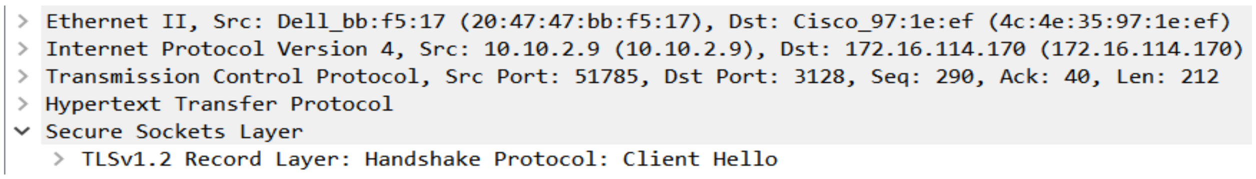
Submitted by **Hareesh Reddi** *Roll Number:* **150101051 (Coursera Videos)**

Traces Link: https://drive.google.com/open?id=1zv2mSDiTi_gFswT7EizOlJT5CD9BysON

Question1:



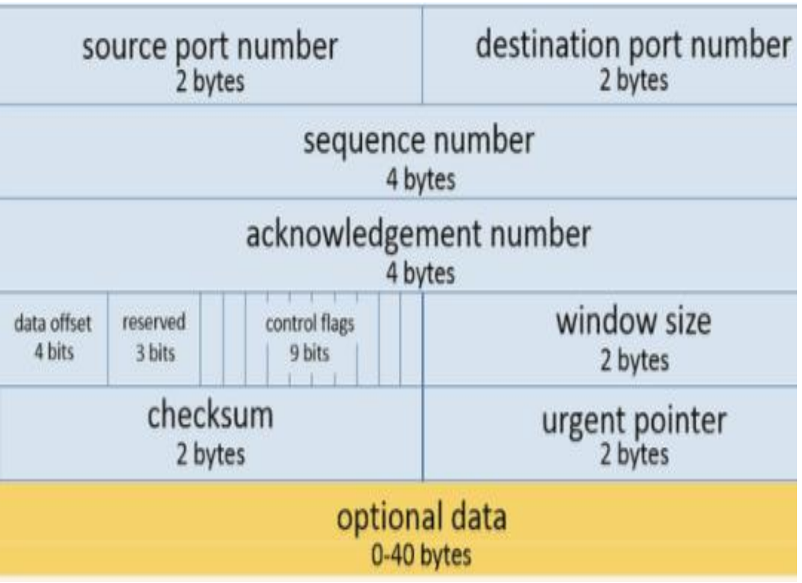
```
> Ethernet II, Src: Dell_bb:f5:17 (20:47:47:bb:f5:17), Dst: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
> Internet Protocol Version 4, Src: 10.10.2.9 (10.10.2.9), Dst: 172.16.114.170 (172.16.114.170)
> Transmission Control Protocol, Src Port: 51785, Dst Port: 3128, Seq: 290, Ack: 40, Len: 212
> Hypertext Transfer Protocol
v Secure Sockets Layer
  > TLSv1.2 Record Layer: Handshake Protocol: Client Hello
```

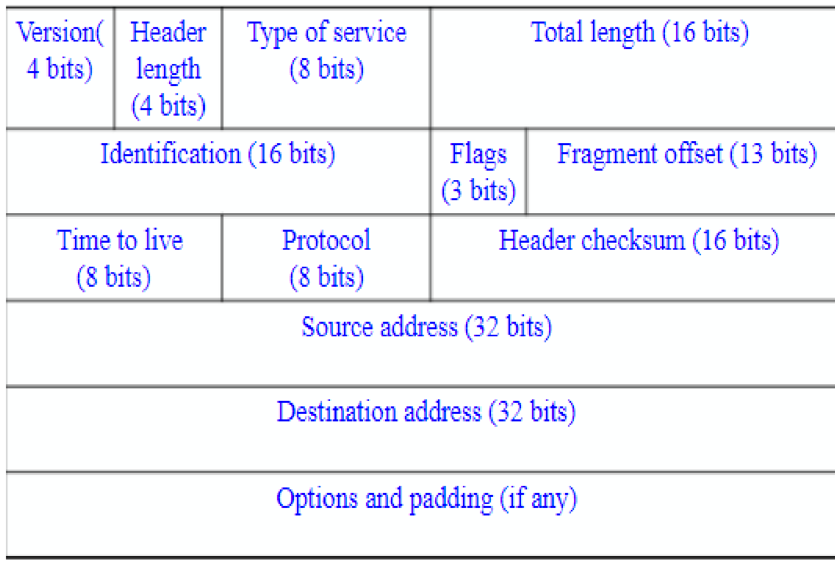
Various protocols used by the functions like Play, Download of application Coursera Video are

1) Transport Layer: **TCP 2)** Network Layer: **IPV4 3)** Application Layer: **HTTP**

4) Secure Socket/Session Layer: **TLSv 1.2 Protocol** **5) Physical Layer: Ethernet II Protocol**

1&2) Figures on the **left side, right side** indicates frame format of **TCP** and **IPV4** respectively.





