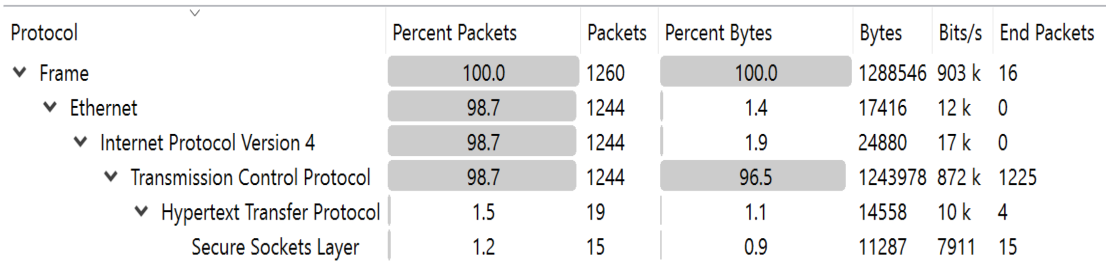
CS 349 Networks Lab Assignment 2 By: Shubham Singhal 150101072

Live Sports Streaming : www.sonyliv.com **Google Drive Link** : https://.goo.gl/NUeApn

Question 1

Protocols used by the application at different layers:

Data Link Layer	**Ethernet**
Network Layer	**IPv4**
Transport Layer	**TCP**
Session Layer	**TLSv1.2**
Application Layer	**HTTP**



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
Frame	100.0	1260	100.0	1288546	903 k	16
Ethernet	98.7	1244	1.4	17416	12 k	0
Internet Protocol Version 4	98.7	1244	1.9	24880	17 k	0
Transmission Control Protocol	98.7	1244	96.5	1243978	872 k	1225
Hypertext Transfer Protocol	1.5	19	1.1	14558	10 k	4
Secure Sockets Layer	1.2	15	0.9	11287	7911	15

HTTP (Hypertext Transfer Protocol):

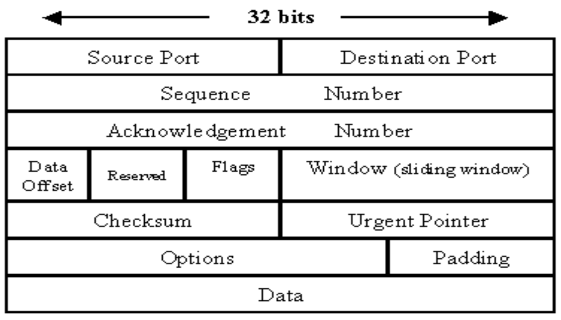
- The Hypertext Transfer Protocol (HTTP) is an Application protocol for distributed, collaborative, hypermedia information systems.
- An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server. Once TCP connection is setup, HTTP connection is started and achieved in a 3-way connection.
- HTTP request contains fields such as Request Method (like GET/POST/PUT) and host name and data.
- HTTP response contains fields: status code, returned html page/other data, Content-type, content-length etc.

TLS (Transport layer Security):

- TLS protocol sits between the Application Layer and the Transport Layer.
- It aims primarily to provide **privacy and data integrity** between two communicating computer applications.
- The connection is private because symmetric cryptography is used to encrypt the data transmitted.
- The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret negotiated at the start of the session(TLS Handshake).
- The connection ensures integrity because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.
- The **TLS header** comprises three fields, necessary to allow the higher layer to be built upon it:
  - Byte 0: TLS record type(example : CHANGE_CIPHER_SPEC , ALERT, HANDSHAKE,APPLICATION DATA)
  - Bytes 1-2: TLS version (major/minor) (example : TLSv1.0, TLSv1.1, TLSv1.2)
  - Bytes 3-4: Length of data in the record (excluding the header itself). The maximum supported is 16384 (16K).
- The **TLS record** contains the TLS protocol for example: Handshake protocol or changecipherspec protocol.

