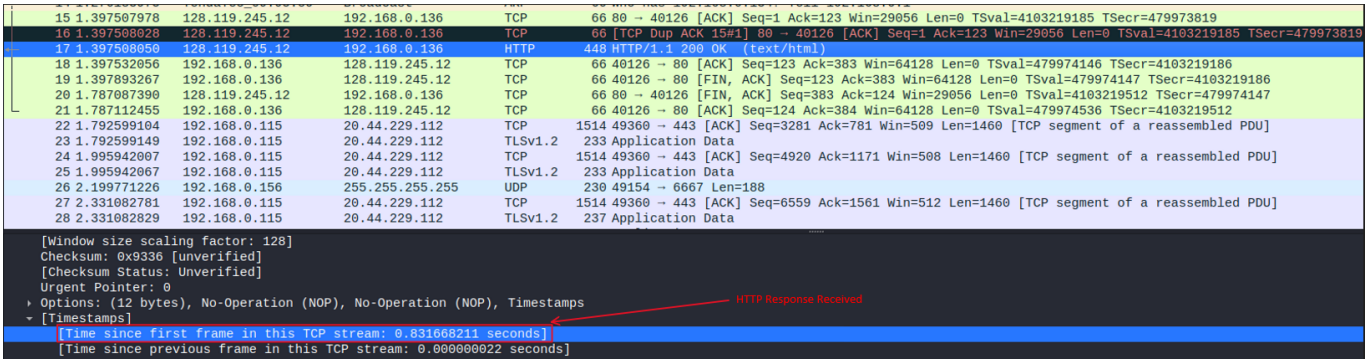
The first `GET` request was made in 0.504310921 s

[Time since first frame in this TCP stream: 0.504310921 seconds]



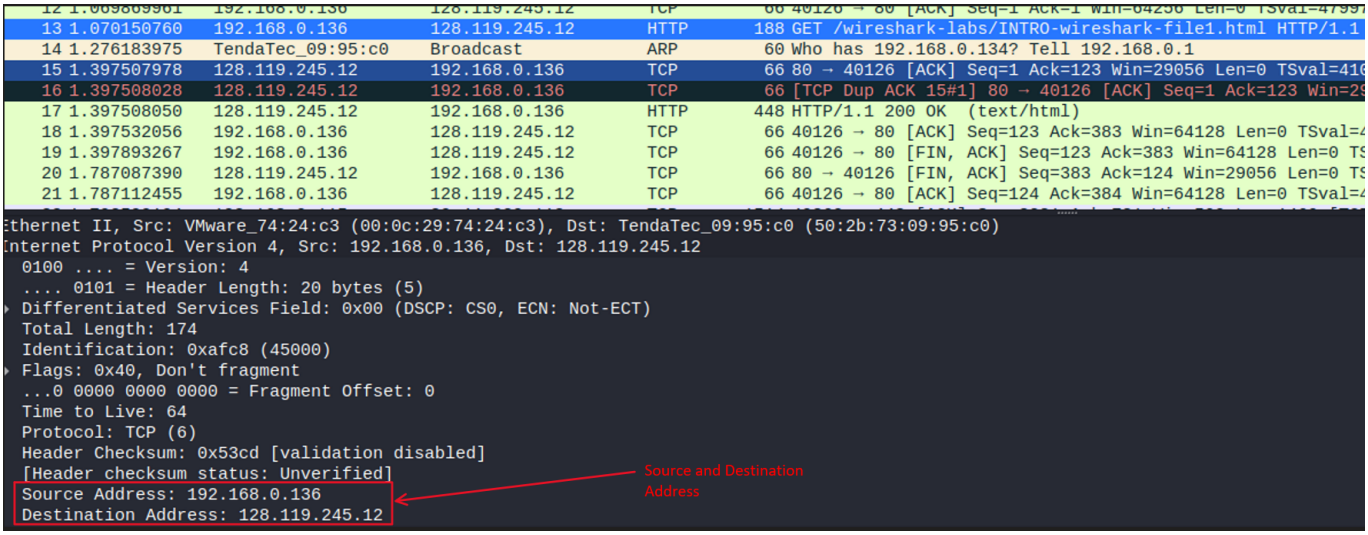
The first `GET` request was made in 0.831668211 s

[Time since first frame in this TCP stream: 0.83166821 seconds]



So Time Taken = 0.831668211 - 0.504310921 = 0.32735729 s

2. What is the Internet address of the `gaia.cs.umass.edu`? What is the Internet address of your computer? Support your answer with an appropriate screenshot from your computer.



- IP address of `gaia.cs.umass.edu` is `Destination Address: 128.119.245.12`
- IP address of my computer is `Source Address: 192.168.0.136`

Question 5

Start the `Wireshark` packet capturing service. Enter the URL: https://www.gmail.com/ on your browser and sign-in to your gmail account by providing credentials (Username/Password). Answer the following from the captured packets:

1. Is there any difference in the application layer protocol?
2. How it is different from the `HTTP` data you analysed in the above problem?

