CS349 NETWORKS LAB ASSIGNMENT – 2

(NPTEL/Coursera video lectures)

https://drive.google.com/open?id=1KdiAY-myGlFOoeJC1bSEdAMDqdS2dqO_

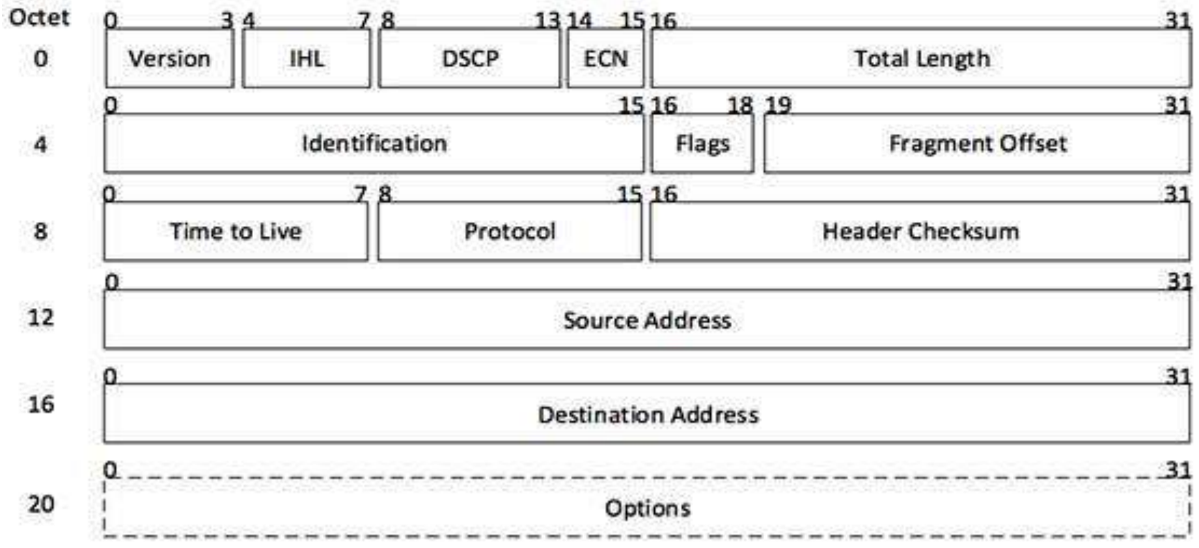
NAME – SUNNY KUMAR ROLL NO -170101068

Question 1. Various Protocols used by the application at different layers are:

- **Physical Layer: Ethernet II**: The first 6 bytes is for the destination address, the next 6 bytes is for source address, next 2 bytes for ether type: IPv4 or IPv6, 46 to 1500 bytes for data, generally consisting upper layer headers. The last 4 bytes is for Frame Check Sequence or CRC.

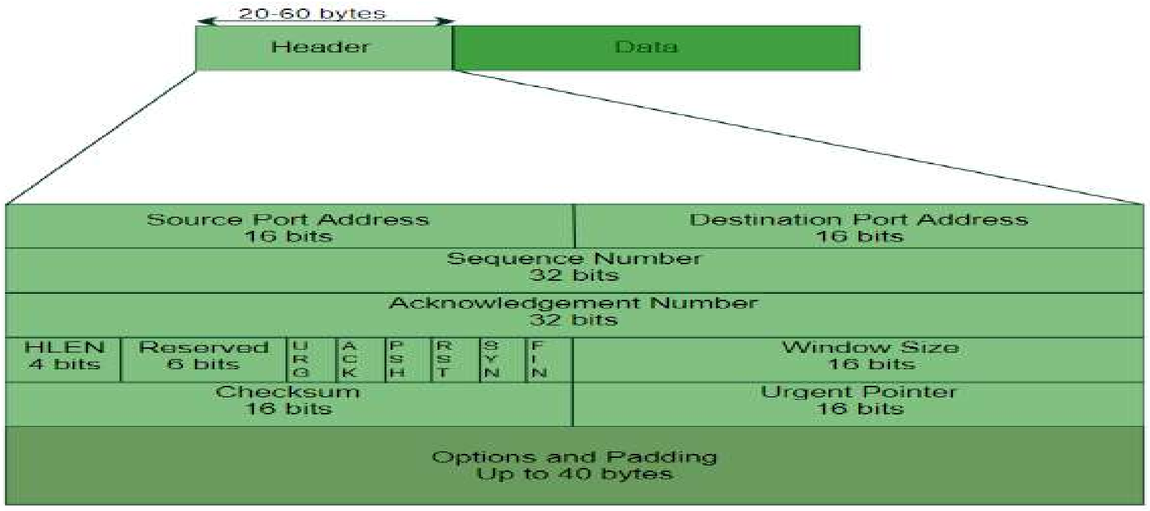
Dest. MAC	Source MAC	Ether Type	Data	FCS
6 bytes	6 bytes	2 bytes	46 to 1500 bytes	4 bytes