I described all the fields of both the protocol Frame Formats in **second answer**.

3) HTTP header format: HTTP messages consist of requests from client to server and responses from server to client.

HTTP-message = **<*Request*> | <*Response*>; HTTP/1.1 messages**

Request and Response messages use the generic message format of RFC 822 for transferring entities (the payload of the message). This generic message format consists of the following four items.

- **Start-line**: Request Line or Status Line.
- **Zero or more header fields followed by CRLF**: The headers are as follows: General Header, Response Header, Request Header and Entity Header.
- **An empty line**: indicating the end of the header fields
- Optionally a **message-body**: It is used to carry the entity-body associated with the request or response.

4) Secure Session Layer(Record Layer) : This layer acts as an intermediate between transport layer and application layer. It deals with session and connection coordination. TLS 1.2 was defined in RFC 5246 in August 2008. **Packet Format**:

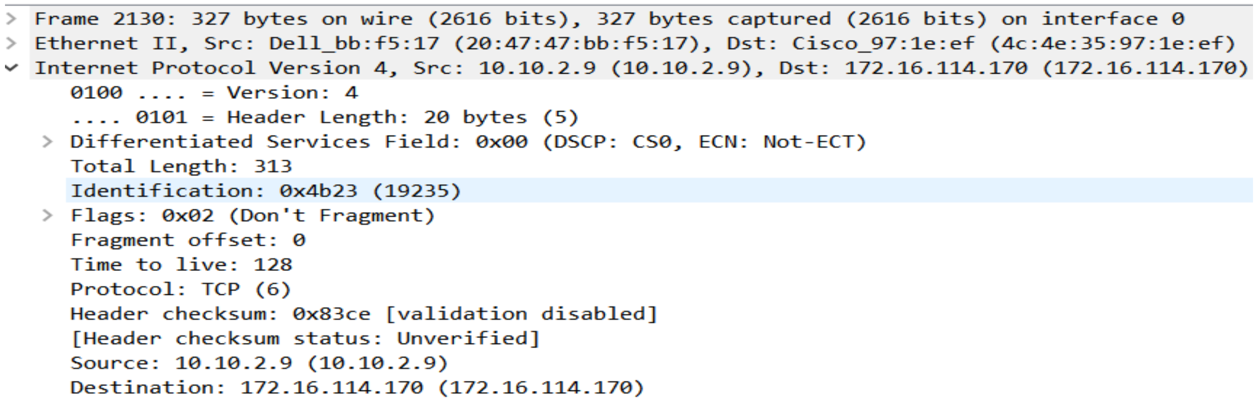
- **Content type**: The type field is identical to TLSCompressed.type.
- **Version**: The version field is identical to TLSCompressed.version.
- **Length**: The length (in bytes) of the following TLSCiphertext.fragment. The length MUST NOT exceed 2^14 + 2048.
- **Fragment**: The encrypted form of TLSCompressed.fragment, with the MAC.
- **Message Authentication Code (MAC)**: It is a one-way hash computed from a message and some secret data. It is difficult to forge without knowing the secret data. Its purpose is to detect if the message has been altered.

5) Ethernet II Protocol Frame Format:

DA	SA	Type	DSAP	SSAP	Ctrl	Data	FCS
6 Bytes	6 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	>46 Bytes	4 Bytes

The fields are as follows: 1) **DA** [Destination MAC Address] 2) **SA** [Source MAC Address] 3) **Type** [0x8870 (Ether type)] 4) **DSAP** [802.2 Destination Service Access Point] 5) **SSAP** [802.2 Source Service Access Point] 6) **Ctrl** [802.2 Control Field] 7) **Data** [Protocol Data] 8) **FCS** [Frame Checksum]

Question 2:

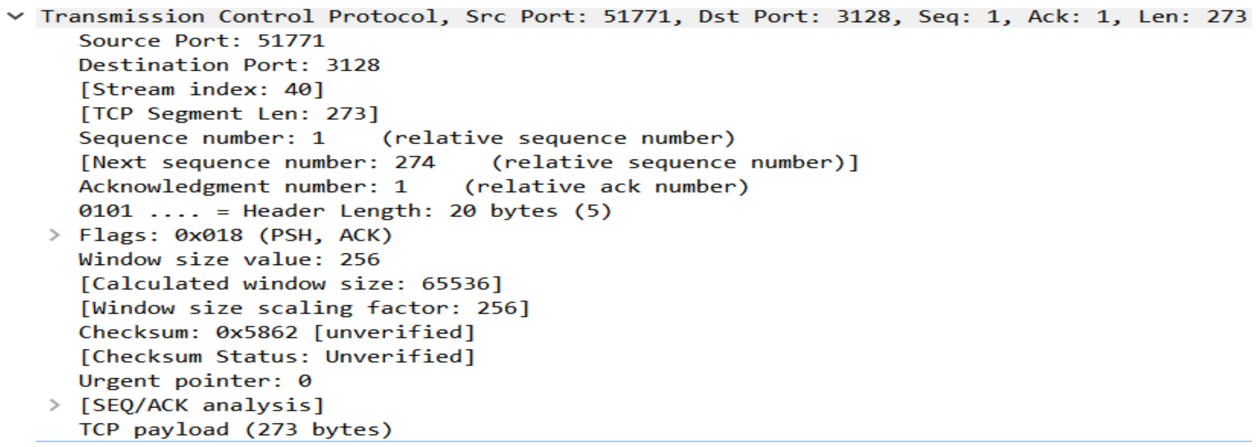


```
> Frame 2130: 327 bytes on wire (2616 bits), 327 bytes captured (2616 bits) on interface 0
> Ethernet II, Src: Dell_bb:f5:17 (20:47:47:bb:f5:17), Dst: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
v Internet Protocol Version 4, Src: 10.10.2.9 (10.10.2.9), Dst: 172.16.114.170 (172.16.114.170)
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 313
     Identification: 0x4b23 (19235)
   > Flags: 0x02 (Don't Fragment)
     Fragment offset: 0
     Time to live: 128
     Protocol: TCP (6)
     Header checksum: 0x83ce [validation disabled]
     [Header checksum status: Unverified]
     Source: 10.10.2.9 (10.10.2.9)
     Destination: 172.16.114.170 (172.16.114.170)
```

1) Frame 2130: The frame number of the trace I took for observation is 2130. The frame protocol is **not a real protocol** itself, but used by Wireshark as a base for all the protocols on top of it. We also have information about the bytes used by the frame (327 bytes)
2) Ethernet II: **Src: Dell_bb:f5:17 (20:47:47:bb:f5:17)** : Hardware Mac address of my PC (Source).
Dst: Cisco_97:1e:ef (4c:4e:35:97:1e:ef) : Hardware mac address of destination.
3) Internet Protocol Version 4: **Src : 10.10.2.9** – This is my computer IP address(Source IP Address). **Dst: 172.16.114.170** – This destination computer IP address. Here, I used the proxy address for internet connection (Destination IP Address). These IP addresses help us to uniquely identify both machines on the network. **Version: 4**: field indicates the format of the internet header. **Header length: 20 bytes**: Number of 32-bit words in the TCP header. **Differentiated Services Field** (DSF): **0x00**: Indicates particular Quality of Service needs from the network, the DSF defines the way routers should queue packets while they are waiting to be forwarded. **Total Length: 313:** is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets**.** **Flags: 3 bits**, Various Control Flags: **Bit 0**: reserved, must be zero

Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

Fragment Offset: 0 bits: This field indicates where in the datagram this fragment belongs. **Time to live: 128:** This field indicates the maximum time the datagram is allowed to remain in the internet system. Field called **Protocol** (Same as protocol number): **TCP (6)**: To identify the next level protocol used. **Header checksum: 0x83ce**: A checksum on the header only and its status is unverified.

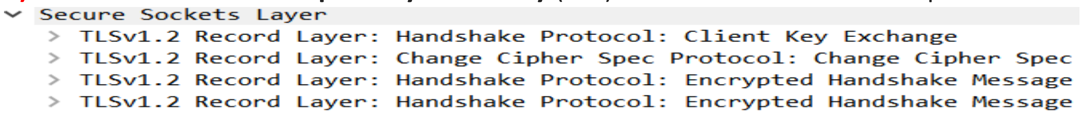


```
⌄ Transmission Control Protocol, Src Port: 51771, Dst Port: 3128, Seq: 1, Ack: 1, Len: 273
    Source Port: 51771
    Destination Port: 3128
    [Stream index: 40]
    [TCP Segment Len: 273]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 274    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x5862 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
    TCP payload (273 bytes)
```

4) Transmission Control Protocol:

- **Src Port: 51711**: My computer port in this case, which is sending packets.
- **Dst Port: 3128**: This is receiving/Destination port which is receiving packets.
- **TCP Segment Length: 273**: This is TCP packet segment length.
- **Sequence number: 1**: If the SYN flag is set (1), then this is the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1.
- **Acknowledgment number: 1**: The ACK flag is set to 1 which means the value of this field is the next sequence number that the sender is expecting.
- **Flags (9 bits):** It contains 9 1-bit flags. **ACK** (1 bit)**:1**: indicates that the Acknowledgment field is significant. All packets after the initial SYN packet sent by the client should have this flag set. **PSH** (1 bit)**:1**: **Push function**. Asks to push the buffered data to the receiving application. Remaining seven flags i.e. NS, CWR, ECE, URG, RST, SYN, FIN have value 0.
- **Window size value: 256**: This is the space for the incoming data.
- **Checksum (16 bits):0x5862**: The 16-bit checksum field is used for error checking of the header and data.
- **Urgent pointer (16 bits)**: **0**: If the URG flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte.