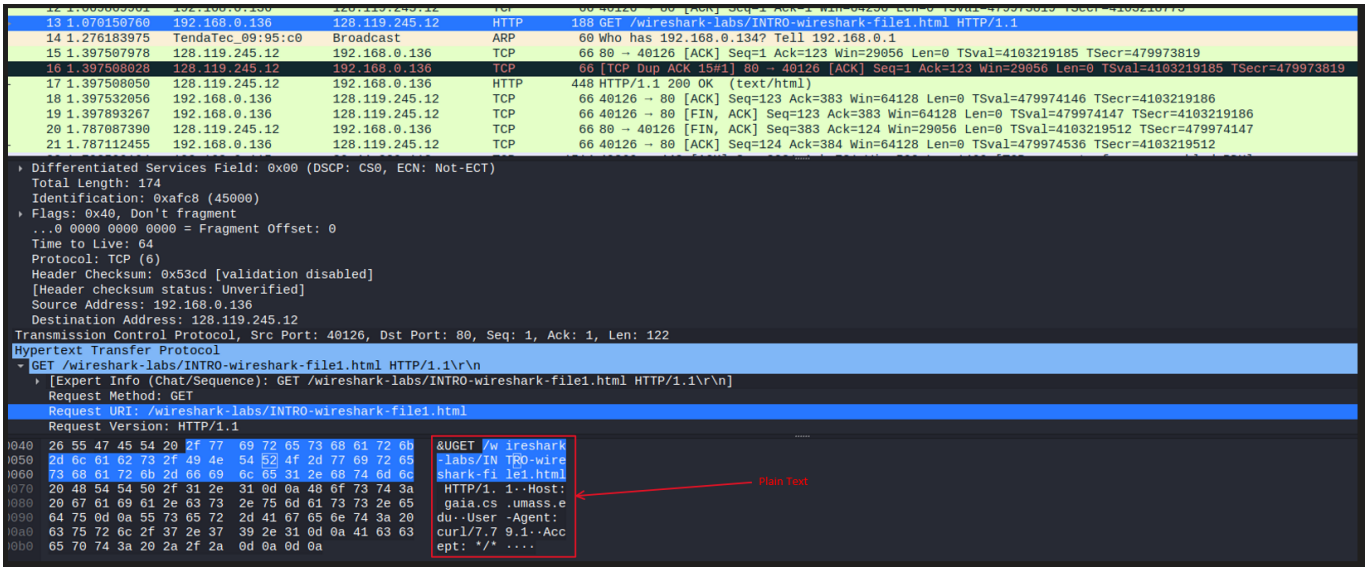
1. Is there any difference in the application layer protocol?

Yes, the current application layer protocol is `TLS` . A primary use case of `TLS` is encrypting the communication between web applications and servers, such as web browsers loading a website.

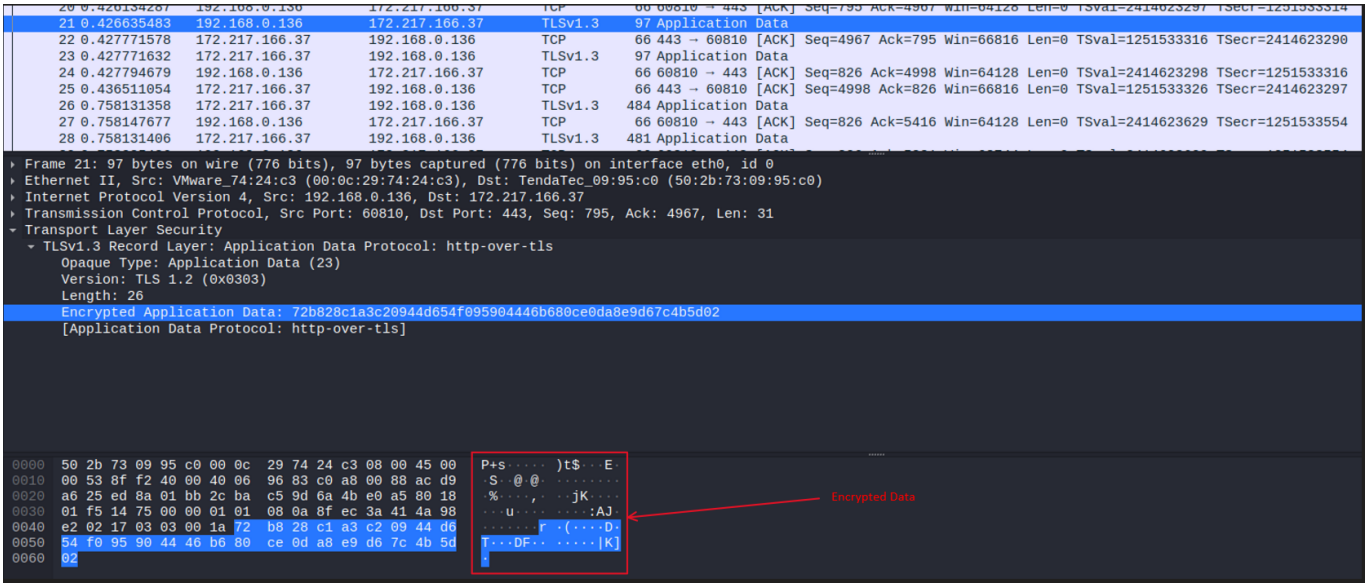
But, in the previous case it was just `HTTP`.

2. How it is different from the `HTTP` data you analysed in the above problem?

Previously, the protocol used was simple HTTP. As a result sniffers like wireshark could decode the data. Here is an example:



But https://www.gmail.com/ is using HTTPS with TLS. So the data will be encrypted.



here sniffers like wireshark cannot decode the data. Hence some random person could read the username and password from the packets.