Sanyam Kaul

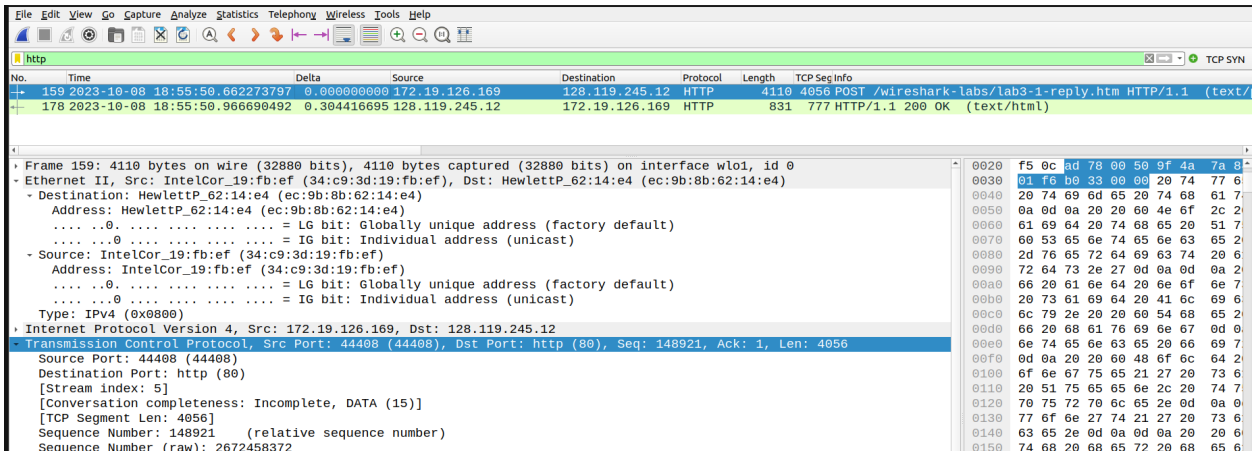
CS23MTECH14011

08 October 2023

Wireshark TCP Assignment

PART - A

Ans 1



We can see in the snapshot above that the source IP is - 172.19.126.169

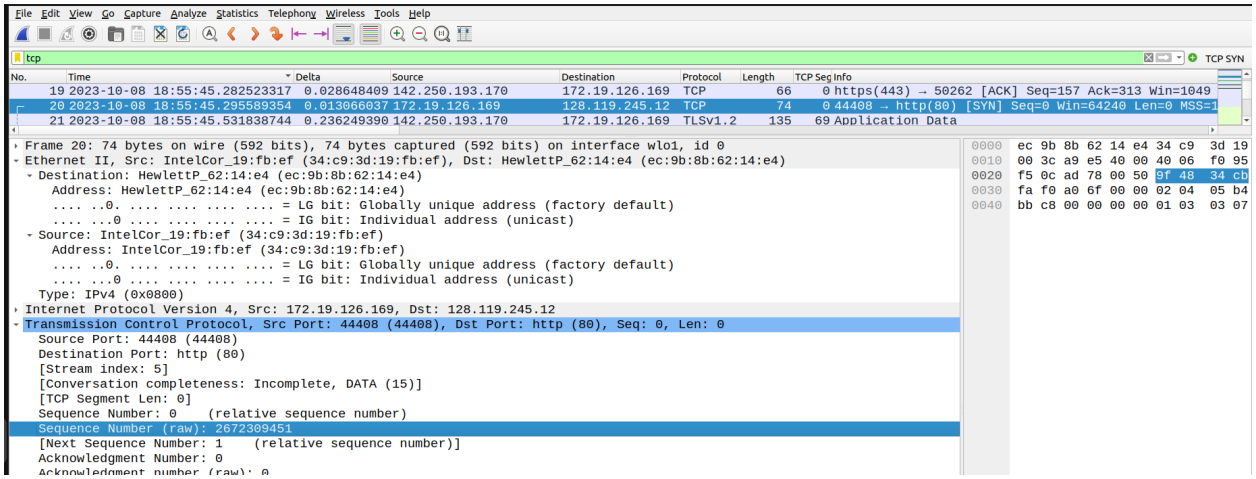
The source port is - 44408

Ans 2

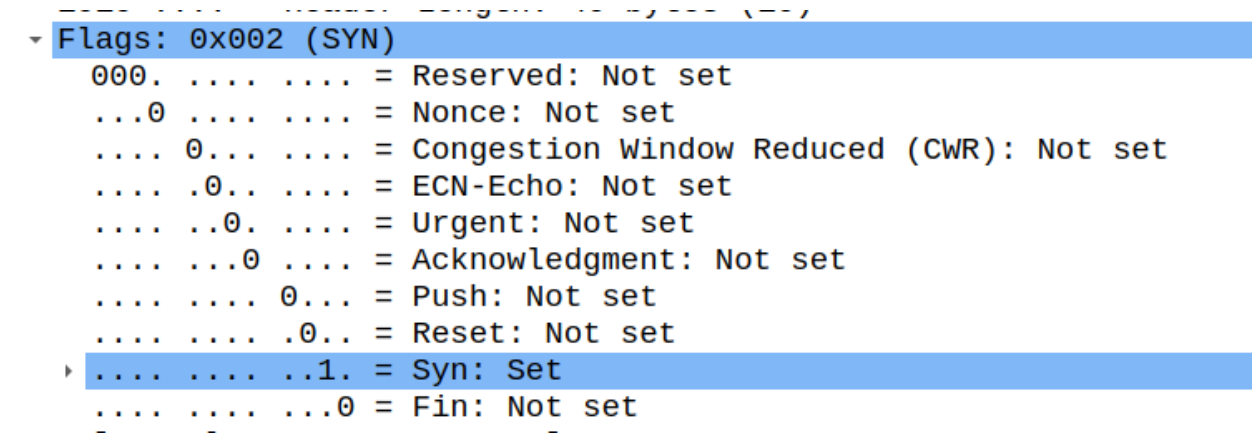
The IP address of the destination is - 128.119.245.12

The port number is 80 which indicates that it is an HTTP server

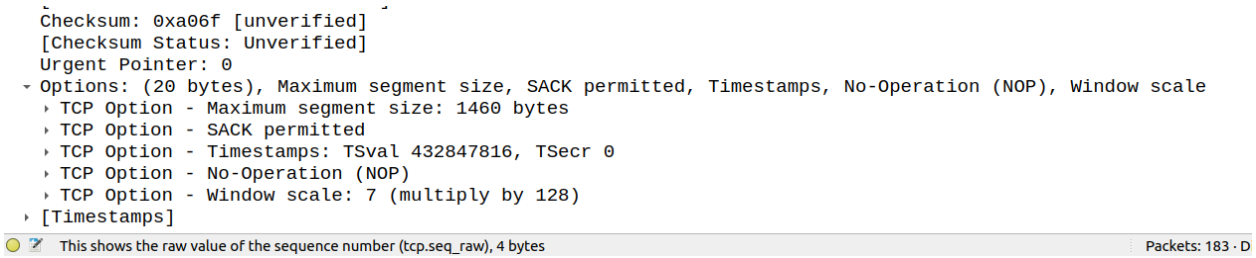
Ans 3

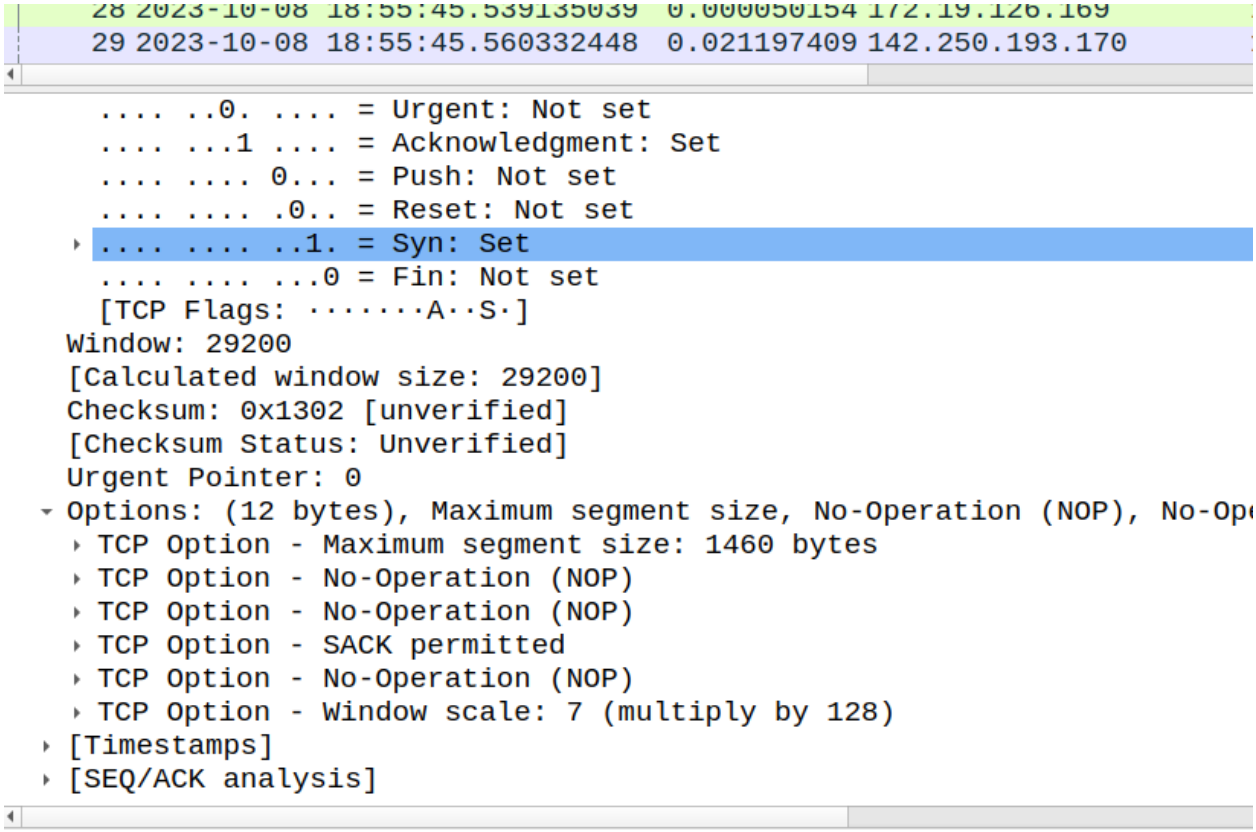


As seen in the snapshot above, the sequence number is - 2672309451



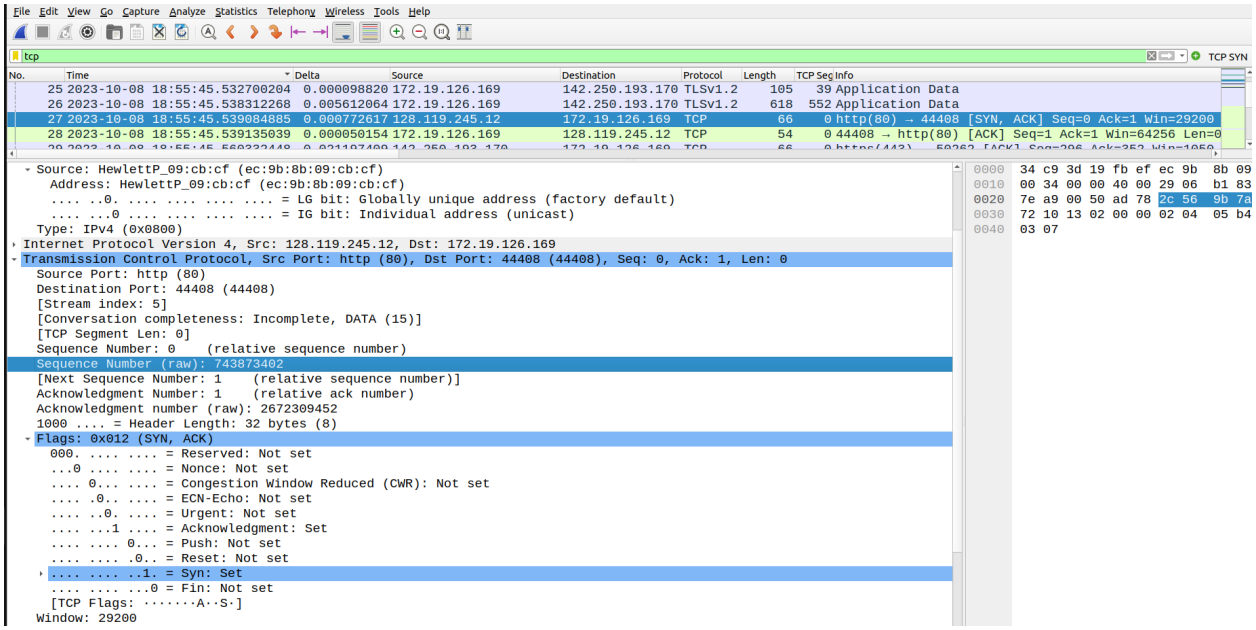
The SYN flag is set to 1 which marks this TCP segment as the SYN segment