- **Network Layer: IPv4**: **Version** of the IP protocol (4 bits), which is 4 for IPv4, **IHL** (Header Length), **DSCP** (Differentiated Services Code Point), **ECN** (Explicit Congestion Notification) allows end to end notification of network congestion without dropping packets, **Total length** is 16-bit field which defines the entire packet size, **identification** is unique Packet Id for identifying the group of fragments of a single IP datagram (16 bits), **Flags:** 3 flags of 1 bit each: reserved bit, do not fragment flag, more fragments flag. **Fragment Offset** represents the no. of data bytes ahead of the particular fragment in the particular Datagram. **Time to Live** represents Datagram's lifetime, **Protocol** field is for the name of protocol, **Header Checksum** for checking errors in the datagram header. **Source Address** sender IP address and **Destination Address** receiver IP address.

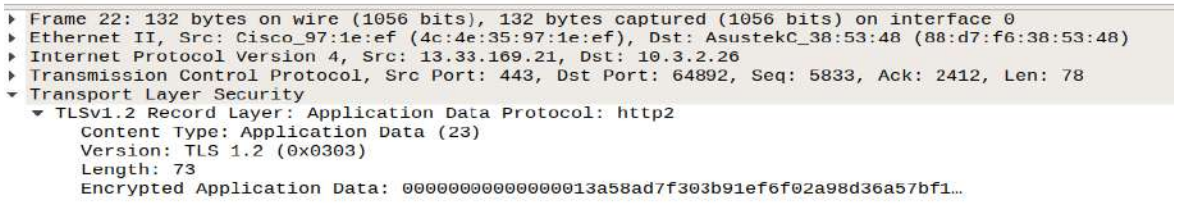


[Image: IP Header]

- **Transport Layer: TCP**: **Source Port** (16 bits) holds the port address of the application that is sending the data segment. **Destination Port** (16 bits) holds the port address of the application in the host that is receiving the data segment. **Sequence number** (32 bits) holds the sequence number. **Acknowledgement number** (32 bits) holds the acknowledgement number. **Header length (HLEN)** is a 4-bit field that indicates the length of the TCP header by no. of 4-byte words in the header. **Flags** are 6 1-bit control bits that control connection establishment, termination, abortion, flow control, mode of transfer etc. **Window Size** tells window size of the sending TCP. **Checksum** field holds the checksum for error control. **Urgent Pointer** is used to point to data that is urgently required.



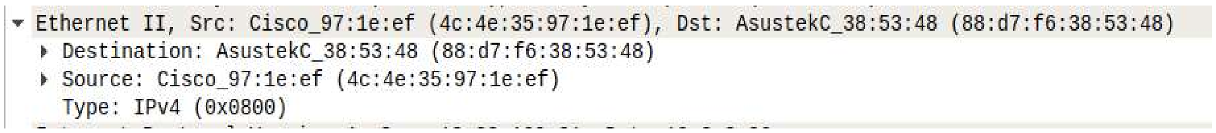
- **TLSv1.2 Record Layer: Application Data Protocol**: This layer acts as an intermediate between transport layer and application layer. Its packet format: **Content Type**, **Version**, **Length** and **Encrypted Application Data**.



```
▸ Frame 22: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
▸ Ethernet II, Src: Cisco_97:1e:ef (4c:4e:35:97:1e:ef), Dst: AsustekC_38:53:48 (88:d7:f6:38:53:48)
▸ Internet Protocol Version 4, Src: 13.33.169.21, Dst: 10.3.2.26
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 64892, Seq: 5833, Ack: 2412, Len: 78
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Application Data Protocol: http2
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 73
      Encrypted Application Data: 00000000000000013a58ad7f303b91ef6f02a98d36a57bf1…
```

Question 2.