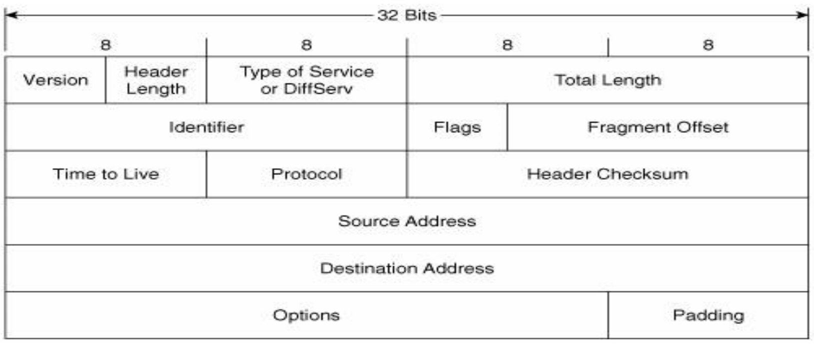
TCP (Transmission Control Protocol): TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating by an IP network. A TCP packet is a sequence of octets (bytes) and consists of a header followed by a body. The body contains the data from the layer above it.
The header describes the following :

- **Source and Destination Port**(2 bytes each) : communication endpoints for sending and receiving devices.
- **Sequence Number and Acknowledgment Number**(4 bytes each) : Sequence numbers of the packets.
- **TCP data offset** (4 bits) : the total size of a TCP header in multiples of four bytes.
- **Reserved data** (3 bits) : his field serves the purpose of aligning the total header size as a multiple of four bytes
- **Control flags** (up to 9 bits): to manage data flow in specific situations.
- **Window size** (2 bytes): to regulate how much data they send to a receiver before requiring an acknowledgment in return
- **TCP checksum** (2 bytes): Error Control
- **Urgent pointer** (2 bytes): to regulate how much data they send to a receiver before requiring an acknowledgment in return
- **TCP optional data** (0-40 bytes): to regulate how much data they send to a receiver before requiring an acknowledgment in return



IPv4(Internet Protocol): IPv4 is a connectionless protocol for use on packet-switched networks. It operates on a best effort delivery model. The encapsulated data is referred to as IP Payload. IP header contains all the necessary information to deliver the packet at the other end.

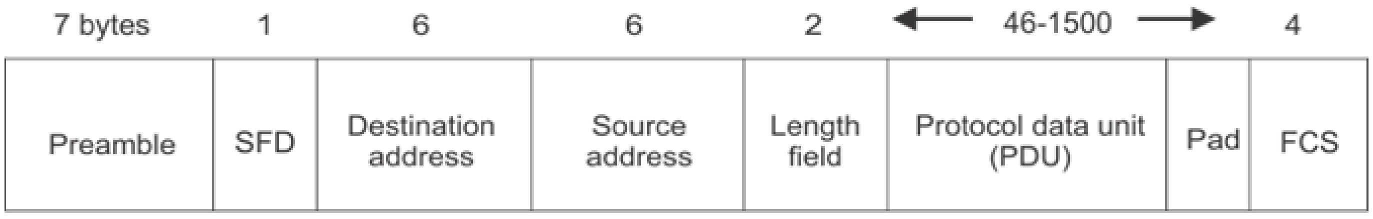
- **Version** (4 bits): four-bit version field. For IPv4, this is always equal to 4.
- **Internet Header Length** (IHL)(4 bits): this field specifies the size of the header in mutiples of 32 bits .The minimum value for this field is 5 which indicates a length of 5 × 32 bits = 160 bits = 20 bytes. As a 4-bit field, the maximum value is 15 words (15 × 32 bits, or 480 bits = 60 bytes).
- **Differentiated Services Code Point** (DSCP)(6 bits): This field is now defined for Differentiated services. New technologies are emerging that require real-time data streaming and therefore make use of the DSCP field. An example is Voice over IP (VoIP), which is used for interactive data voice exchange.
- **Explicit Congestion Notification (ECN)**(2 bits): allows end-to-end notification of network congestion without dropping packets.
- **Total Length**(16 bits): This 16-bit field defines the entire packet size in bytes, including header and data. The minimum size is 20 bytes (header without data) and the maximum is 65,535 bytes.
- **Identification**(16 bits): for uniquely identifying the group of fragments of a single IP datagram.
- **Flags** (3 bits): A three-bit field follows and is used to control or identify fragments. They are
  bit 0: Reserved; must be zero.
  bit 1: Don't Fragment (DF)
  bit 2: More Fragments (MF)
- **Fragment Offset**(13bits): specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram. The first fragment has an offset of zero. This allows a maximum offset of (213 – 1) × 8 = 65,528 bytes, which would exceed the maximum IP packet length of 65,535 bytes with the header length included (65,528 + 20 = 65,548 bytes).
- **Time To Live (TTL)**(8 bits): An eight-bit time to live field helps prevent datagrams from persisting the field has become a hop count—when the datagram arrives at a router, the router decrements the TTL field by one.
- **Protocol**(8 bits): This field defines the protocol used in the data portion of the IP datagram.
- **Header Checksum**(16 bits): The 16-bit checksum field is used for error-checking of the header.
- **Source address**(32 bits): This field is the IPv4 address of the sender of the packet.
- **Destination address**(32 bits): This field is the IPv4 address of the receiver of the packet.