I can see in the snapshots above that SACK is permitted for both the SYN and SYNACK segments which shows that the TCP receiver can use selective acknowledgement

Ans 4



The sequence number is - 743873402

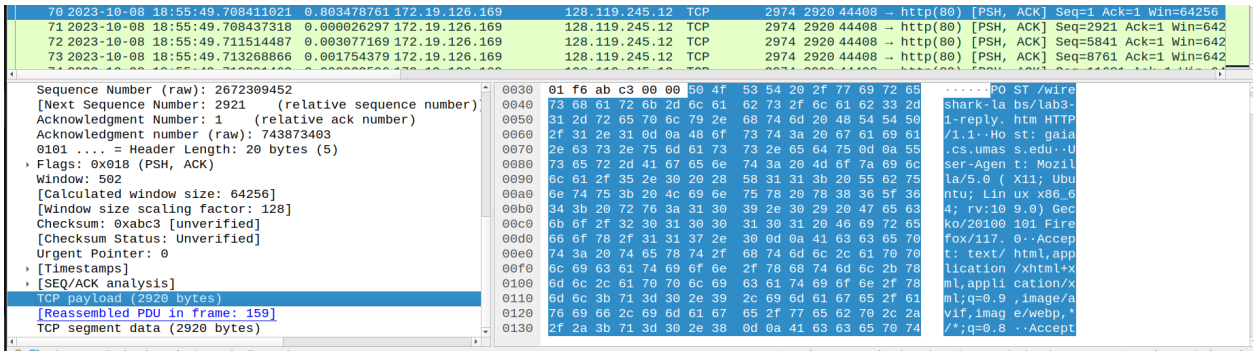
The SYN and ACK flags are set to 1 which shows that this is the SYN, ACK packet

The Acknowledgement number is - 2672309452

The seq. No. of the SYN segment sent before this SYNACK segment was - 2672309451.

Therefore the ack no. of this SYNACK segment is ‘seq. No. of the SYN segment’ + 1. The server is saying to the client that the segment with seq. No - seq. No. 2672309451 is ACKed, send me the next segment with seq. No - 2672309452 now

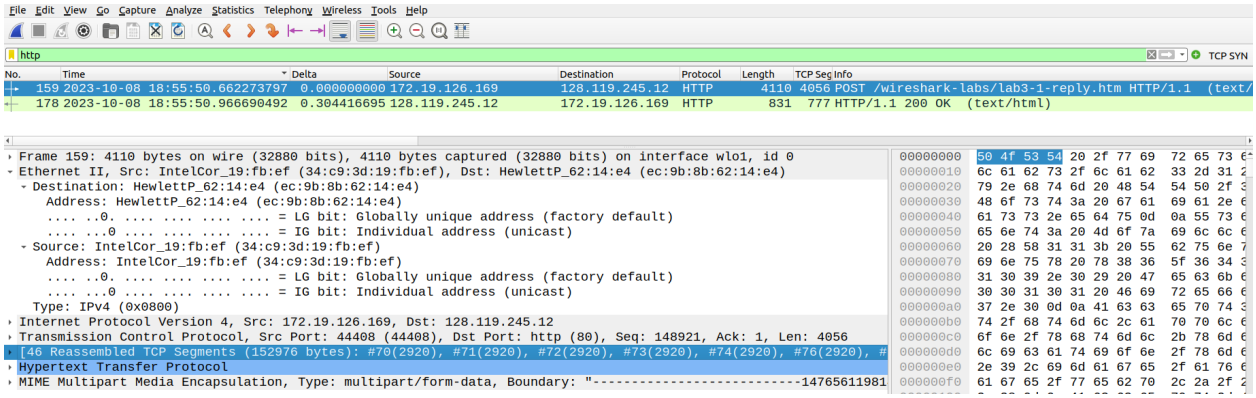
Ans 5



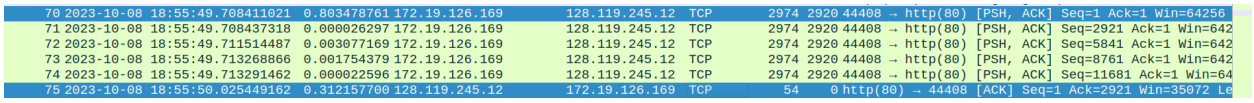
The snapshot above shows details of the TCP segment containing the header of the HTTP POST command. We can see in the segment data details shown on the right side of the screen that the segment has HTTP/1.1 POST command details

The seq. No of this segment is - 2672309452

The payload field of this TCP segment has 2920 bytes. The following snapshot of the HTTP packet shows that the total size of the payload (alice.txt) being sent is 152976 bytes. A single segment can not contain all 152976 bytes



Ans 6



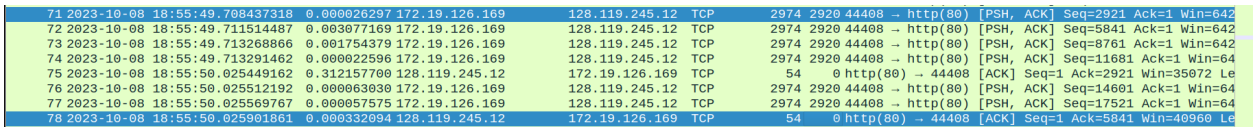
70	2023-10-08 18:55:49.708411021	0.803478761	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=1 Ack=1 Win=64256
71	2023-10-08 18:55:49.708437318	0.000026297	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=2921 Ack=1 Win=642
72	2023-10-08 18:55:49.711514487	0.003077169	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=5841 Ack=1 Win=642
73	2023-10-08 18:55:49.713268866	0.001754379	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=8761 Ack=1 Win=642
74	2023-10-08 18:55:49.713291462	0.000022596	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=11681 Ack=1 Win=64
75	2023-10-08 18:55:50.025449162	0.312157700	128.119.245.12	172.19.126.169	TCP	54	0	http(80) → 44408	[ACK] Seq=1 Ack=2921 Win=35072 Le

The snapshot above shows the TCP segment with HTTP POST (first blue row) and its ACK (second blue row).

The first segment was sent at - 2023-10-08 18:55:49.708411021

The ACK was received at - 2023-10-08 18:55:50.025449162

RTT for the first data-containing segment - 0.3174 sec



71	2023-10-08 18:55:49.708437318	0.000026297	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=2921 Ack=1 Win=642
72	2023-10-08 18:55:49.711514487	0.003077169	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=5841 Ack=1 Win=642
73	2023-10-08 18:55:49.713268866	0.001754379	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=8761 Ack=1 Win=642
74	2023-10-08 18:55:49.713291462	0.000022596	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=11681 Ack=1 Win=64
75	2023-10-08 18:55:50.025449162	0.312157700	128.119.245.12	172.19.126.169	TCP	54	0	http(80) → 44408	[ACK] Seq=1 Ack=2921 Win=35072 Le
76	2023-10-08 18:55:50.025512192	0.000063030	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=14601 Ack=1 Win=64
77	2023-10-08 18:55:50.025569767	0.000057575	172.19.126.169	128.119.245.12	TCP	2974	2920	44408 → http(80)	[PSH, ACK] Seq=17521 Ack=1 Win=64
78	2023-10-08 18:55:50.025901861	0.000332094	128.119.245.12	172.19.126.169	TCP	54	0	http(80) → 44408	[ACK] Seq=1 Ack=5841 Win=40960 Le

Snapshot above shows the 2nd data-carrying TCP segment and its ACK response (both highlighted in blue)

The second segment was sent at - 2023-10-08 18:55:49.708437318

The ACK was received at - 2023-10-08 18:55:50.025901861

RTT of second data carrying segment - 0.3175 sec

EstimatedRTT value after the ACK of second data carrying segment is received: -

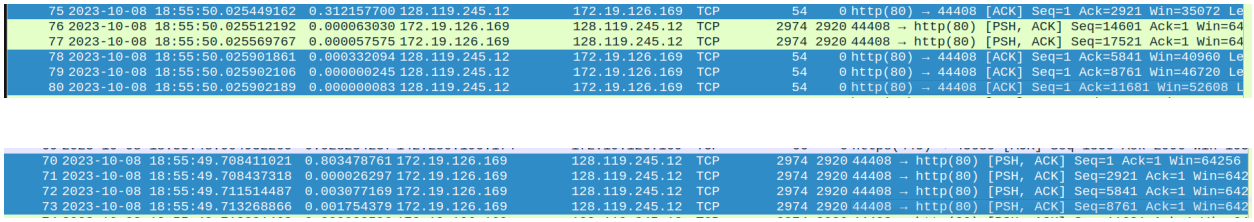
Alpha = 0.125

Initial Estimated RTT = 0.3174 sec

New Estimated RTT = 0.875*0.3174 + 0.125*0.3175 = 0.2777 + 0.0396 = 0.3173 sec

Ans 7

The length of each of the first 4 data-carrying TCP segment= 2974 bytes (snapshot below)



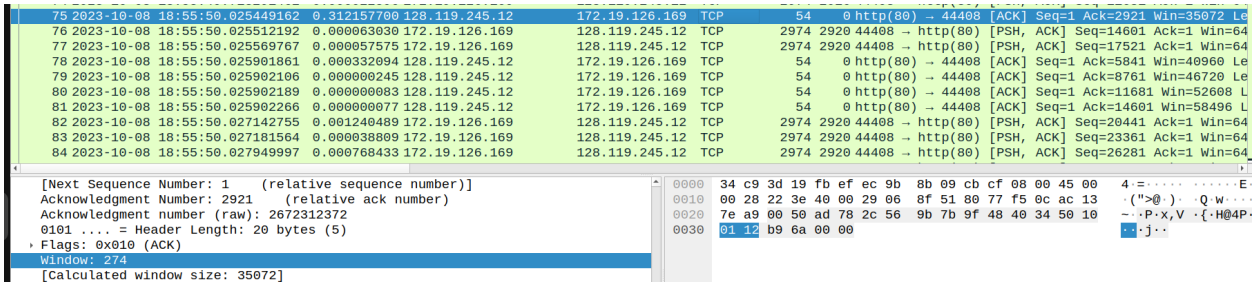
Ans 8

The snapshot above shows the ACKs of the first five data-carrying TCP segments.