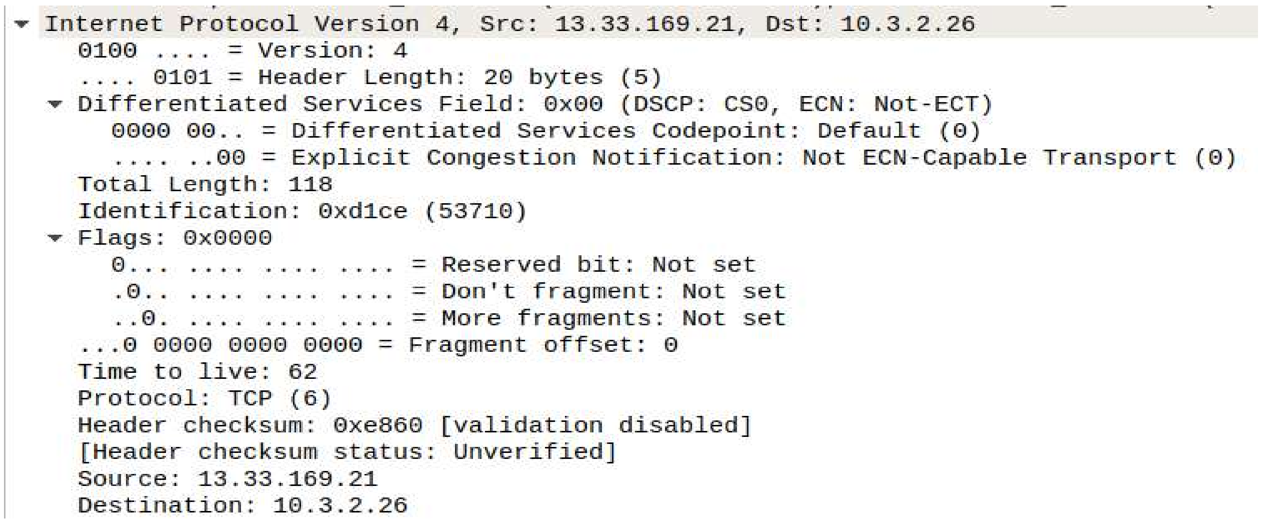
- **Ethernet II: Src: ASUSTekc_38:53:48 (88:d7:f6:38:53:48)**: Hardware MAC address of my PC(source). **Dst: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)**: Hardware MAC address of destination. **Type**: IPv4 (internet protocol version type).



```
▾ Ethernet II, Src: Cisco_97:1e:ef (4c:4e:35:97:1e:ef), Dst: AsustekC_38:53:48 (88:d7:f6:38:53:48)
  ▸ Destination: AsustekC_38:53:48 (88:d7:f6:38:53:48)
  ▸ Source: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
    Type: IPv4 (0x0800)
```

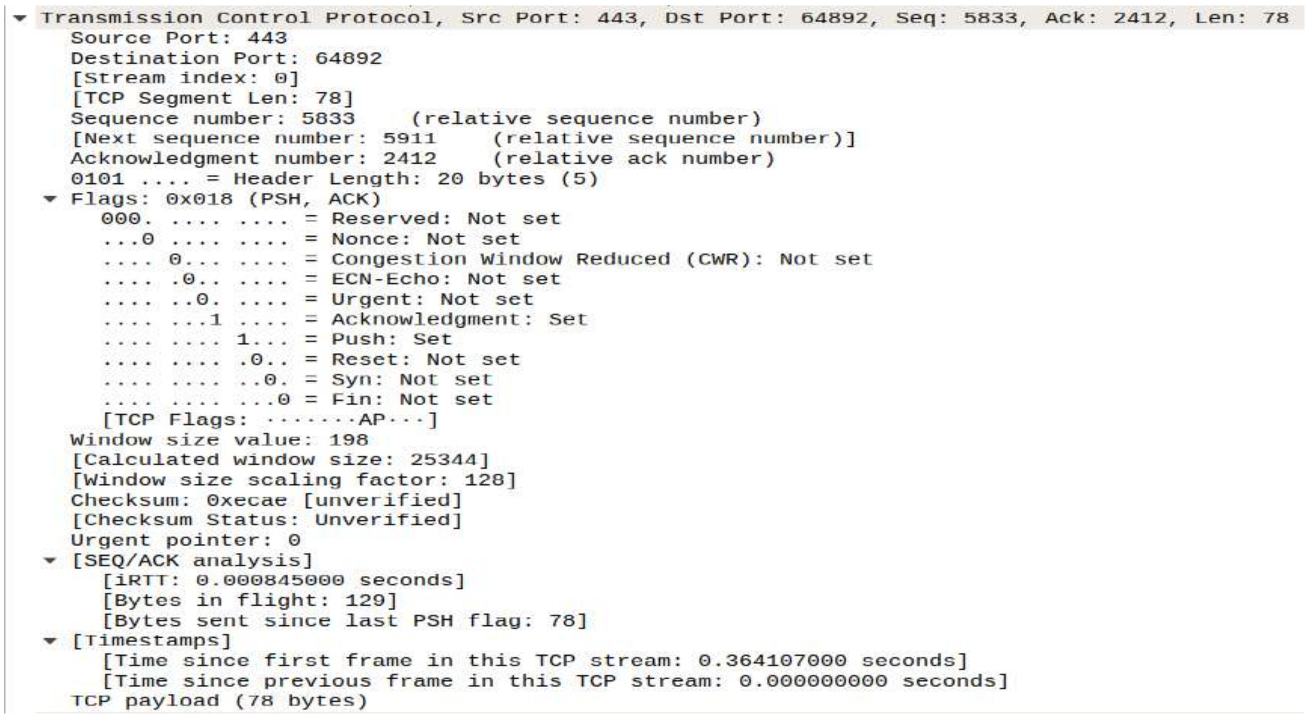
- **Internet Protocol Version 4: Src: 13.33.169.21**- This is source IP address. **Dst: 10.3.2.26**- destination IP address. **Version** is **4** as we are using IPv4. **Header Length** is (5) **20 bytes** as it counts in 4 Bytes word. **Differentiated Services Field**: **0x00** indicates particular quality of service needs from the network. **Total Length** is **118**: is the length of datagram (entire packet size). **Identification**- unique packet id (**0xd1ce**). **Flags: 0x0000** i.e. first 3 bits used for flag bits and rest 13 bits are **fragment offset**. Here **000** is 3-bit flag in which **Bit 0**: reserved, must be zero, **Bit**