5) TLSv1.2 Protocol: Transport Layer Security (TLS): Some of the fields of the protocol can be seen.

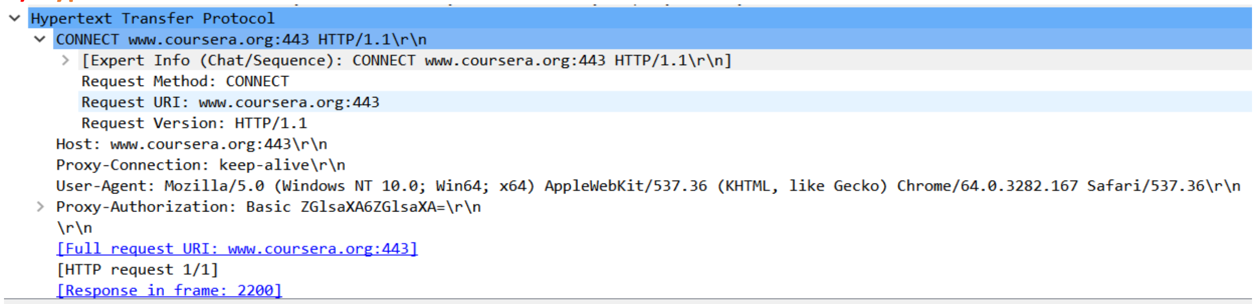


```
⌄ Secure Sockets Layer
  > TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
  > TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  > TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
  > TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
```

Some other messages includes **Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done** between **client** (10.10.2.9) and **server** (172.16.114.170) and TLS is implemented on two levels- The

TLS Record protocol and the TLS Handshake protocol to ensure **security, efficiency and extensibility**.

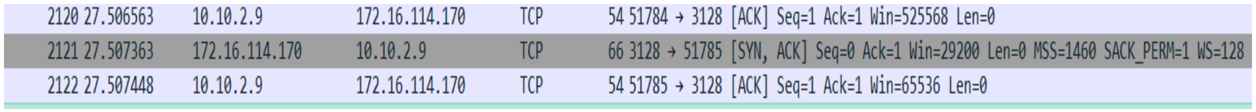
6) Hypertext Transfer Protocol:



```
v Hypertext Transfer Protocol
  v CONNECT www.coursera.org:443 HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): CONNECT www.coursera.org:443 HTTP/1.1\r\n]
      Request Method: CONNECT
      Request URI: www.coursera.org:443
      Request Version: HTTP/1.1
    Host: www.coursera.org:443\r\n
    Proxy-Connection: keep-alive\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.167 Safari/537.36\r\n
  > Proxy-Authorization: Basic ZGlsaXA6ZGlsaXA=\r\n
    \r\n
    [Full request URI: www.coursera.org:443]
    [HTTP request 1/1]
    [Response in frame: 2200]
```

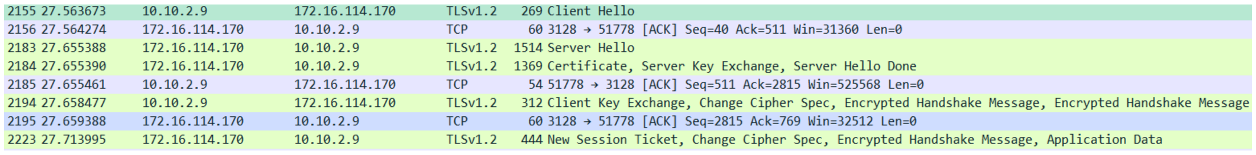
- **CONNECT www.coursera.org:443 HTTP/1.1\r\n**: This is start line, indicating it is request packet.
- **Request Method**: The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.
- **Host:** www.coursera.org:443\r\n: This tells hostname to which requesting the packet.
- **User-Agent**: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.167 Safari/537.36\r\n: This is part of requesting header. Tells about host information about the browser and operating system**.**