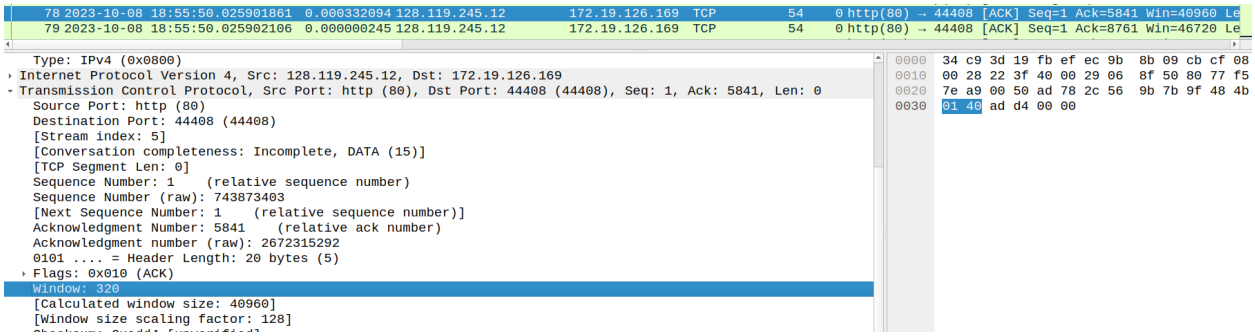
The window sizes advertised are as follows: -

Window size = (window value) * (scaling factor value)

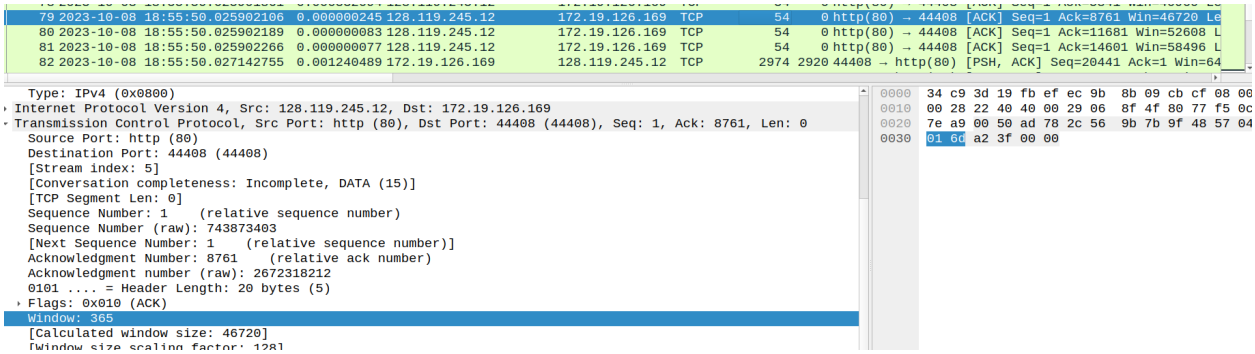
Window size advertised by first ACK = 35072 bytes



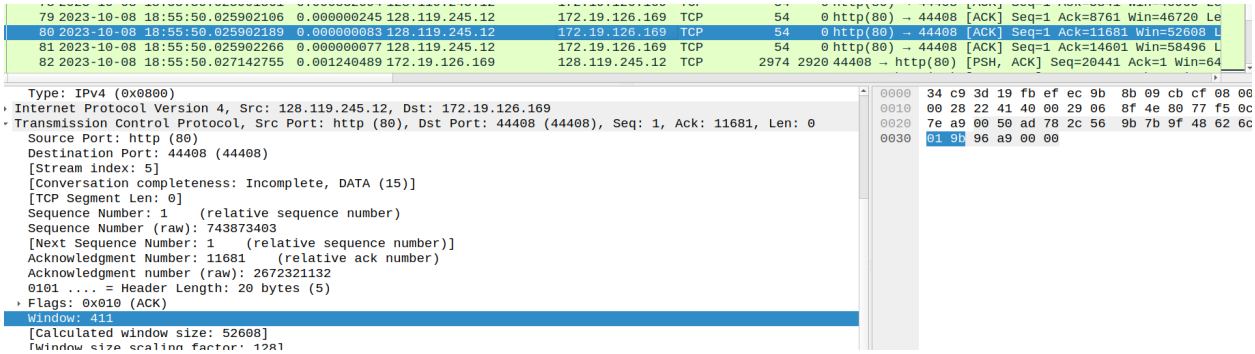
Window size advertised by second ACK = 40960 bytes



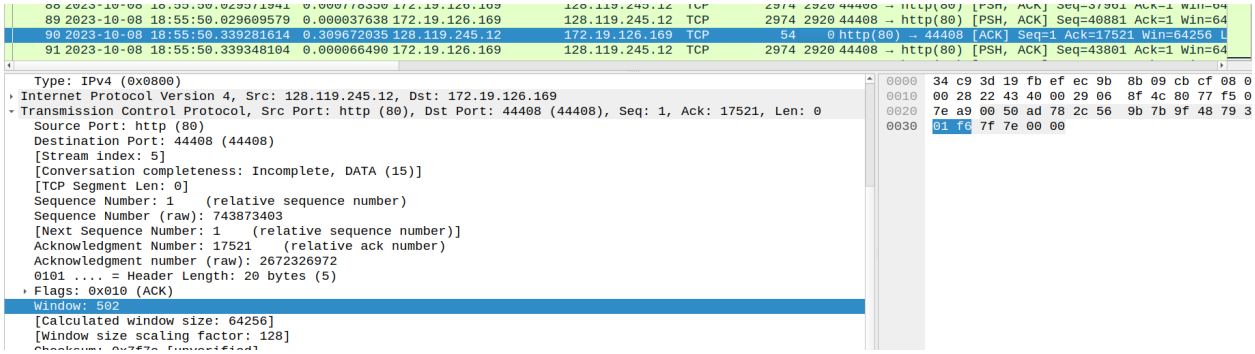
Window size advertised by third ACK = 46720 bytes



Window size advertised by fourth ACK = 52608 bytes



Window size advertised by fifth ACK = 64256 bytes

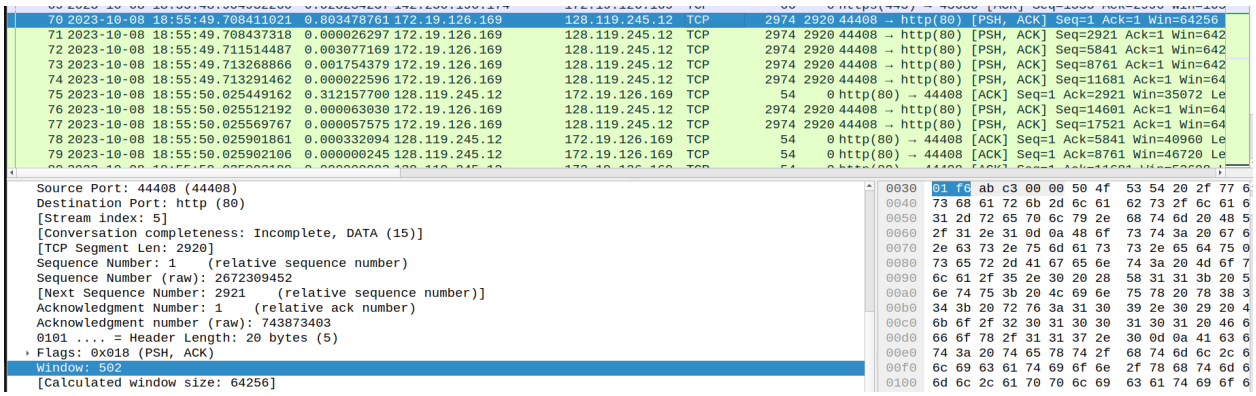


Hence the minimum size is 35072 bytes

As can be seen from the snapshots above, the window size has increased from the first to fifth ACK. The receiver has never advertised a smaller window size than the previous one in the first five ACKs

Ans 9

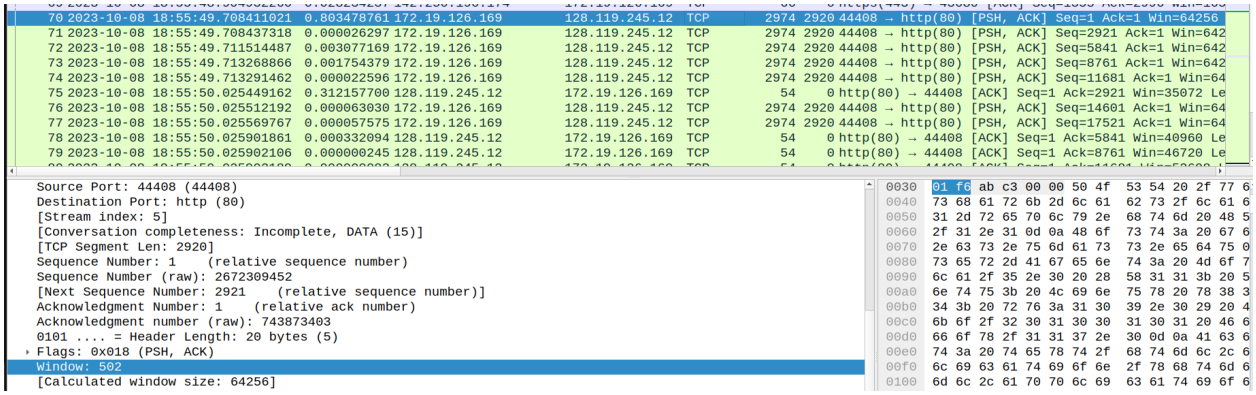
In this case we will look at the Window value in the first five data-carrying TCP segments. For me, this window value is 64256 bytes for each of the first five TCP segments. This is because the client is not receiving anything from the server apart from the ACK segments. So there is no buffer utilization on the client side.



Ans 10

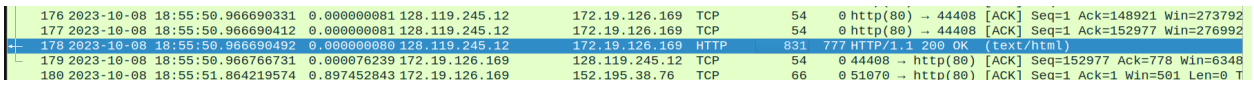
No, there are no re-transmitted segments in the trace file. In order to look for any re-transmitted segments, I looked at the ack values of all the ACK packets received and looked for any 2 packets with the same ack values. The same ACK values would have specified the duplicate ACK which would have happened in case of re-transmission.

Ans 11



The receiver is typically acknowledging 2920 bytes. Client is sending one TCP segment with length 2920 bytes and the server is acking it. In this scenario, there is no instance of receiver acking every other segment. I am getting ACK for every TCP segment sent.

Ans 12



The snapshot above shows the TCP segment received from the server (highlighted in blue) with HTTP code 200. Just above that is the last ACK segment received from the server. This segment confirms that at this time server had received the whole file.