__**Ethernet II(Data Link layer):**__ Data link layer protocols are point to point protocols. The data link layer is concerned with local delivery of frames between devices on the same LAN.

__**Ethernet Frame Format:**__

- **Preamble**(7 bytes) : This is a stream of bits (alternating 1's and 0's) used to allow the transmitter and reciever to synchronize their communication.
- **SFD**(1 bytes) : This is always 10101011 and is used to indicate the beginning of the frame information.
- **Destination/Source MAC**(6 bytes each): This is the MAC address of the machine receiving/sending data. The MAC address is 6bytes longs. Initial 3 bytes are unique to each manufacturer. Last 3 bytes are unique to each hardware.
- **Length(2 bytes)**: This is the length of the entire Ethernet frame in bytes.
- **Data** (Payload): The data is inserted here. This is where the data from IP layer is placed.
- **FCS(4 bytes)**: This field contains the Frame Check Sequence (FCS) which is calculated using a Cyclic Redundancy Check (CRC). The FCS allows Ethernet to detect errors in the Ethernet frame and reject the frame if it appears damaged.

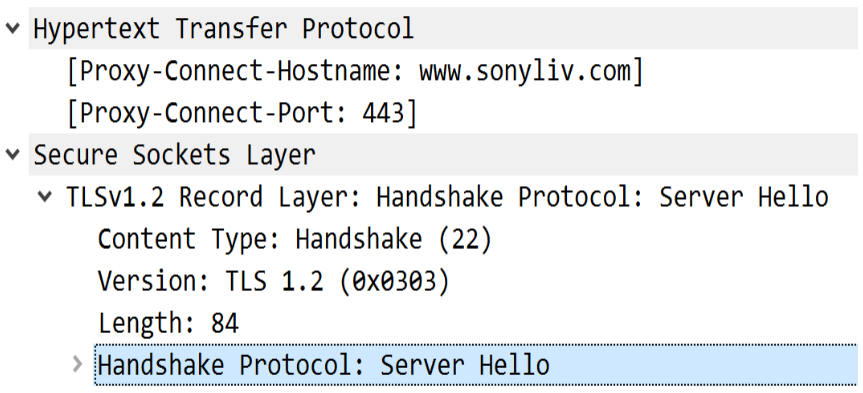


Question 2

Observed values for various fields of the protocols:

- All the fields are explained for the protocols in question 1. Here I am reporting the values specific to my observed values.