Question3: **a) 3-Way TCP Handshake Message sequence**: Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. Establishing a normal TCP connection requires three separate steps: **1) SYN**: The client sending a SYN to the server performs the active open. The **client** sets the segment's sequence number to a random value **A**. **2) SYN-ACK**: In response, the **server replies with a SYN-ACK**. The acknowledgment number is set to one more than the received sequence number i.e. **A+1**, and the sequence number that the server chooses for the packet is another random number, **B**. **3) ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. **A+1**, and the acknowledgement number is set to one more than the received sequence number i.e. **B+1**.



```
2120 27.506563   10.10.2.9        172.16.114.170   TCP    54 51784 → 3128 [ACK] Seq=1 Ack=1 Win=525568 Len=0
2121 27.507363   172.16.114.170   10.10.2.9        TCP    66 3128 → 51785 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
2122 27.507448   10.10.2.9        172.16.114.170   TCP    54 51785 → 3128 [ACK] Seq=1 Ack=1 Win=65536 Len=0
```

b) TLS Handshaking Message Sequence observed:

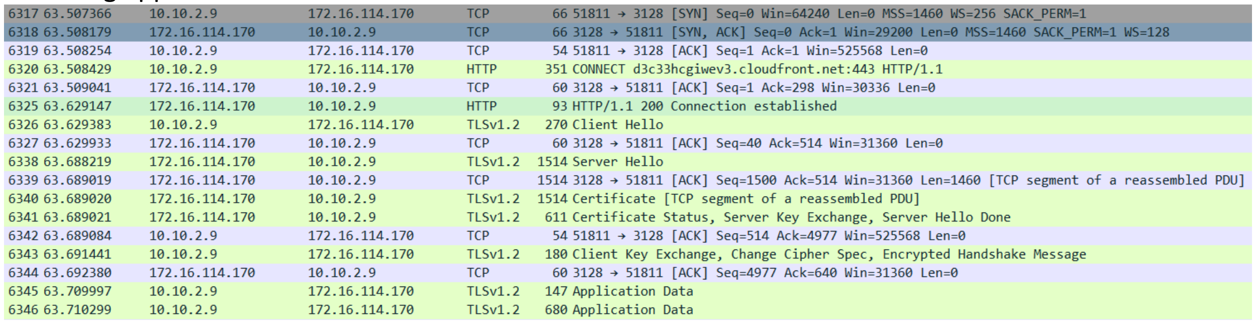


```
2155 27.563673  10.10.2.9       172.16.114.170  TLSv1.2   269 Client Hello
2156 27.564274  172.16.114.170  10.10.2.9       TCP        60 3128 → 51778 [ACK] Seq=40 Ack=511 Win=31360 Len=0
2183 27.655388  172.16.114.170  10.10.2.9       TLSv1.2  1514 Server Hello
2184 27.655390  172.16.114.170  10.10.2.9       TLSv1.2  1369 Certificate, Server Key Exchange, Server Hello Done
2185 27.655461  10.10.2.9       172.16.114.170  TCP        54 51778 → 3128 [ACK] Seq=511 Ack=2815 Win=525568 Len=0
2194 27.658477  10.10.2.9       172.16.114.170  TLSv1.2   312 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message, Encrypted Handshake Message
2195 27.659388  172.16.114.170  10.10.2.9       TCP        60 3128 → 51778 [ACK] Seq=2815 Ack=769 Win=32512 Len=0
2223 27.713995  172.16.114.170  10.10.2.9       TLSv1.2   444 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message, Application Data
```

The record encapsulates a "control" protocol (the handshake messaging protocol) when the TLS connection starts. This protocol is used to exchange all the information required by both sides for the exchange of the actual application data by TLS and to negotiate the **secure attributes of a session**. It defines the messages formatting or containing this information and the order of their exchange. These may vary according to the demands of the client and server. The TLS Handshake protocol allows **authenticated communication** to commence between the server and client. This protocol allows the client and server to speak the same language, allowing them to agree upon an **encryption algorithm and encryption keys** before the selected application protocol begins to send data. We can see **various**

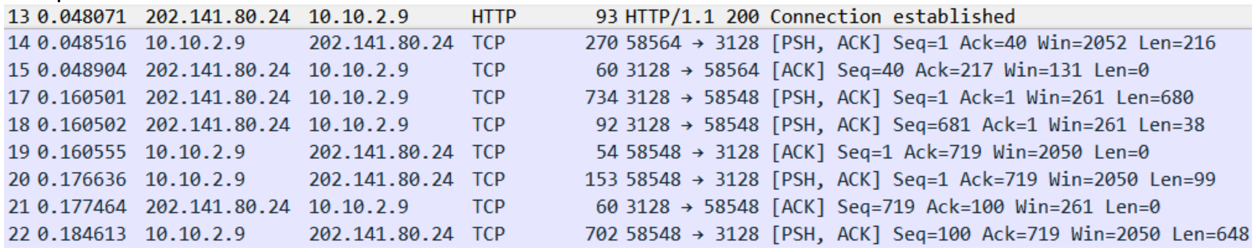
messages i.e. Client Hello, Server Hello, Certificate, Server Key Exchange, Server Hello Done, Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message b/n **client** (10.10.2.9) and **server** (172.16.114.170). As **Coursera Videos** was not mentioned any particular function, I chose **Play, Running and Download functions** for carrying the observations, as Upload is only available to Professors.