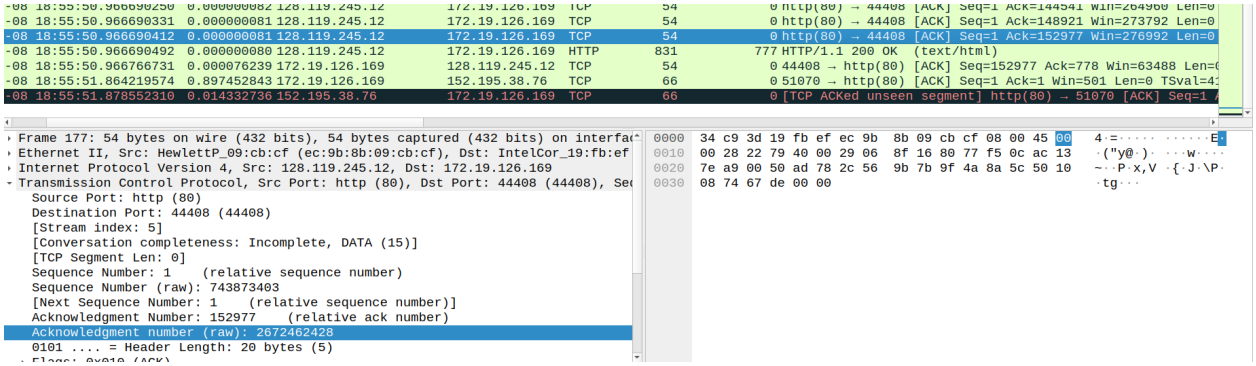
The time of final ack = 18:55:50.966690331

The SYN TCP segment was sent at = 18:55:45.295589354

Total time taken to transfer the file by TCP = 5.6711 sec

Total bytes transferred = File size = 152976 bytes (this can be confirmed by looking at the analysis below)

The snapshot below shows the last TCP ACK received before receiving the HTTP 200 response. Note that its relative ACK value is 152977. Subtracting 1 from this because this 1 corresponds to the first TCP segment sent to server with seq no - 1 and ack - 0, we get 152976. This means that the last ACK segment received from the server is saying that the server has acked 152976 bytes which is the total file size.



Therefore, Throughput = (file size) / time = 152976 / 5.6711 = 26.97 Kbps

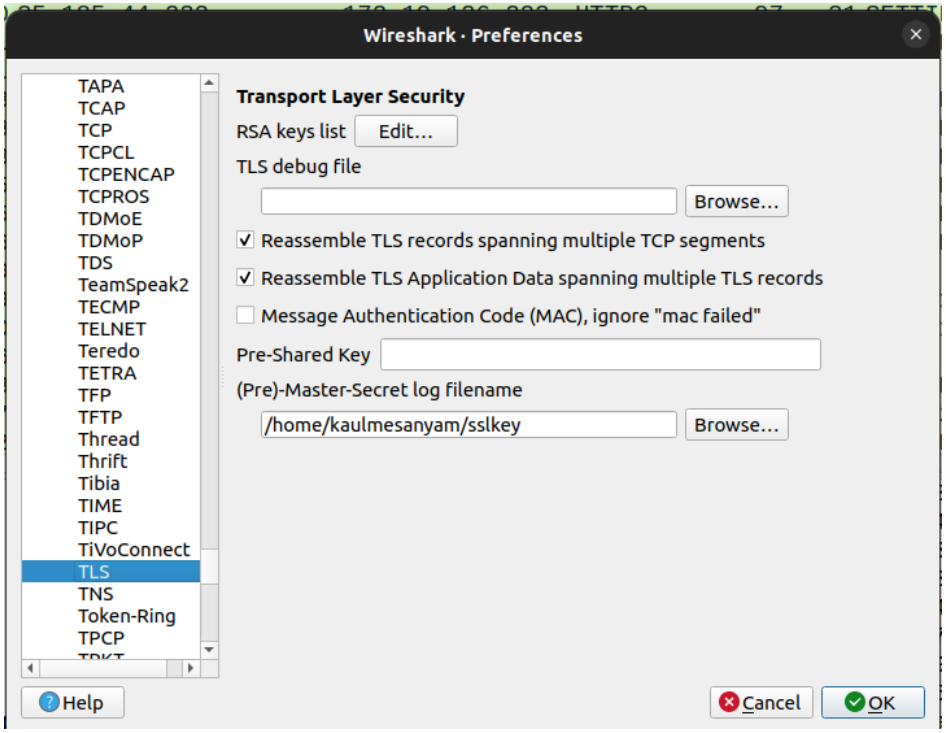
PART - B

Since the server is HTTPS, in this case, we will be getting SSL/TLS connection with encrypted data. I have set up an SSL keylog file to get hold of all the encryption keys which are used by TLS connection for encryption.

The command used to make the keylog file: -

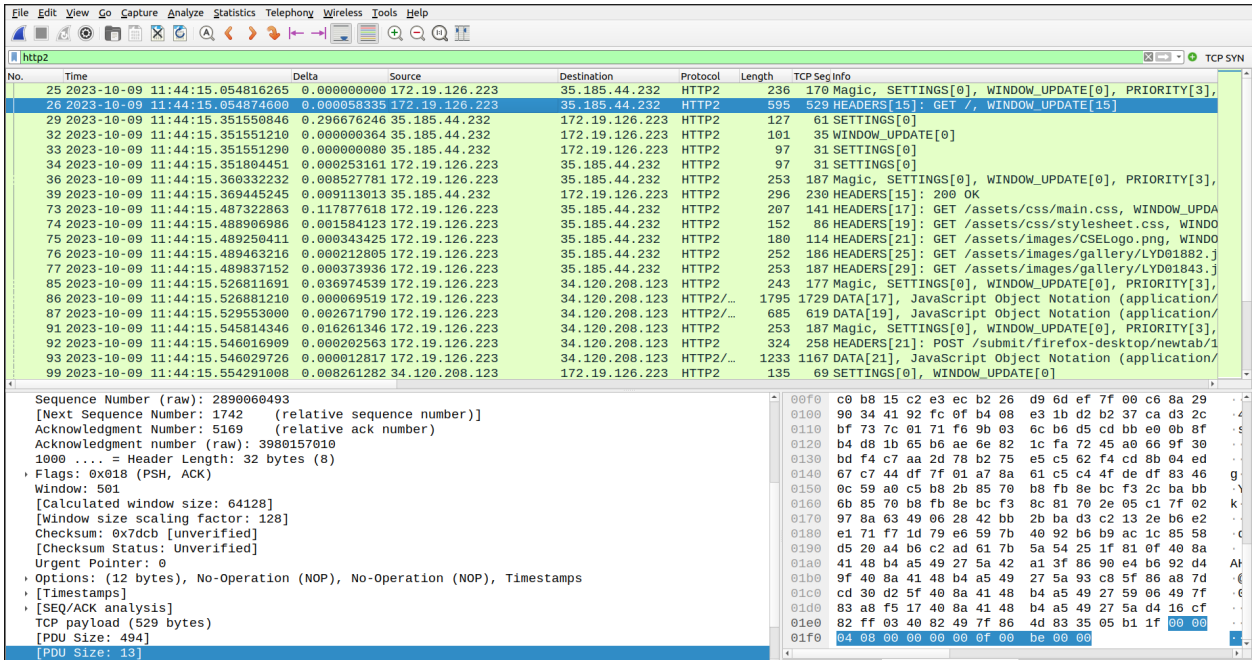
export SSLKEYLOGFILE=”/home/kaulmesanyam/sslkey”

Next, I have added the file path of this key log file in Wireshark : -



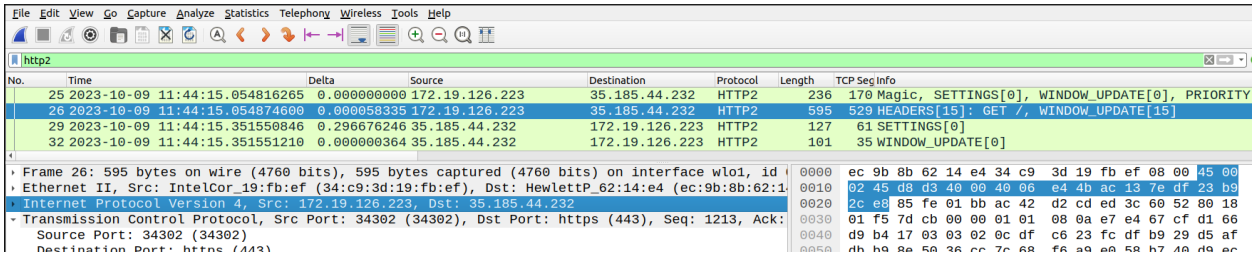
Because of this, all the TLS v1.3 packets in my capture have been decrypted to respective

HTTP2 packets and I have detailed information about the packet transfer that have taken place: -



I will be using this capture profile to answer the questions now

Ans 1



The snapshot above shows the decrypted HTTP2 GET request packet which was sent from my system to the server.

Source IP address - 172.19.126.223

Port Number - 34302

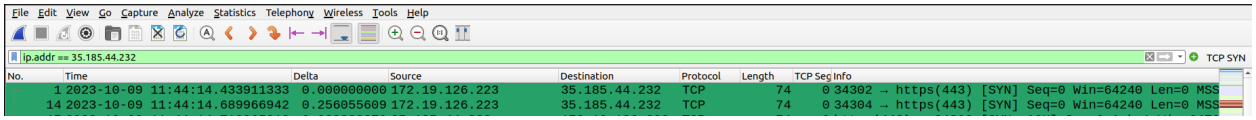
Ans 2

Using the same snapshot given above: -

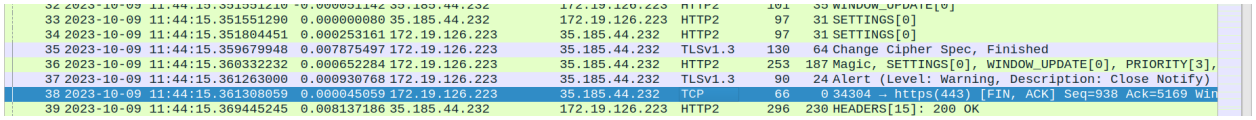
Destination IP = 35.18544.232

Port Number = 443 (depicting that it is an HTTPS server)