1 is **0** that denotes **Don’t Fragment** and **Bit 2** is 0. **Time to Live (62)-** maximum time the datagram is allowed to remain in internet system. **Protocol** is TCP and **Header Checksum:** status unverified (validation disabled).



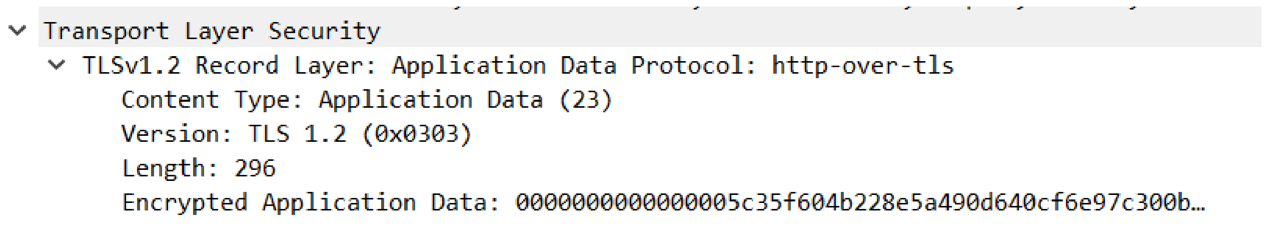
```
▾ Internet Protocol Version 4, Src: 13.33.169.21, Dst: 10.3.2.26
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▾ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 118
    Identification: 0xd1ce (53710)
  ▾ Flags: 0x0000
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..0. .... .... .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 62
    Protocol: TCP (6)
    Header checksum: 0xe860 [validation disabled]
    [Header checksum status: Unverified]
    Source: 13.33.169.21
    Destination: 10.3.2.26
```

- **Transmission Control Protocol:** **Src Port: 443**: source port, which is sending packets. **Dst Port: 64892**: This is receiving/Destination port which is receiving packets. **Sequence number** and **Acknowledgment number** is 5833/2412 (relatively). **Flags**:**018** used to indicate a particular state of connection. **Window size value**: **198**: This is the space for the incoming data. Checksum is **0xecae. Urgent Pointer: 0:** If the URG flag is set, then this field is an offset from the sequence number indicating the last urgent data type.



```
▾ Transmission Control Protocol, Src Port: 443, Dst Port: 64892, Seq: 5833, Ack: 2412, Len: 78
    Source Port: 443
    Destination Port: 64892
    [Stream index: 0]
    [TCP Segment Len: 78]
    Sequence number: 5833    (relative sequence number)
    [Next sequence number: 5911    (relative sequence number)]
    Acknowledgment number: 2412    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  ▾ Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······AP···]
    Window size value: 198
    [Calculated window size: 25344]
    [Window size scaling factor: 128]
    Checksum: 0xecae [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ [SEQ/ACK analysis]
      [iRTT: 0.000845000 seconds]
      [Bytes in flight: 129]
      [Bytes sent since last PSH flag: 78]
  ▾ [Timestamps]
      [Time since first frame in this TCP stream: 0.364107000 seconds]
      [Time since previous frame in this TCP stream: 0.000000000 seconds]
    TCP payload (78 bytes)
```

- **TLSv1.2 Protocol:** Since the http is encrypted by TLS, it cannot be decoded by Wireshark separately. But it shows Application Data Protocol that has field of **content type:** Application Data, **Version** is **TLS 1.2** and **Length** is 296 bytes that is entire packet size.



```
v Transport Layer Security
  v TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 296
      Encrypted Application Data: 0000000000000005c35f604b228e5a490d640cf6e97c300b…
```

Question 3. Protocols used by the application to achieve the important functionalities like 'play', 'pause' and 'download' are **HTTP**, **TCP**, **IPv4**, **TLSv1.2** and **Ethernet II**.

HTTP:

- HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response.
- TCP is in charge of setting up a reliable connection between two machines and HTTP uses this connection to transfer data between the servers and client.

TCP:

- It is evident from the traces that TCP layer handles all the handshaking which is done for the establishment and termination of the connection.
- TCP provides host to host connectivity and control. If the packet is lost, it requests for retransmission.
- It is optimized for accurate delivery.

IPv4:

- This version of IP is used as the basis of the internet, and it establishes all the rules and regulations for the computer networks that function on the principle of packet exchange.
- The good thing about the IP address is that it is used as a unique identifier for computing devices that are connected to a local network or internet.