a) **Play & Running Functions**: This function shows a sequence of TCP connection establishment through 3-way handshake, then a HTTP request to the host **d3c33hcgiwev3.cloudfront.net**:443 HTTP/1.1. This got response from TCP after a TLS handshake to ensure security. Work of **Play** button finishes here where as when video is **running** it gives response through continuously getting TCP data transfer and obtaining application data.



```
6317 63.507366   10.10.2.9        172.16.114.170   TCP        66 51811 → 3128 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
6318 63.508179   172.16.114.170   10.10.2.9        TCP        66 3128 → 51811 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
6319 63.508254   10.10.2.9        172.16.114.170   TCP        54 51811 → 3128 [ACK] Seq=1 Ack=1 Win=525568 Len=0
6320 63.508429   10.10.2.9        172.16.114.170   HTTP      351 CONNECT d3c33hcgiwev3.cloudfront.net:443 HTTP/1.1
6321 63.509041   172.16.114.170   10.10.2.9        TCP        60 3128 → 51811 [ACK] Seq=1 Ack=298 Win=30336 Len=0
6325 63.629147   172.16.114.170   10.10.2.9        HTTP       93 HTTP/1.1 200 Connection established
6326 63.629383   10.10.2.9        172.16.114.170   TLSv1.2   270 Client Hello
6327 63.629933   172.16.114.170   10.10.2.9        TCP        60 3128 → 51811 [ACK] Seq=40 Ack=514 Win=31360 Len=0
6338 63.688219   172.16.114.170   10.10.2.9        TLSv1.2  1514 Server Hello
6339 63.689019   172.16.114.170   10.10.2.9        TCP      1514 3128 → 51811 [ACK] Seq=1500 Ack=514 Win=31360 Len=1460 [TCP segment of a reassembled PDU]
6340 63.689020   172.16.114.170   10.10.2.9        TLSv1.2  1514 Certificate [TCP segment of a reassembled PDU]
6341 63.689021   172.16.114.170   10.10.2.9        TLSv1.2   611 Certificate Status, Server Key Exchange, Server Hello Done
6342 63.689084   10.10.2.9        172.16.114.170   TCP        54 51811 → 3128 [ACK] Seq=514 Ack=4977 Win=525568 Len=0
6343 63.691441   10.10.2.9        172.16.114.170   TLSv1.2   180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
6344 63.692380   172.16.114.170   10.10.2.9        TCP        60 3128 → 51811 [ACK] Seq=4977 Ack=640 Win=31360 Len=0
6345 63.709997   10.10.2.9        172.16.114.170   TLSv1.2   147 Application Data
6346 63.710299   10.10.2.9        172.16.114.170   TLSv1.2   680 Application Data
```

b) **Download Function**: After clicking on the download button, initially connection is established through 3-way handshake, then a HTTP request to the host d3njjcbhbojbot.cloudfront.net:443 HTTP/1.1. Then followed by TLS handshake. Finally, **TCP segments** started being transferred from server to our computer.



```
13 0.048071  202.141.80.24  10.10.2.9      HTTP     93 HTTP/1.1 200 Connection established
14 0.048516  10.10.2.9      202.141.80.24  TCP     270 58564 → 3128 [PSH, ACK] Seq=1 Ack=40 Win=2052 Len=216
15 0.048904  202.141.80.24  10.10.2.9      TCP      60 3128 → 58564 [ACK] Seq=40 Ack=217 Win=131 Len=0
17 0.160501  202.141.80.24  10.10.2.9      TCP     734 3128 → 58548 [PSH, ACK] Seq=1 Ack=1 Win=261 Len=680
18 0.160502  202.141.80.24  10.10.2.9      TCP      92 3128 → 58548 [PSH, ACK] Seq=681 Ack=1 Win=261 Len=38
19 0.160555  10.10.2.9      202.141.80.24  TCP      54 58548 → 3128 [ACK] Seq=1 Ack=719 Win=2050 Len=0
20 0.176636  10.10.2.9      202.141.80.24  TCP     153 58548 → 3128 [PSH, ACK] Seq=1 Ack=719 Win=2050 Len=99
21 0.177464  202.141.80.24  10.10.2.9      TCP      60 3128 → 58548 [ACK] Seq=719 Ack=100 Win=261 Len=0
22 0.184613  10.10.2.9      202.141.80.24  TCP     702 58548 → 3128 [PSH, ACK] Seq=100 Ack=719 Win=2050 Len=648
```

Question4: Protocols used by our network to Play, Download the Coursera Videos are explained below with their functions. The protocols used were **TCP, HTTP and TLSv1.2**.