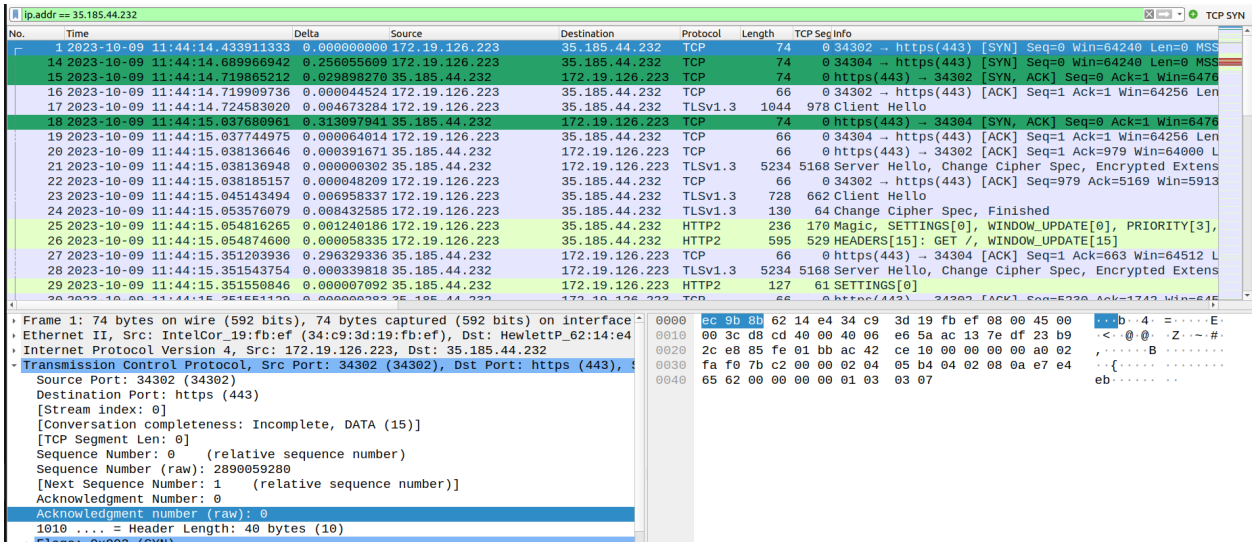
Ans 3



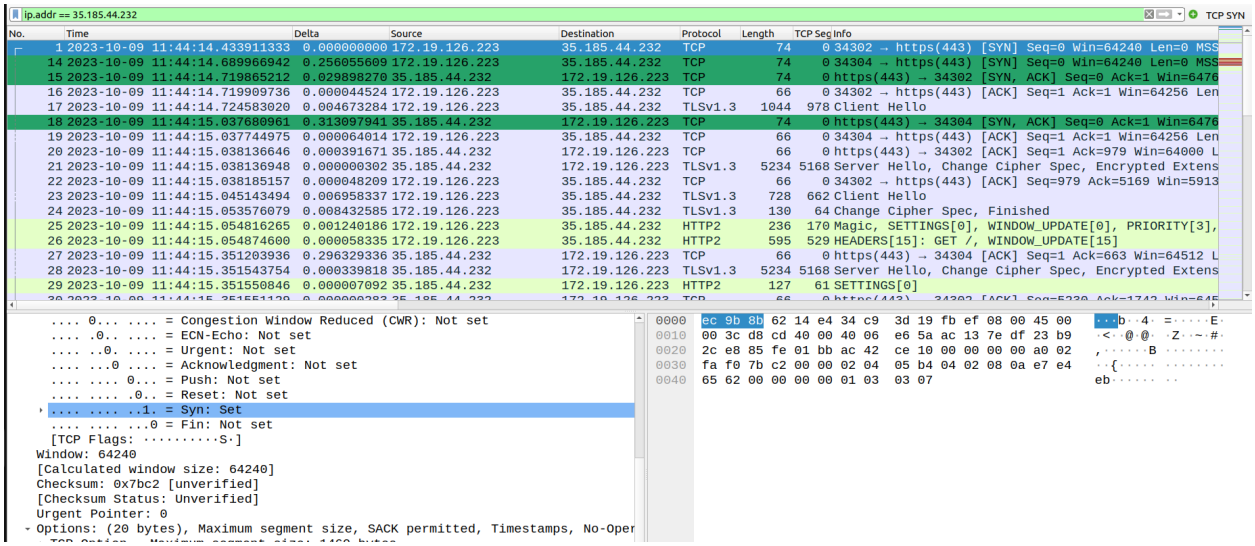
The scan shows that the browser sent 2 SYN packets, one on port 34302 and other on port 34304. I can see a complete TCP handshake for both of these TCP SYNs. This might be the case that the browser created 2 TCP connections in parallel and fetched the response from the one which was faster than the other.



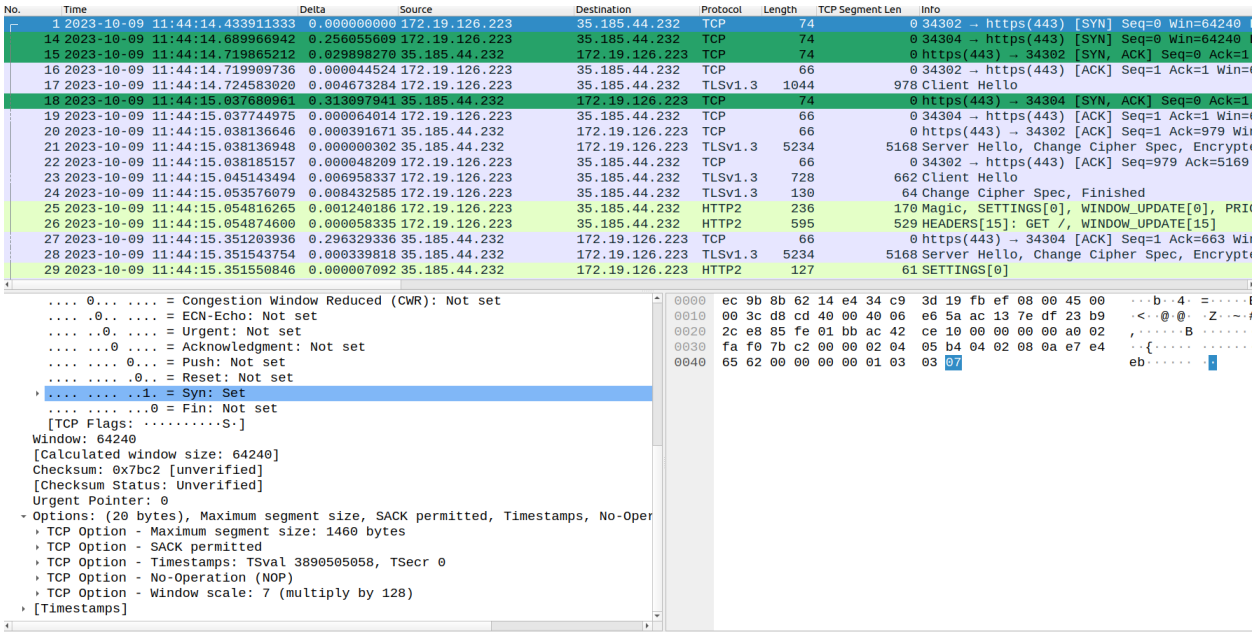
This snapshot shows that a FIN packet has been sent to the server from port 34304 but the TCP connection on port 34302 is still alive. From this, it seems that the TCP connection from port 34302 was the faster one in getting established. The timestamps of SYNACK and ACK for port 34302 also show that they came earlier. Moreover, as I proceed with the scan I am getting a lot of exchange of packets on the TCP connection on port 34302. They seem to be the packets containing payloads of images which are displayed on the website. Hence, it looks like the TCP connection is on port 34302 So I will pick this connection for further analysis.

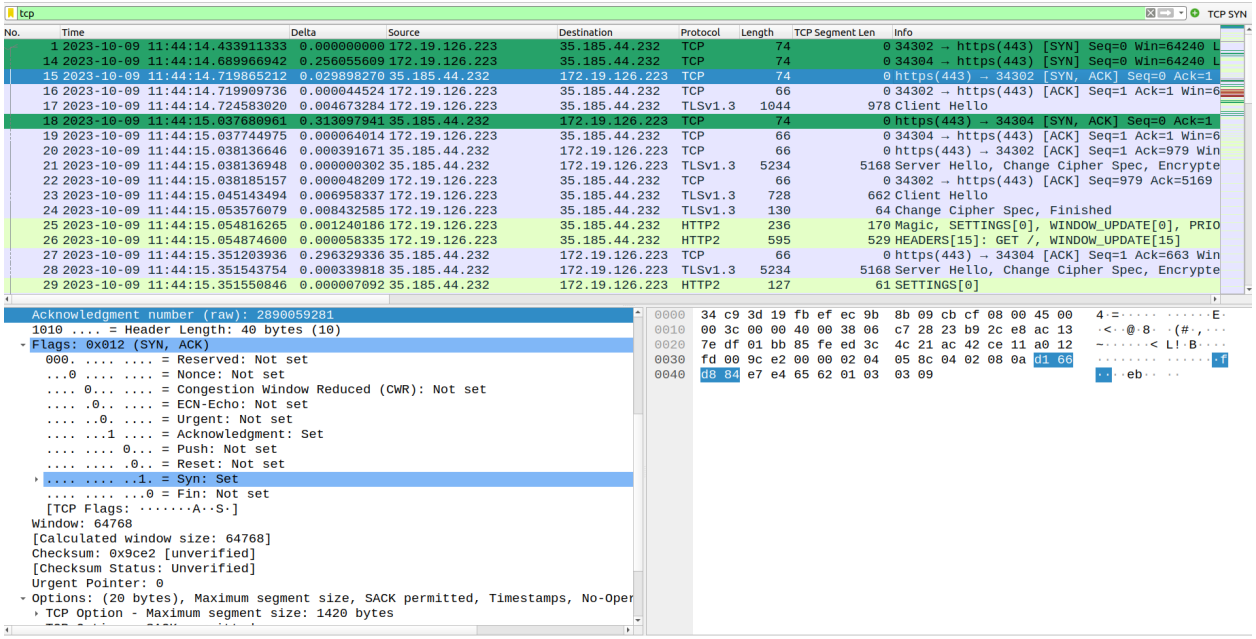


The Sequence number of the TCP SYN from port 34302 is = 2890059280



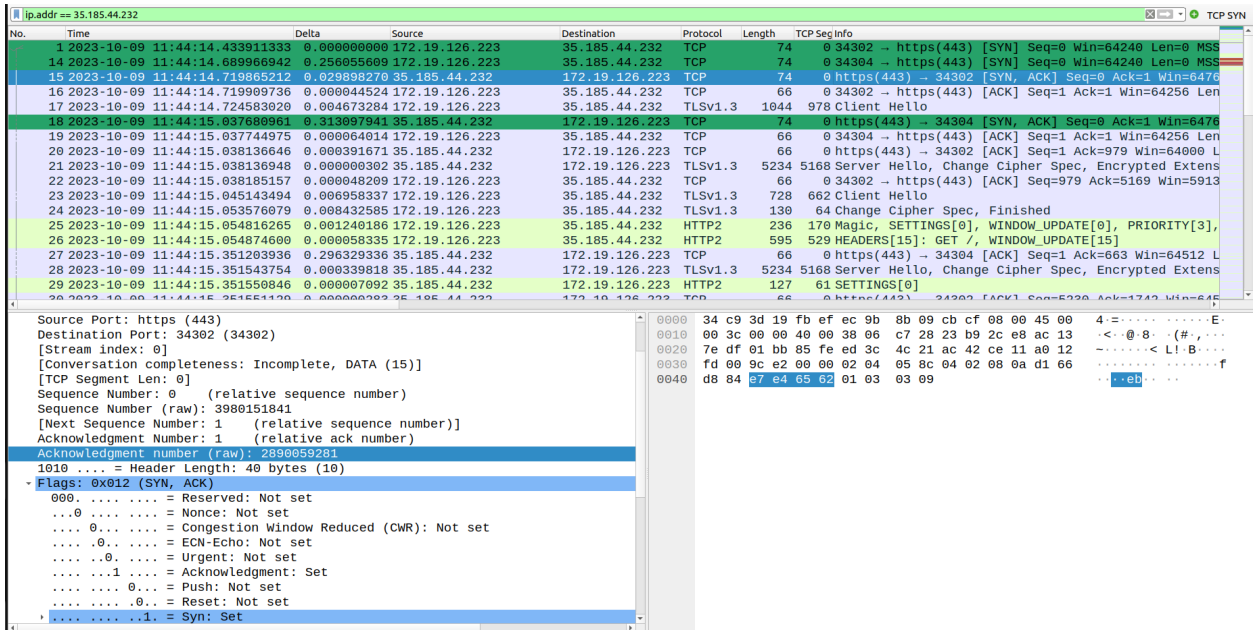
The SYN flag is set to 1 for SYN flag which confirms that this is a SYN TCP segment





SACK is permitted for both SYN and SYNACK segments which confirms that the receiver can use selective acknowledgement