TLSv1.2:

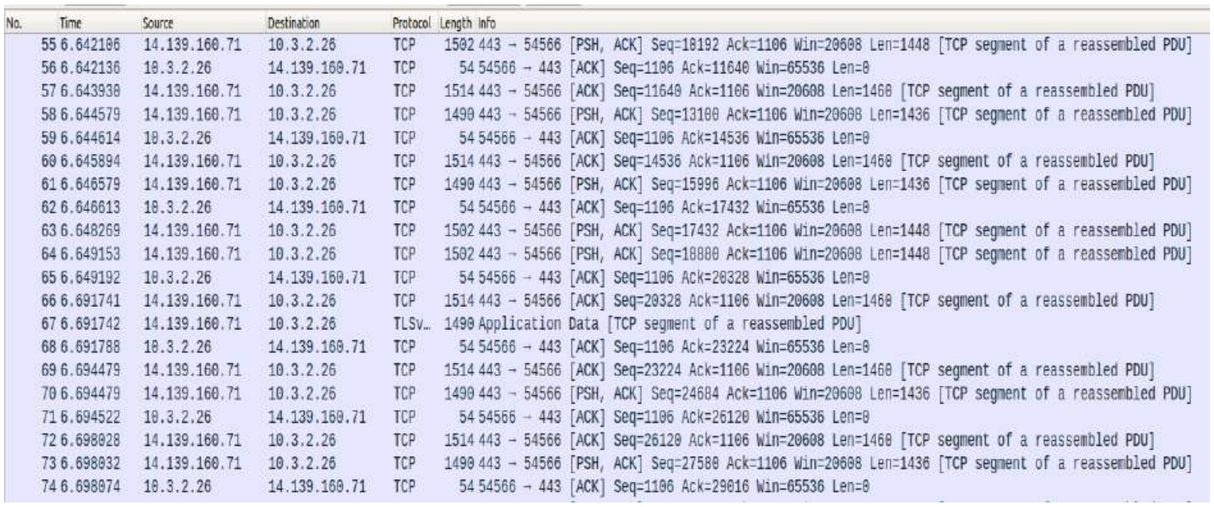
- It provides privacy and data integrity between two communicating devices.
- It provides authentication and data encryption between servers and machines operating over a network.

Ethernet II:

- Ethernet lying in data link layer contains information about MAC addresses of source and destination.
- It determines which switching protocols are used that is important for segmenting the data stream and preventing data congestion.

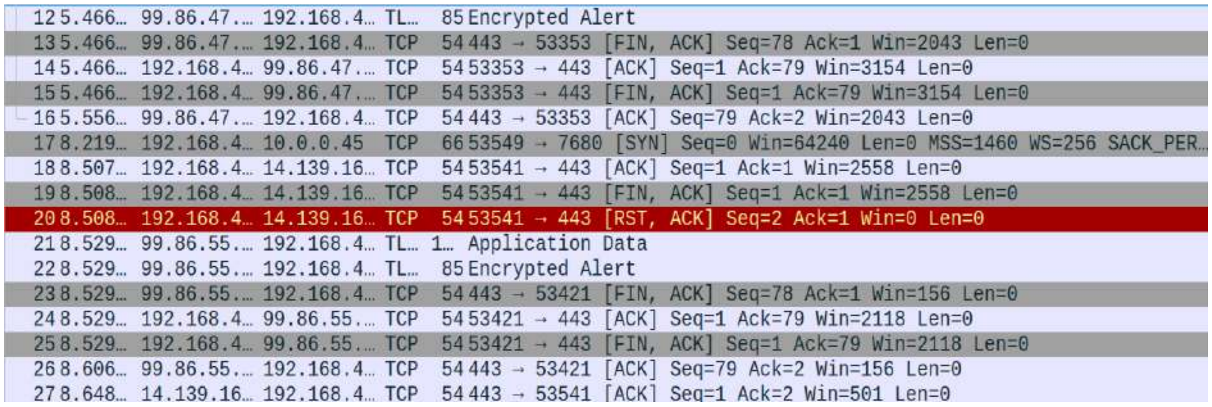
Question 4. I am mentioning here 3 functionalities of the application (**nptel.ac.in**): **Download the video**, **Pause the running video** and **running video**.

- **Download the video:** During downloading video, following traces displayed.
  Initially connection is established through 3-way handshake and followed by TLS handshake. TCP segments continuously transferred from server to our computer. PSH flags received during downloading the video ensures data should be sent immediately.