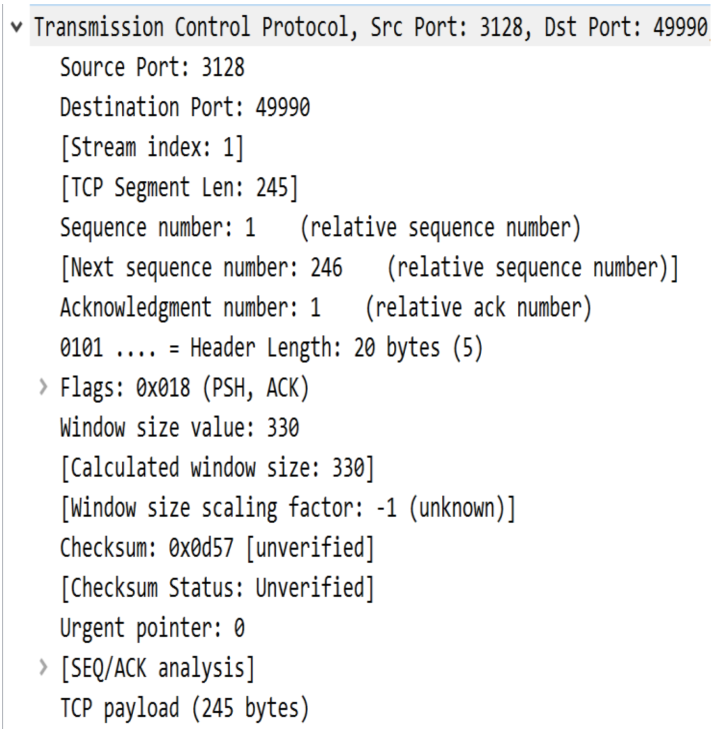
HTTP (Hypertext Transfer Protocol) and TLSv1.2(Transport layer security):



```
v Hypertext Transfer Protocol
    [Proxy-Connect-Hostname: www.sonyliv.com]
    [Proxy-Connect-Port: 443]
v Secure Sockets Layer
  v TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 84
    > Handshake Protocol: Server Hello
```

HTTP	
Hostname	www.sonyliv.com
Port	443 (For https)
SSL	
Record Type	Server Hello(message from server to client in 3 way handshaking protocol)
Version	TLSv1.2
Length	84 bytes (length of record excluding header)
Protocol	Handshake protocol

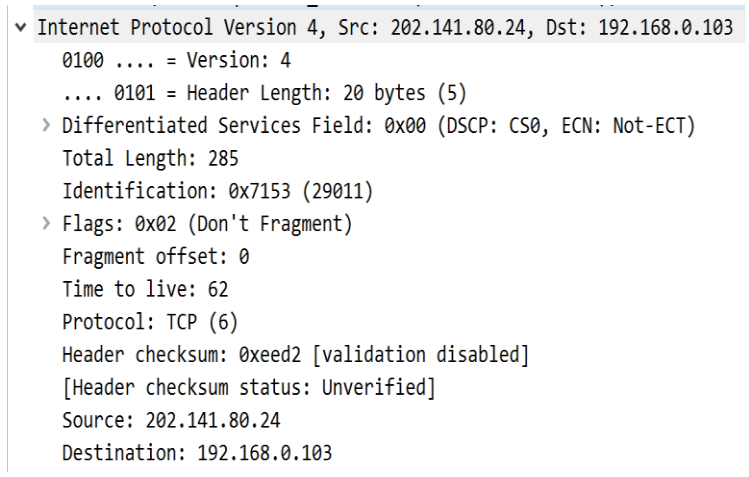
TCP(Transport Control Protocol):



```
v Transmission Control Protocol, Src Port: 3128, Dst Port: 49990
    Source Port: 3128
    Destination Port: 49990
    [Stream index: 1]
    [TCP Segment Len: 245]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 246    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 330
    [Calculated window size: 330]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x0d57 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
    TCP payload (245 bytes)
```

TCP	
Source Port	3128 (Port of proxy server)
Destination Port	49,900 (Local port opened by chrome)
Length(in multiples of 4)	5 (length of packet: 20 bytes)
Seq Number	246 (This is relative related to tcp connection)
Ack Number	1
Flag	PSH and ACK flags are set.
Checksum	0x0d57 (For error detection)
Payload	245 Bytes