Ans 4

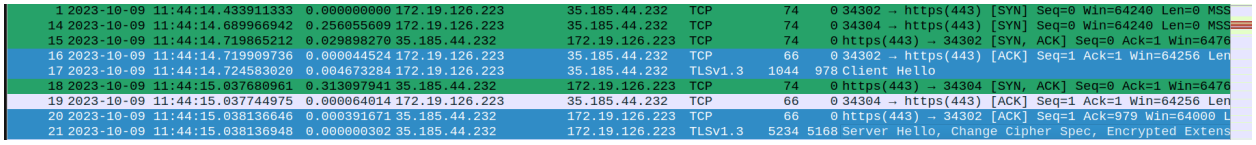


The sequence number of the SYNACK segment is 3980151841

Both the SYN and ACK flags are set confirming that this is the SYNACK segment

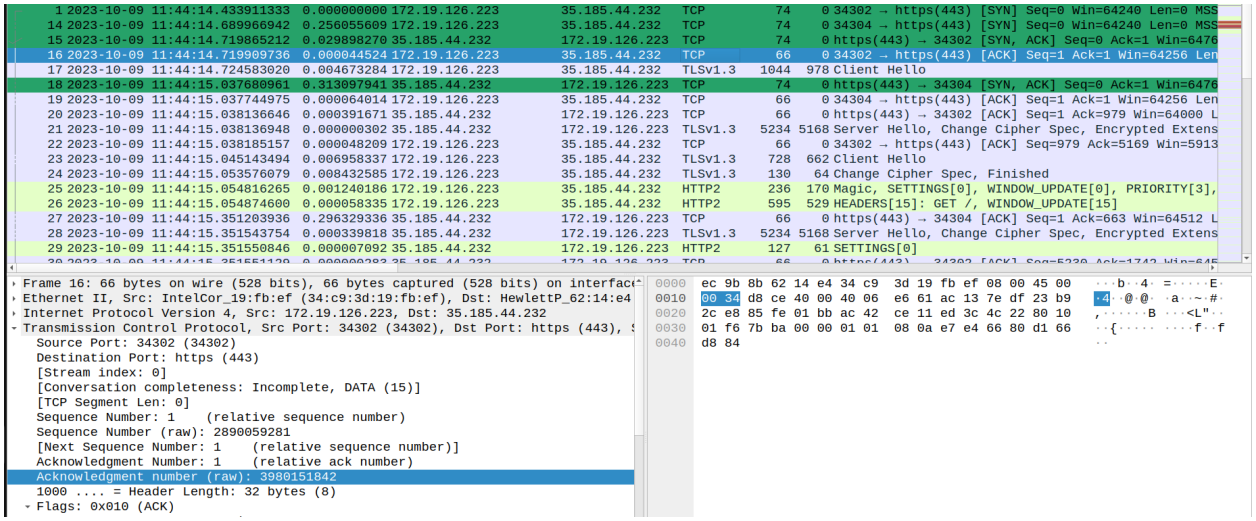
The Ack number of the SYNACK number is 2890059281. This is because the seq. No. of SYN segment was 2890059280. The server is saying that it has acked all segments till 2890059280 and is now waiting for (2890059280 + 1)th segment

Ans 5

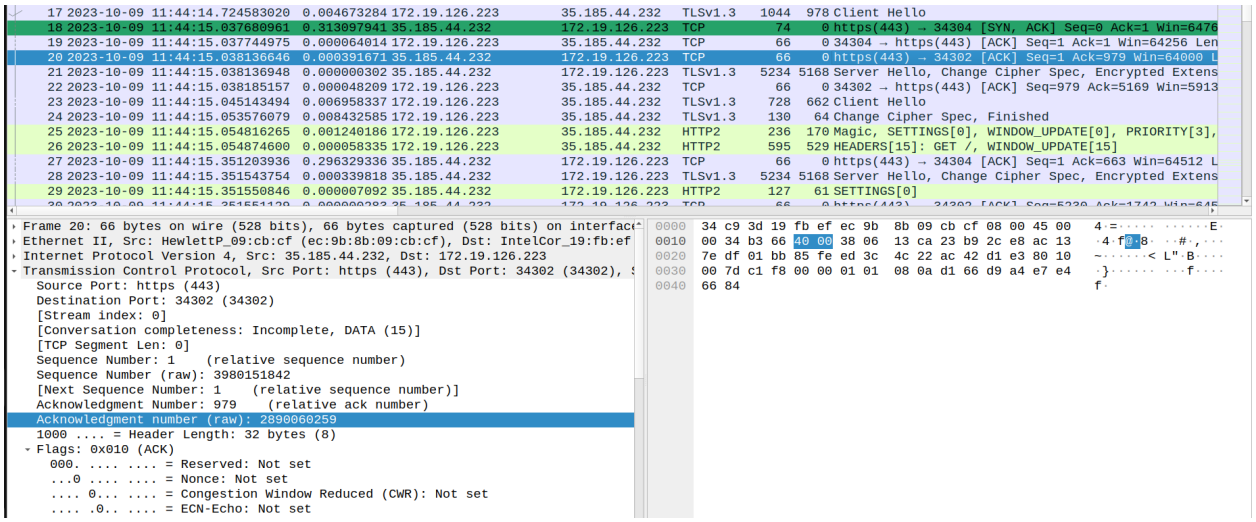


Notice the packets highlighted in blue.

After the SYNACK is received, the client has sent the ACK segment. Along with this ACK a TLSv1.3 packet is also sent which is the ‘Client hello’ packet. This packet has initiated the TLS handshake which establishes the HTTPS connection between the client and server. The sequence number of the TCP segment initiating this is 2890059281 and the ack no is 3980151842 (snapshot below)

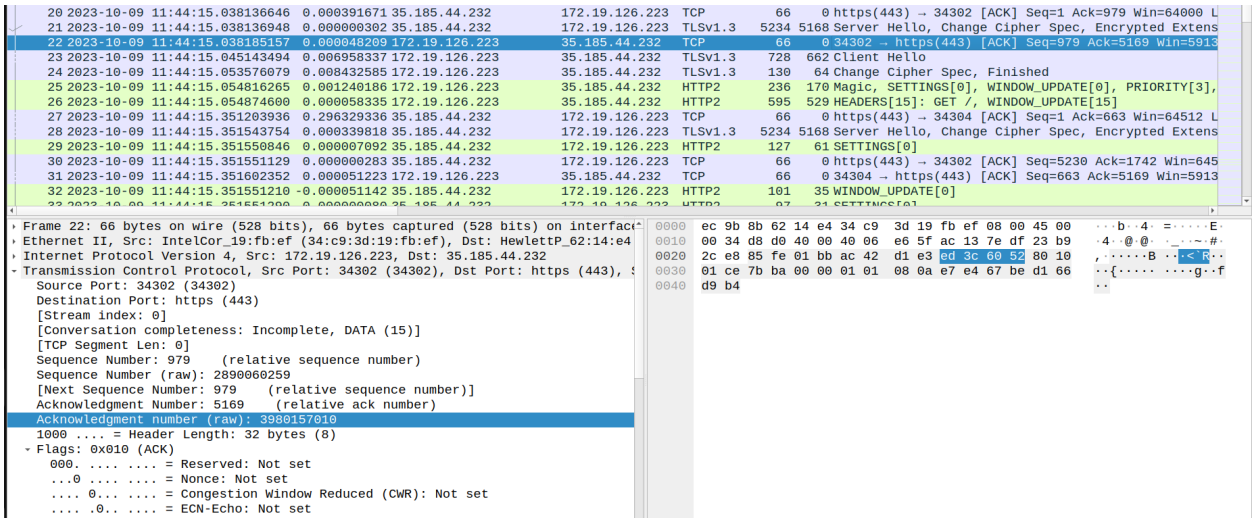


In response to this server has sent an ACK TCP segment and along with this server has also sent the ‘Server hello’ TLS packet which confirms that the TLS handshake is complete and the HTTPS connection is established. The sequence number of the TCP segment initiating this is 3980151842 and the ack no is 2890060259 (snapshot below)

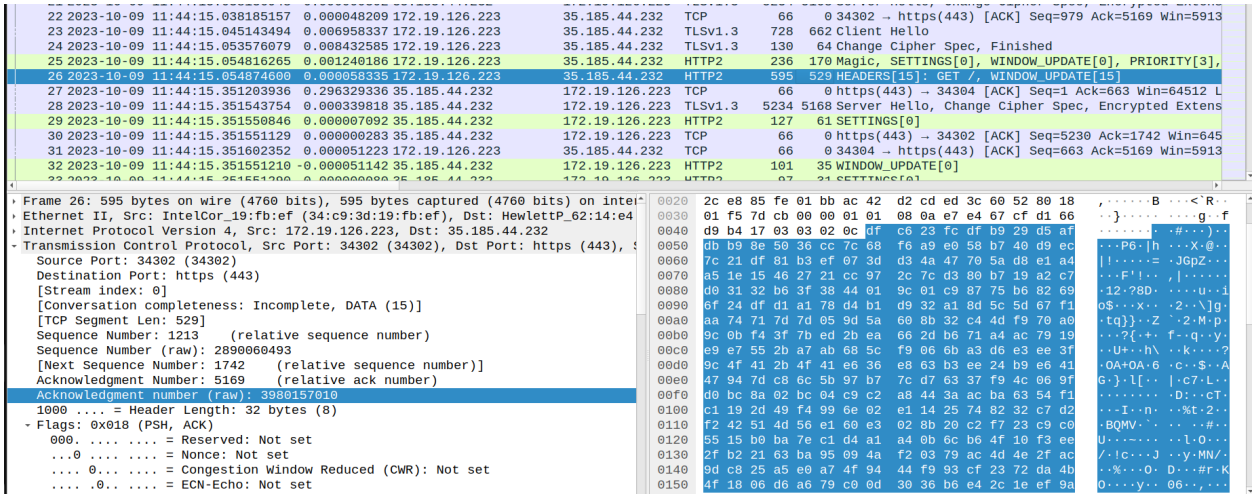


Notice that the Ack no (2980060259) of the TCP segment of the ‘Server hello’ TLS packet sent from the server is 978 more than the seq no of the TCP segment of the ‘Client Hello’ TLS packet (2980059281). This 978 is nothing but the payload of the TLS ‘Client Hello’ packet. Hence the server is saying that it has acked this payload at this stage.

The Get request is sent in the HTTPS connection established in the TLS handshake. The TCP segment initiating this TLS handshake has sequence no - 2980039281



This snapshot shows the ACK TCP segment sent from client to server after the TLS handshake is complete. Its ack no is 3980157010

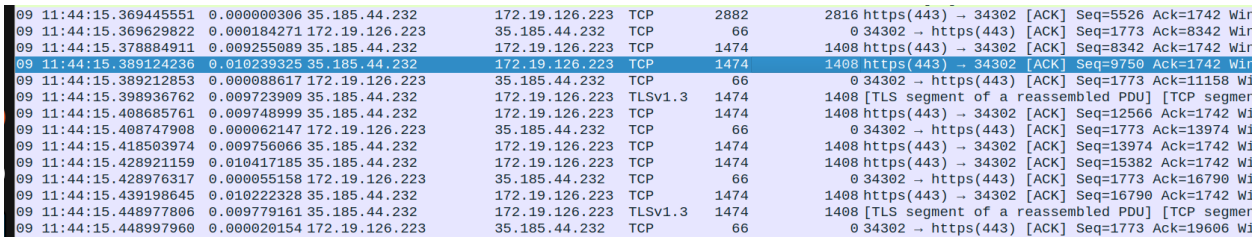


This snapshot shows the HTTP2 GET request packet. See that its TCP ack no is the same. And its TCP sequence number is 2980069493

The website contains many images along with hypertext and because of this, there are many TCP segments coming from the server which has the website payload. Hence, the whole payload is not coming in a single TCP segment.