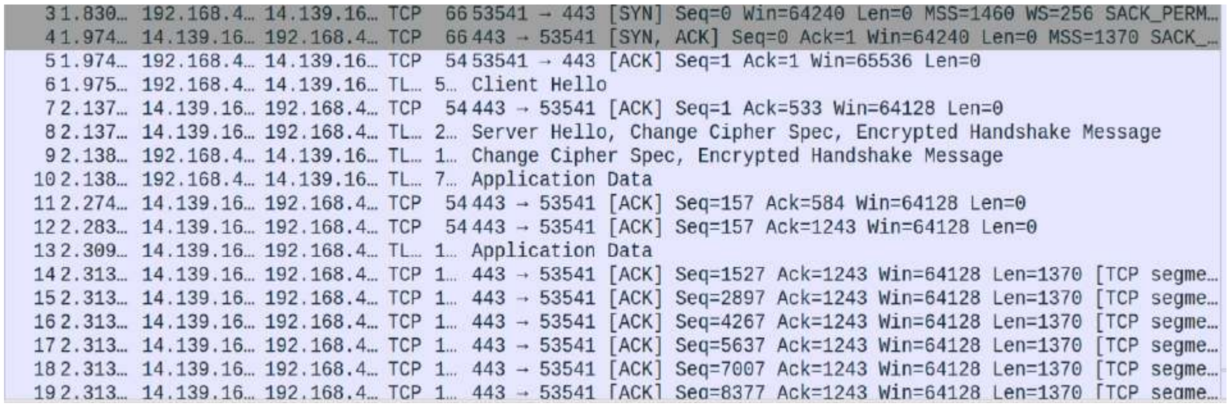
```
No.   Time      Source          Destination     Protocol Length Info
   55 6.642106  14.139.160.71   10.3.2.26       TCP      1502 443 → 54566 [PSH, ACK] Seq=10192 Ack=1106 Win=20608 Len=1448 [TCP segment of a reassembled PDU]
   56 6.642136  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=11640 Win=65536 Len=0
   57 6.643938  14.139.160.71   10.3.2.26       TCP      1514 443 → 54566 [ACK] Seq=11640 Ack=1106 Win=20608 Len=1460 [TCP segment of a reassembled PDU]
   58 6.644579  14.139.160.71   10.3.2.26       TCP      1490 443 → 54566 [PSH, ACK] Seq=13100 Ack=1106 Win=20608 Len=1436 [TCP segment of a reassembled PDU]
   59 6.644614  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=14536 Win=65536 Len=0
   60 6.645894  14.139.160.71   10.3.2.26       TCP      1514 443 → 54566 [ACK] Seq=14536 Ack=1106 Win=20608 Len=1460 [TCP segment of a reassembled PDU]
   61 6.646579  14.139.160.71   10.3.2.26       TCP      1490 443 → 54566 [PSH, ACK] Seq=15996 Ack=1106 Win=20608 Len=1436 [TCP segment of a reassembled PDU]
   62 6.646613  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=17432 Win=65536 Len=0
   63 6.648269  14.139.160.71   10.3.2.26       TCP      1502 443 → 54566 [PSH, ACK] Seq=17432 Ack=1106 Win=20608 Len=1448 [TCP segment of a reassembled PDU]
   64 6.649153  14.139.160.71   10.3.2.26       TCP      1502 443 → 54566 [PSH, ACK] Seq=18880 Ack=1106 Win=20608 Len=1448 [TCP segment of a reassembled PDU]
   65 6.649192  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=20328 Win=65536 Len=0
   66 6.691741  14.139.160.71   10.3.2.26       TCP      1514 443 → 54566 [ACK] Seq=20328 Ack=1106 Win=20608 Len=1460 [TCP segment of a reassembled PDU]
   67 6.691742  14.139.160.71   10.3.2.26       TLSv…    1490 Application Data [TCP segment of a reassembled PDU]
   68 6.691788  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=23224 Win=65536 Len=0
   69 6.694479  14.139.160.71   10.3.2.26       TCP      1514 443 → 54566 [ACK] Seq=23224 Ack=1106 Win=20608 Len=1460 [TCP segment of a reassembled PDU]
   70 6.694479  14.139.160.71   10.3.2.26       TCP      1490 443 → 54566 [PSH, ACK] Seq=24684 Ack=1106 Win=20608 Len=1436 [TCP segment of a reassembled PDU]
   71 6.694522  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=26120 Win=65536 Len=0
   72 6.698028  14.139.160.71   10.3.2.26       TCP      1514 443 → 54566 [ACK] Seq=26120 Ack=1106 Win=20608 Len=1460 [TCP segment of a reassembled PDU]
   73 6.698032  14.139.160.71   10.3.2.26       TCP      1490 443 → 54566 [PSH, ACK] Seq=27580 Ack=1106 Win=20608 Len=1436 [TCP segment of a reassembled PDU]
   74 6.698074  10.3.2.26       14.139.160.71   TCP        54 54566 → 443 [ACK] Seq=1106 Ack=29016 Win=65536 Len=0
```