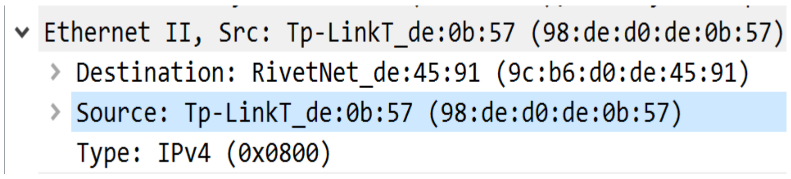
IPv4(Internet Protocol):



```
v Internet Protocol Version 4, Src: 202.141.80.24, Dst: 192.168.0.103
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 285
    Identification: 0x7153 (29011)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 62
    Protocol: TCP (6)
    Header checksum: 0xeed2 [validation disabled]
    [Header checksum status: Unverified]
    Source: 202.141.80.24
    Destination: 192.168.0.103
```

IPv4	
Version	4 (IPv4)
Header Length(In multiple of 4 bytes)	5 (and hence size is 5*4= 20 Bytes)
Total Length	285 Bytes
Checksum	0xeed2
Flags	Don't Fragment(If the DF flag is set, and fragmentation is required to route the packet, then the packet is dropped.)
TTL	62
Protocol	TCP
Source	202.141.80.24(Proxy server's IP)
Destination	192.168.0.103 (My ip)

Ethernet:



```
⌄ Ethernet II, Src: Tp-LinkT_de:0b:57 (98:de:d0:de:0b:57)
  › Destination: RivetNet_de:45:91 (9c:b6:d0:de:45:91)
  › Source: Tp-LinkT_de:0b:57 (98:de:d0:de:0b:57)
    Type: IPv4 (0x0800)
```

Ethernet	Packet coming to my pc from Router
Destination MAC	9c:b6:d0:de:0b:57(MAC address of my wifi card by manufacturer RivetNET
SOURCE MAC	98:de:d0:de:0b:57(MAC address of my Tp-Link router)

Question 3

Messages exchanged by the application:

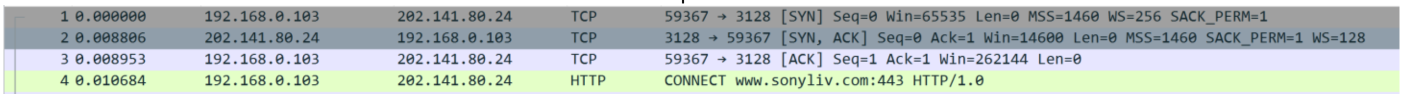
- **TCP connection Establishment:**
  - **ON OPENING WEBSITE**

    To establish a connection, TCP uses a three-way handshake. 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.

    The three-way handshake:
    - SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a some value A.
    - SYN-ACK: In response, the server replies with a SYN-ACK (acknowledging our SYN request). The ack number is set to one more than the received sequence number (A + 1), and the sequence number that the server chooses for the packet is another some number, B.
    - ACK: Finally, the client sends an ACK back to the server to acknowledge the SYN-ACK packet from server. The sequence number is set to the received ack value i.e. A + 1, and the ack number is set to one more than the received sequence number i.e. B + 1.

    Note : In Below image: A=0;B=0;

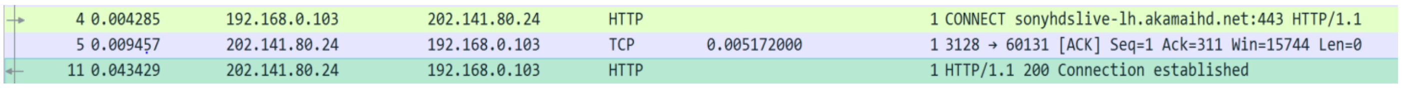
    As soon as TCP connection is established HTTP request is sent.



```
1 0.000000   192.168.0.103   202.141.80.24   TCP    59367 → 3128 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
2 0.008806   202.141.80.24   192.168.0.103   TCP    3128 → 59367 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=128
3 0.008953   192.168.0.103   202.141.80.24   TCP    59367 → 3128 [ACK] Seq=1 Ack=1 Win=262144 Len=0
4 0.010684   192.168.0.103   202.141.80.24   HTTP   CONNECT www.sonyliv.com:443 HTTP/1.0
```