Ans 6

Since, it's a GET request the client is receiving data from the server. Hence the RTT, in this case, is the difference between the time when the server has sent the payload and the time when client has sent the ACK segment.



09	11:44:15.369445551	0.000000306	35.185.44.232	172.19.126.223	TCP	2882	2816	https(443) → 34302 [ACK] Seq=5526 Ack=1742 Wir
09	11:44:15.369629822	0.000184271	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=8342 Wir
09	11:44:15.378884911	0.009255089	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=8342 Ack=1742 Wir
09	11:44:15.389124236	0.010239325	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=9750 Ack=1742 Wir
09	11:44:15.389212853	0.000088617	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=11158 Wi
09	11:44:15.398936762	0.009723909	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408	[TLS segment of a reassembled PDU] [TCP segmer
09	11:44:15.408685761	0.009748999	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=12566 Ack=1742 Wi
09	11:44:15.408747908	0.000062147	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=13974 Wi
09	11:44:15.418503974	0.009756066	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=13974 Ack=1742 Wi
09	11:44:15.428921159	0.010417185	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=15382 Ack=1742 Wi
09	11:44:15.428976317	0.000055158	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=16790 Wi
09	11:44:15.439198645	0.010222328	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=16790 Ack=1742 Wi
09	11:44:15.448977806	0.009779161	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408	[TLS segment of a reassembled PDU] [TCP segmer
09	11:44:15.448997960	0.000020154	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=19606 Wi

Time when first payload is received = 11:44:15.369445551

Time when first ack is sent = 11:44:15.369629822

RTT 1 = 0.00018 sec

Time when second payload is received = 11:44:15.378884911

Time when second ack is sent = 11:44:15.389212853

RTT 2 = 0.1033 sec

Alpha = 0.125

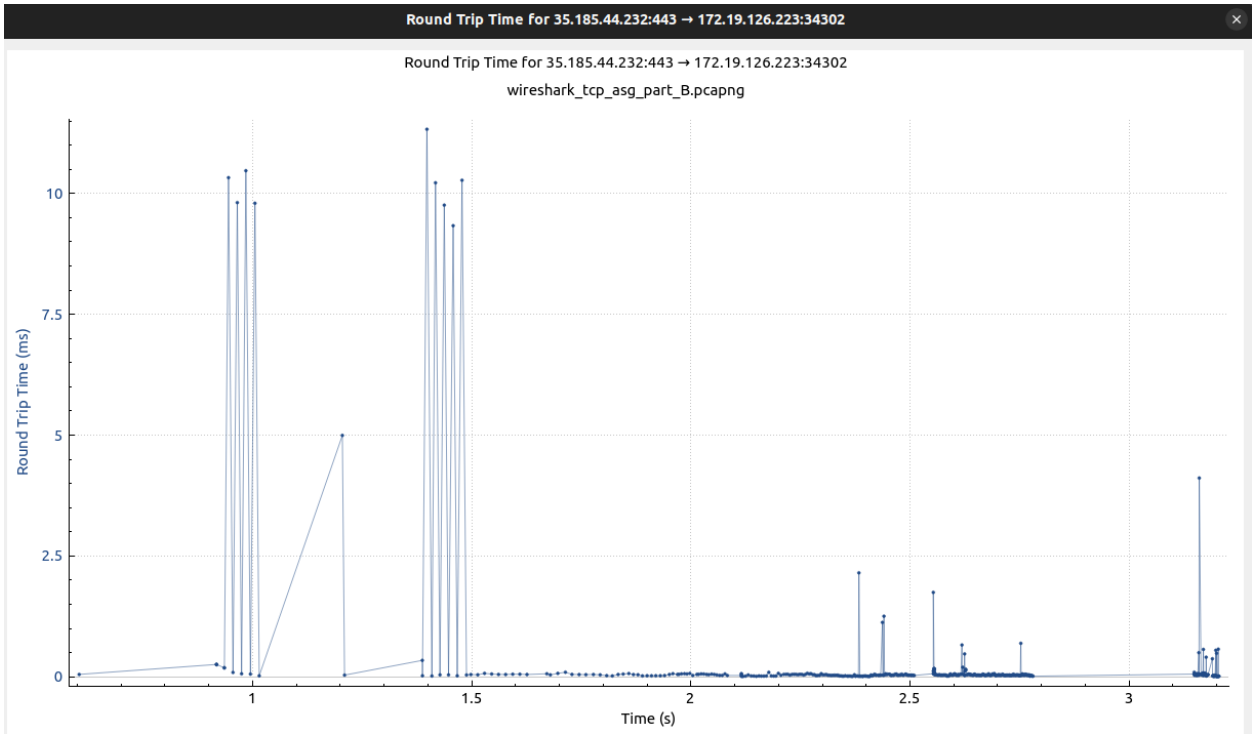
Initial Estimated RTT = 0.00018 sec

New Estimated RTT = (1 - 0.125)*0.00018 + 0.125*0.1033

= 0.0001575 + 0.125*0.1033

= 0.01307 sec

RTT Graph: -



Ans 7

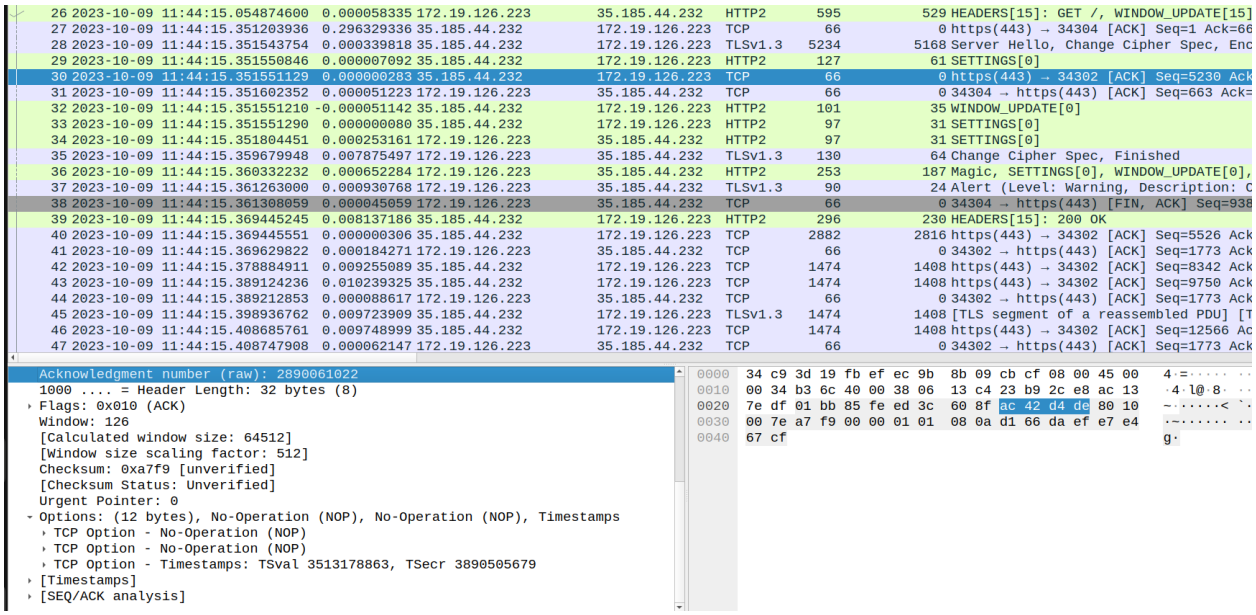
26	2023-10-09 11:44:15.054874600	0.000058335	172.19.126.223	35.185.44.232	HTTP2	595	529	HEADERS[15]: GET /, WINDOW_UPDATE[15]
27	2023-10-09 11:44:15.351203936	0.296329336	35.185.44.232	172.19.126.223	TCP	66	0	https(443) → 34304 [ACK] Seq=1 Ack=663 Win
28	2023-10-09 11:44:15.351543754	0.000339818	35.185.44.232	172.19.126.223	TLSv1.3	5234	5168	Server Hello, Change Cipher Spec, Encrypte
29	2023-10-09 11:44:15.351550846	0.000007092	35.185.44.232	172.19.126.223	HTTP2	127	61	SETTINGS[0]
30	2023-10-09 11:44:15.351551129	0.000000283	35.185.44.232	172.19.126.223	TCP	66	0	https(443) → 34302 [ACK] Seq=5230 Ack=1742
31	2023-10-09 11:44:15.351602352	0.000051223	172.19.126.223	35.185.44.232	TCP	66	0	34304 → https(443) [ACK] Seq=663 Ack=5169
32	2023-10-09 11:44:15.351551210	-0.000051142	35.185.44.232	172.19.126.223	HTTP2	101	35	WINDOW_UPDATE[0]
33	2023-10-09 11:44:15.351551290	0.000000080	35.185.44.232	172.19.126.223	HTTP2	97	31	SETTINGS[0]
34	2023-10-09 11:44:15.351804451	0.000253161	172.19.126.223	35.185.44.232	HTTP2	97	31	SETTINGS[0]
35	2023-10-09 11:44:15.359679948	0.007875497	172.19.126.223	35.185.44.232	TLSv1.3	130	64	Change Cipher Spec, Finished
36	2023-10-09 11:44:15.360332232	0.000652284	172.19.126.223	35.185.44.232	HTTP2	253	187	Magic, SETTINGS[0], WINDOW_UPDATE[0], PRIO
37	2023-10-09 11:44:15.361263000	0.000930768	172.19.126.223	35.185.44.232	TLSv1.3	90	24	Alert (Level: Warning, Description: Close
38	2023-10-09 11:44:15.361308059	0.000045059	172.19.126.223	35.185.44.232	TCP	66	0	34304 → https(443) [FIN, ACK] Seq=938 Ack=
39	2023-10-09 11:44:15.369445245	0.008137186	35.185.44.232	172.19.126.223	HTTP2	296	230	HEADERS[15]: 200 OK
40	2023-10-09 11:44:15.369445551	0.000000306	35.185.44.232	172.19.126.223	TCP	2882	2816	https(443) → 34302 [ACK] Seq=5526 Ack=1742
41	2023-10-09 11:44:15.369629822	0.000184271	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=8342
42	2023-10-09 11:44:15.378884911	0.009255089	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=8342 Ack=1742
43	2023-10-09 11:44:15.389124236	0.010239325	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=9750 Ack=1742
44	2023-10-09 11:44:15.389212853	0.000088617	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=1115
45	2023-10-09 11:44:15.398936762	0.009723909	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408	[TLS segment of a reassembled PDU] [TCP se
46	2023-10-09 11:44:15.408685761	0.009748999	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=12566 Ack=174
47	2023-10-09 11:44:15.408747908	0.000062147	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=1397
48	2023-10-09 11:44:15.418503974	0.009756066	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=13974 Ack=174
49	2023-10-09 11:44:15.428921159	0.010417185	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=15382 Ack=174
50	2023-10-09 11:44:15.428976317	0.000055158	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=1679
57	2023-10-09 11:44:15.439198645	0.010222328	35.185.44.232	172.19.126.223	TCP	1474	1408	https(443) → 34302 [ACK] Seq=16790 Ack=174
59	2023-10-09 11:44:15.448877806	0.009779161	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408	[TLS segment of a reassembled PDU] [TCP se
60	2023-10-09 11:44:15.448997960	0.000020154	172.19.126.223	35.185.44.232	TCP	66	0	34302 → https(443) [ACK] Seq=1773 Ack=1960

(packets highlighted in blue)