- **<u>Pause the running video:</u>** After the video was paused, FIN flags received from server **99.86.47.53** which stands **Finished** means there is no more data in the sender. It terminates all TCP connections and so data transfer is stopped.



```
12 5.466… 99.86.47.… 192.168.4… TL…  85 Encrypted Alert
13 5.466… 99.86.47.… 192.168.4… TCP  54 443 → 53353 [FIN, ACK] Seq=78 Ack=1 Win=2043 Len=0
14 5.466… 192.168.4… 99.86.47.… TCP  54 53353 → 443 [ACK] Seq=1 Ack=79 Win=3154 Len=0
15 5.466… 192.168.4… 99.86.47.… TCP  54 53353 → 443 [FIN, ACK] Seq=1 Ack=79 Win=3154 Len=0
16 5.556… 99.86.47.… 192.168.4… TCP  54 443 → 53353 [ACK] Seq=79 Ack=2 Win=2043 Len=0
17 8.219… 192.168.4… 10.0.0.45  TCP  66 53549 → 7680 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PER…
18 8.507… 192.168.4… 14.139.16… TCP  54 53541 → 443 [ACK] Seq=1 Ack=1 Win=2558 Len=0
19 8.508… 192.168.4… 14.139.16… TCP  54 53541 → 443 [FIN, ACK] Seq=1 Ack=1 Win=2558 Len=0
20 8.508… 192.168.4… 14.139.16… TCP  54 53541 → 443 [RST, ACK] Seq=2 Ack=1 Win=0 Len=0
21 8.529… 99.86.55.… 192.168.4… TL… 1… Application Data
22 8.529… 99.86.55.… 192.168.4… TL…  85 Encrypted Alert
23 8.529… 99.86.55.… 192.168.4… TCP  54 443 → 53421 [FIN, ACK] Seq=78 Ack=1 Win=156 Len=0
24 8.529… 192.168.4… 99.86.55.… TCP  54 53421 → 443 [ACK] Seq=1 Ack=79 Win=2118 Len=0
25 8.529… 192.168.4… 99.86.55.… TCP  54 53421 → 443 [FIN, ACK] Seq=1 Ack=79 Win=2118 Len=0
26 8.606… 99.86.55.… 192.168.4… TCP  54 443 → 53421 [ACK] Seq=79 Ack=2 Win=156 Len=0
27 8.648… 14.139.16… 192.168.4… TCP  54 443 → 53541 [ACK] Seq=1 Ack=2 Win=501 Len=0
```

- **<u>Running video:</u>** First a sequence of TCP connection establishment through 3-way handshake. This got response from TCP after a TLS handshake to ensure security. While video is running it gives response through continuously getting TCP data transfer and obtaining **Application Data**.

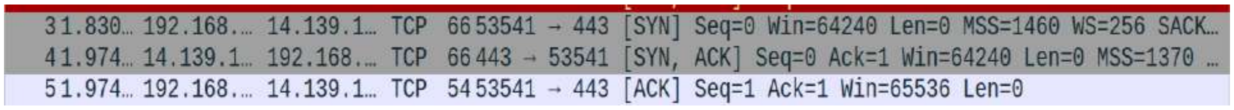


```
 3 1.830… 192.168.4… 14.139.16… TCP 66 53541 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM…
 4 1.974… 14.139.16… 192.168.4… TCP 66 443 → 53541 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1370 SACK_…
 5 1.974… 192.168.4… 14.139.16… TCP 54 53541 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
 6 1.975… 192.168.4… 14.139.16… TL… 5… Client Hello
 7 2.137… 14.139.16… 192.168.4… TCP 54 443 → 53541 [ACK] Seq=1 Ack=533 Win=64128 Len=0
 8 2.137… 14.139.16… 192.168.4… TL… 2… Server Hello, Change Cipher Spec, Encrypted Handshake Message
 9 2.138… 192.168.4… 14.139.16… TL… 1… Change Cipher Spec, Encrypted Handshake Message
10 2.138… 192.168.4… 14.139.16… TL… 7… Application Data
11 2.274… 14.139.16… 192.168.4… TCP 54 443 → 53541 [ACK] Seq=157 Ack=584 Win=64128 Len=0
12 2.283… 14.139.16… 192.168.4… TCP 54 443 → 53541 [ACK] Seq=157 Ack=1243 Win=64128 Len=0
13 2.309… 14.139.16… 192.168.4… TL… 1… Application Data
14 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=1527 Ack=1243 Win=64128 Len=1370 [TCP segme…
15 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=2897 Ack=1243 Win=64128 Len=1370 [TCP segme…
16 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=4267 Ack=1243 Win=64128 Len=1370 [TCP segme…
17 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=5637 Ack=1243 Win=64128 Len=1370 [TCP segme…
18 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=7007 Ack=1243 Win=64128 Len=1370 [TCP segme…
19 2.313… 14.139.16… 192.168.4… TCP 1… 443 → 53541 [ACK] Seq=8377 Ack=1243 Win=64128 Len=1370 [TCP segme…
```