a) **TCP**: Its responsibility includes end-to-end message transfer independent of the underlying network and structure of user data, along with error control, segmentation, **flow control**, and helps to minimize **traffic congestion** control. It is a connection-oriented protocol that addresses numerous **reliability** issues in providing a reliable byte stream: data arrives in-order, data has minimal error (i.e., correctness), duplicate data is discarded and lost or discarded packets are resent. TCP is optimized for **accurate delivery** rather than timely delivery, as correct sequence of buffer to be fetched. TCP's bandwidth probing and congestion control will attempt to use all of the available bandwidth between server and client.

b) **HTTP**: TCP works in the **Transport layer** while HTTP works in **Application layer** of TCP/IP model. TCP is in charge of setting up a reliable connection between two machines and HTTP uses this connection to transfer data between the web servers and the client in the communication process, we can say connection is fundamentally out of scope of HTTP as it is controlled at Transport Layer. HTTP is a **stateless protocol** though not sessionless, meaning that the server does not keep any data (state) between two requests/each request message can be understood in isolation. HTTP follows a classical

client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is an **extensible protocol** that is easy to use. The client-server structure, combined with the ability to simply add headers, allows HTTP to **advance** along with the extended capabilities of the Web. Usually, HTTP responses are **buffered** rather than streamed. **HTTP 1.1** added supports for streaming through keep-alive header so data could be streamed. HTTP is Media independent i.e. any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content.

c) **TLSv1.2**: TLS was designed to operate on top of a reliable transport protocol such as TCP. Web servers use cookies to identify the web user. They are small piece of data stored into the web user's disk. TLS is used to protect **session cookies** on the rest of the sites from being intercepted to protect user accounts. The TLS protocol aims primarily to provide **privacy** and **data integrity** between two communicating computer applications. SSL and TLS are both cryptographic protocols utilizing X.509 certificates, public/private key encryption that provide **authentication** and **data encryption** between servers, machines and applications operating over a network. TLS provides verification of **identity** of server, which is as important as encryption. The goals of the TLS protocol are cryptographic **security**, **extensibility**, and relative **efficiency**. These goals are achieved through implementation of the TLS protocol on **two** levels: the TLS Record protocol and the TLS Handshake protocol. TLS allows the peers to negotiate **a shared secret key** without having to establish any prior knowledge of each other, and to do so over an unencrypted channel, Client-server applications use the TLS protocol to communicate across a network in a way designed to prevent eavesdropping and tampering.

Question5:

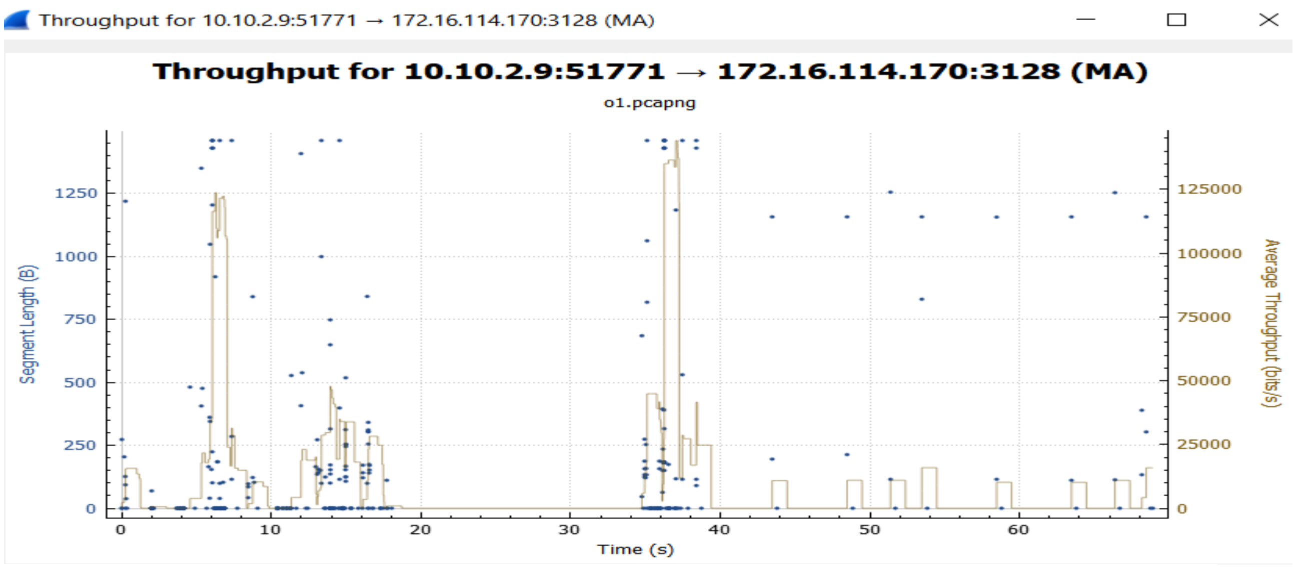
Playing Function:

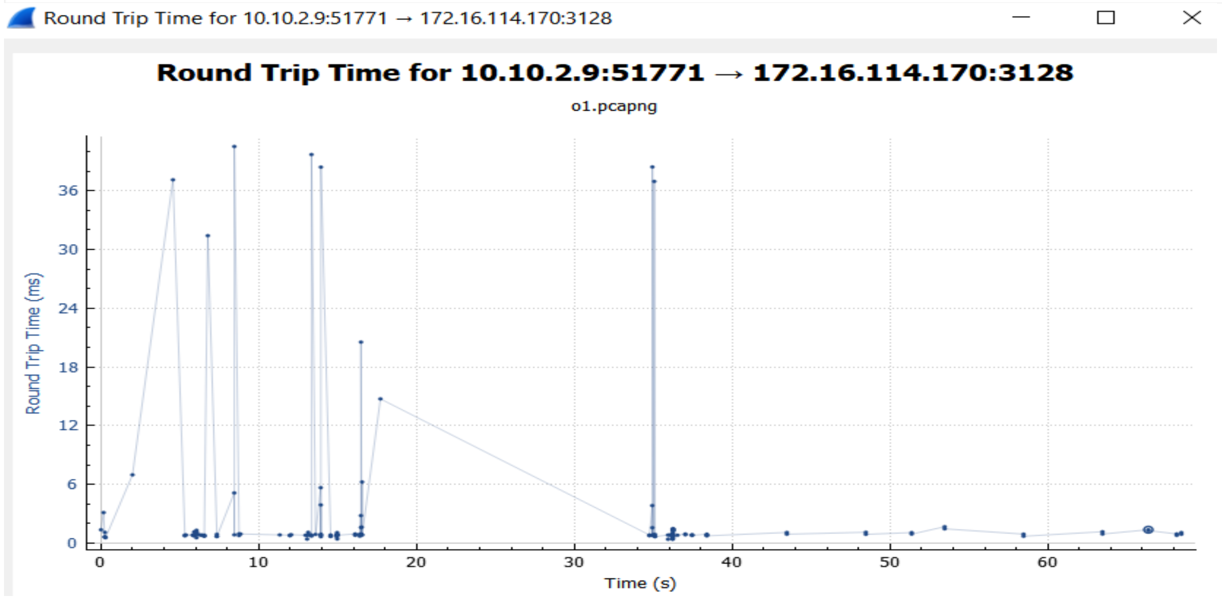
Details	01:07 AM(Lab Proxy :172.16.114.170)	03:27 PM(Room Proxy:202.141.80.24)	06:02 PM(Room Proxy: 202.141.80.24)
Throughput(in bytes/sec)	64k	302k	1219k
RTT(in ms)	0.1	0.921	0.901
Packet Size(in Bytes)	561.5	1118.5	1052.5
No.of packets lost	0	0	0
No.of UDP Packets	2166	683	30
No.of TCP Packets	6773	9276	9698
No.of responses per one request sent	1	1	1

Download Function:

Details	03:05 PM(Room Proxy: 202.141.80.24)	04:36 PM(Jio Wi-Fi)	09:11 PM(Room Proxy:202.141.80.24)
Throughput(in bytes/sec)	386k	115k	1219k
RTT(in ms)	0.813	0.23	0.901
Packet Size(in Bytes)	1062.5	950.5	1052.5
No.of packets lost	0	0	0
No.of UDP Packets	233	124	30
No.of TCP Packets	8587	13707	9698
No.of responses per one request sent	1	1	1

I am attaching graphs to show **RTT and Throughput** when I used Lab Proxy for Playing Video.





Question6: We **cannot** check whether the whole content is being sent from same location **if** we are running the Wireshark **behind a proxy**. So while downloading/Playing the videos from Coursera Website, Destination and Source address would be one of the proxy server address and my computer IP address. If we trace the website **coursera.org** we will find different server IP's. A **proxy server** is an intermediate program or computer used when navigating through different networks of the Internet. They facilitate access to content on the World Wide Web. A proxy intercepts requests and serves back responses. In **real scenarios** content can be sent from different location/source because of **fast switching** with it, after a packet has been sent to the next hop, the **routing information** about how to get to the destination is stored in a fast cache. When the router receives another packet that is directed to the same destination, it uses the cache, which is faster than the traditional way. So, now suppose some router IP address is selected from the routing table which was used earlier and now is **found to be inactive then another router IP address will be selected**, so the routes to same host during the day **can be Different** and content can be provided from **various locations**. When I used **Jio Wi-Fi**,I found **multiple content providers**, their IP Addresses are 52.222.128.123(www-cloudfront-alias.coursera.org), 13.107.21.200(a-0001.a-msedge.net), 52.71.181.107, 23.213.64.35(e10663.g.akamaiedge.net), 172.217.163.174 (clients.l.google.com).