The only payload sent from client to server is the HTTPS GET request after the TCP and TLS handshake is complete and the HTTPS connection has been established. The length of the TCP segment in the HTTP2 get request is 529 bytes with the whole packet being 595 bytes

The TCP segments being sent from client to server after this are just the ACK segments with a total length of 66 bytes (corresponding to the header length) and their TCP segment length being 0

Ans 8

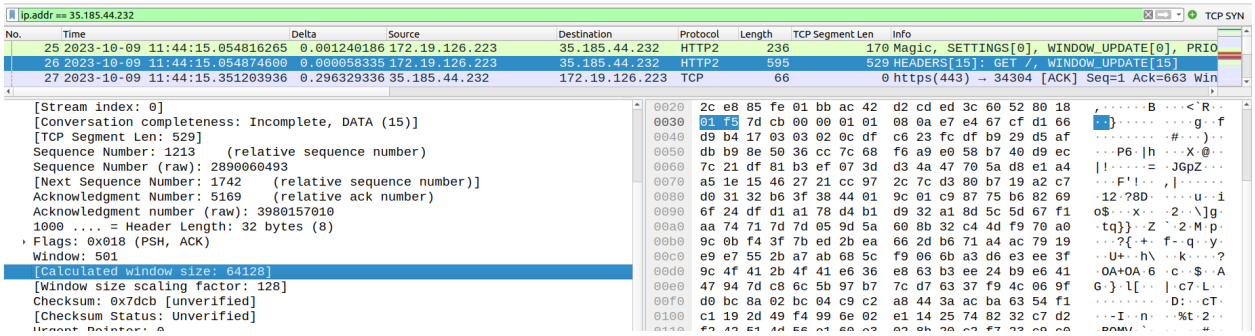


Window size = (window value) * (scaling factor value)

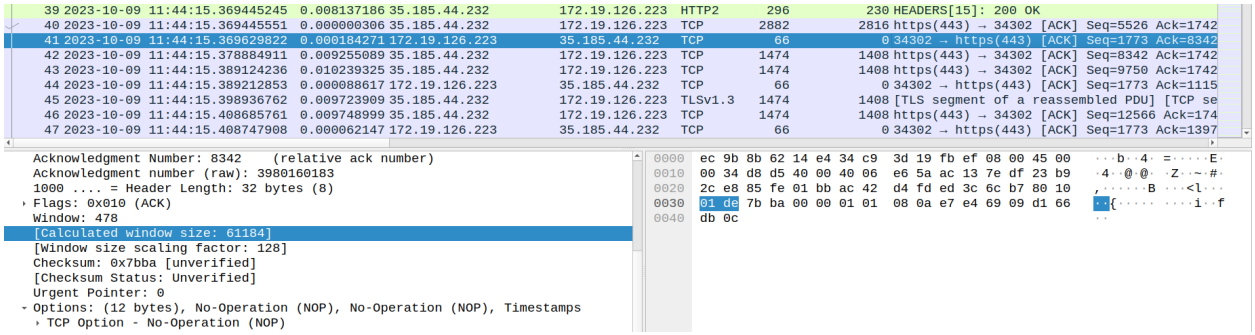
This is the first response from server for the GET request. The window size advertised to the client is 64512 bytes

The window size advertised by all the subsequent TCP segments received from the server is 64512 bytes. This is not changing because the client is not sending anything to the server as the client has sent a GET request.

Ans 9



The window size advertised by the client at the time of sending the GET request is 64128 bytes



In the next TCP ACK segment sent from the client, the window size has been decreased to 61184 bytes.

The subsequent 3 advertisements are 63104 bytes.

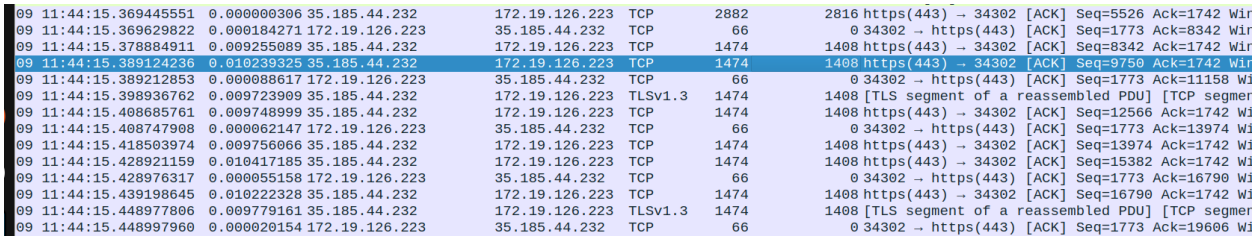
There is variation in the window size on the client side. This is because the client is receiving the website data from the server and is processing it in order to show it on the browser. This process can take time and the buffer space would dynamically change.

Ans 10

There are no re-transmitted segments. I checked the ACK segments and looked for duplicate ack values but found none. This indicates that no packet has been re-requested.

Ans 11

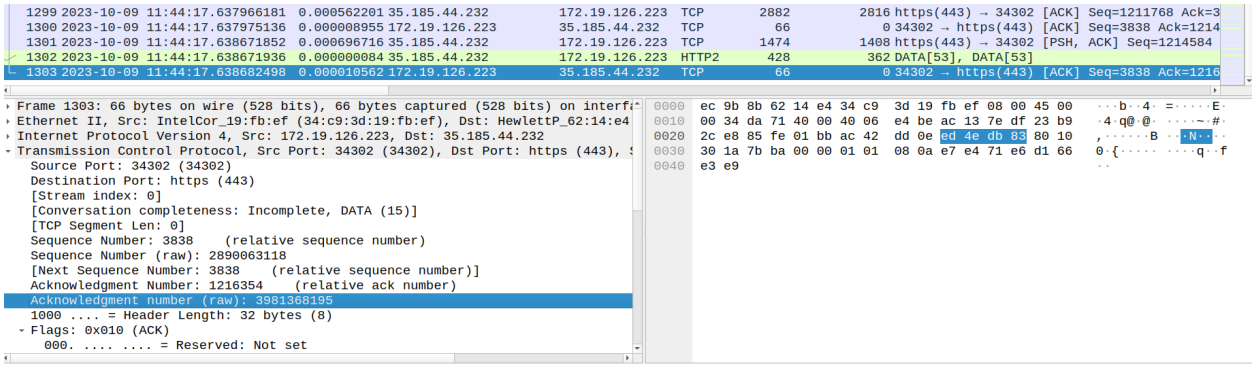
In this case, since we have made a GET request, the server is sending us the data and the client is sending ACK segments to the server. On looking at the ACK segments being sent from the client to the server: -



09	11:44:15.369445551	0.000000306	35.185.44.232	172.19.126.223	TCP	2882	2816 https(443) → 34302 [ACK] Seq=5526 Ack=1742 Wir
09	11:44:15.369629822	0.000184271	172.19.126.223	35.185.44.232	TCP	66	0 34302 → https(443) [ACK] Seq=1773 Ack=8342 Wir
09	11:44:15.378884911	0.009255089	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=8342 Ack=1742 Wir
09	11:44:15.389124236	0.010239325	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=9750 Ack=1742 Wir
09	11:44:15.389212853	0.000088617	172.19.126.223	35.185.44.232	TCP	66	0 34302 → https(443) [ACK] Seq=1773 Ack=11158 Wi
09	11:44:15.398936762	0.009723909	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408 [TLS segment of a reassembled PDU] [TCP segmer
09	11:44:15.408685761	0.009748999	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=12566 Ack=1742 Wi
09	11:44:15.408747908	0.000062147	172.19.126.223	35.185.44.232	TCP	66	0 34302 → https(443) [ACK] Seq=1773 Ack=13974 Wi
09	11:44:15.418503974	0.009756066	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=13974 Ack=1742 Wi
09	11:44:15.428921159	0.010417185	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=15382 Ack=1742 Wi
09	11:44:15.428976317	0.000055158	172.19.126.223	35.185.44.232	TCP	66	0 34302 → https(443) [ACK] Seq=1773 Ack=16790 Wi
09	11:44:15.439198645	0.010222328	35.185.44.232	172.19.126.223	TCP	1474	1408 https(443) → 34302 [ACK] Seq=16790 Ack=1742 Wi
09	11:44:15.448977806	0.009779161	35.185.44.232	172.19.126.223	TLSv1.3	1474	1408 [TLS segment of a reassembled PDU] [TCP segmer
09	11:44:15.448997960	0.000020154	172.19.126.223	35.185.44.232	TCP	66	0 34302 → https(443) [ACK] Seq=1773 Ack=19606 Wi

Here we can see that the server sent TCP segment of length 2816 bytes which got acked by the client. After this, the server is sending two segments of length 1408 bytes each which are being acked commutatively (1 ack for 2 segments). Hence, the client is typically acking 2816 bytes.

Ans 12

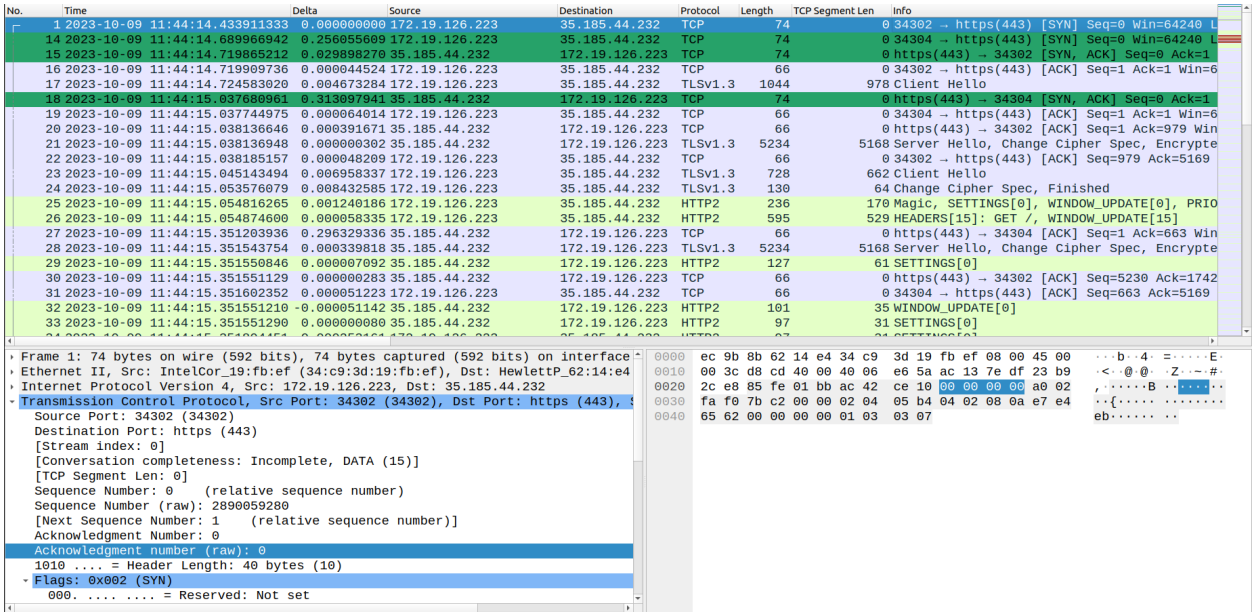


The last ACK sent from client to server has been sent,

At time - 11:44:17.638682498

With ACK value = 1216354

This means that 1216354 bytes have been received from server in response to the GET request to view the cse.iith.ac.in/ website.



The TCP SYN was sent at - 11:44:14.433911333

Total time taken to load the website = 3.2047 sec

Therefore throughput = 1216354 / 3.2047 = 379.553 kbps