The following sequence of message exchanges are observed:

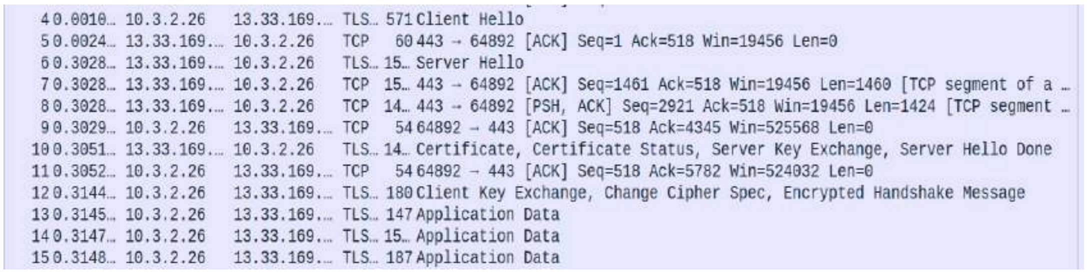
- **<u>TCP Establishment:</u>** Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections, this is called passive open. Once the passive open is established, a client may initiate an active open. Establishing a normal TCP connection requires these steps:

a) The client sends a [SYN] frame for requesting the server to synchronize.
b) The server sends the [SYN, ACK] frame for acknowledging the request for synchronization that was sent by the client.
c) The client then sends [ACK] acknowledging the request from the server.



```
 31.830… 192.168.… 14.139.1… TCP  66 53541 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK…
 41.974… 14.139.1… 192.168.… TCP  66 443 → 53541 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1370 …
 51.974… 192.168.… 14.139.1… TCP  54 53541 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
```

- **<u>TLS Handshaking Message:</u>** 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**. The TLS Handshake protocol allows authenticated communication between the server and client. This protocol allows the client and server to agree upon encryption keys before the selected application protocol begins to send data. There are various messages as **Client Hello, Server Hello, Change Cipher spec, Encrypted Handshake Message.**



```
  4 0.0010… 10.3.2.26    13.33.169.… TLS… 571 Client Hello
  5 0.0024… 13.33.169.… 10.3.2.26    TCP   60 443 → 64892 [ACK] Seq=1 Ack=518 Win=19456 Len=0
  6 0.3028… 13.33.169.… 10.3.2.26    TLS… 15… Server Hello
  7 0.3028… 13.33.169.… 10.3.2.26    TCP  15… 443 → 64892 [ACK] Seq=1461 Ack=518 Win=19456 Len=1460 [TCP segment of a …
  8 0.3028… 13.33.169.… 10.3.2.26    TCP  14… 443 → 64892 [PSH, ACK] Seq=2921 Ack=518 Win=19456 Len=1424 [TCP segment …
  9 0.3029… 10.3.2.26    13.33.169.… TCP   54 64892 → 443 [ACK] Seq=518 Ack=4345 Win=525568 Len=0
 10 0.3051… 13.33.169.… 10.3.2.26    TLS… 14… Certificate, Certificate Status, Server Key Exchange, Server Hello Done
 11 0.3052… 10.3.2.26    13.33.169.… TCP   54 64892 → 443 [ACK] Seq=518 Ack=5782 Win=524032 Len=0
 12 0.3144… 10.3.2.26    13.33.169.… TLS… 180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
 13 0.3145… 10.3.2.26    13.33.169.… TLS… 147 Application Data
 14 0.3147… 10.3.2.26    13.33.169.… TLS… 15… Application Data
 15 0.3148… 10.3.2.26    13.33.169.… TLS… 187 Application Data
```

<u>Question 5.</u> Statistical Analysis:

Details	Morning (10:38 AM)	Afternoon (12:12 PM)	Night (11:39 PM)
Address A	10.3.2.26	192.168.43.128	10.3.2.26
Address B	13.33.169.21	13.35.130.79	13.33.169.115
Throughput (bits/s)	130k	3.138k	128k
Average RTT (ms)	4.07	62.01	6.92
Packet Size (bytes)	385	349	364
No. of packets lost	0	0	0
TCP packets	27	34	29
UDP packets	0	0	0
Response per Request	1.25	1	1.23

<u>Question 6</u>. Yes, I have captured packets at 3 different hours as mentioned in the above table and got multiple sources from which content is being sent and their IPs are:

Time	10:38 AM	12:12 PM	11:39 PM
IP	13.33.169.21	13.35.130.79	13.33.169.115

Here, I have used my room IP in morning and night and Jio-wifi in afternoon for experiment.

<u>Reasons for multiple IPs</u>

- To prevent traffic from being exchanged via the gateway, speeding things up and reducing the load.
- To compensate for a host that's down at that moment
- To serve web pages based on the user location. While using ping with domain name or type the domain name in a browser, the IP address is fetched from a DNS server. So, the application uses whatever IP address it gets from the DNS server (probably using round-robin).