- **HTTP connection establishment:**
  - **ONCE TCP CONNECTION HAS BEEN BUILT:**

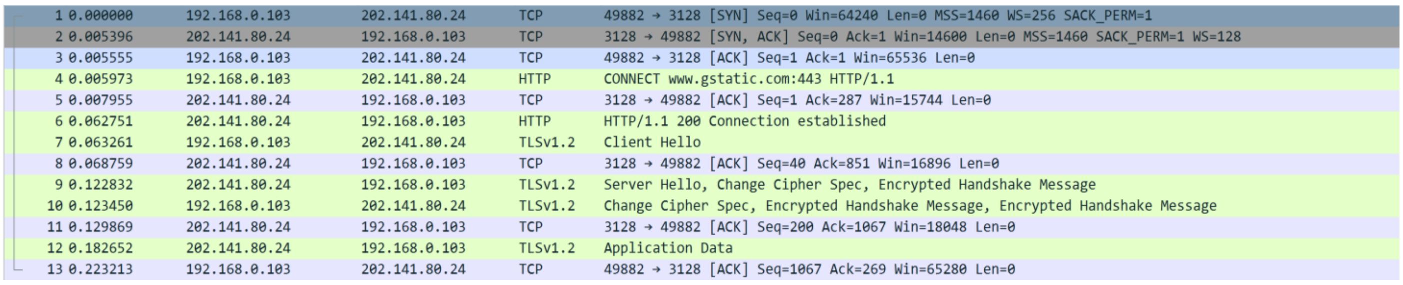
    In client-server protocols, like HTTP, sessions consist of three phases:
    - The client establishes a TCP connection (HTTP request is sent once TCP connection is established).
    - The client sends its request, and waits for the acknowledgement from server.
    - The server processes the request, sending back its acknowledgement, providing a status code and appropriate data.



```
 4 0.004285   192.168.0.103   202.141.80.24   HTTP                  1 CONNECT sonyhdslive-lh.akamaihd.net:443 HTTP/1.1
 5 0.009457   202.141.80.24   192.168.0.103   TCP    0.005172000   1 3128 → 60131 [ACK] Seq=1 Ack=311 Win=15744 Len=0
11 0.043429   202.141.80.24   192.168.0.103   HTTP                  1 HTTP/1.1 200 Connection established
```

- **Handshaking Sequences**:
  - **While opening website after HTTP connection**

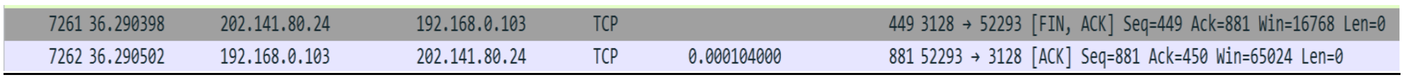
  The TLS Handshake Protocol is responsible for the authentication and key exchange necessary to establish or resume secure sessions. When establishing a secure session, the Handshake Protocol manages the following:.
  - The client sends a "Client hello" message to the server, along with the client's random value and supported cipher suites.
  - The server responds by sending a "Server hello" message to the client, along with the server's random value.
  - The server sends its certificate to the client for authentication and may request a certificate from the client. The server sends the "Server hello done" message.
  - If the server has requested a certificate from the client, the client sends it.
  - The client creates a random Pre-Master Secret and encrypts it with the public key from the server's certificate, sending the encrypted Pre-Master Secret to the server.
  - The server receives the Pre-Master Secret. The server and client each generate the Master Secret and session keys based on the Pre-Master Secret.
  - The client sends "Change cipher spec" notification to server to indicate that the client will start using the new session keys for hashing and encrypting messages. Client also sends "Client finished" message.
  - Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption using the session keys. Server sends "Server finished" message to the client.
  - Client and server can now exchange application data over the secured channel they have established. All messages sent from client to server and from server to client are encrypted using session key.



```
 1 0.000000   192.168.0.103   202.141.80.24   TCP      49882 → 3128 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
 2 0.005396   202.141.80.24   192.168.0.103   TCP      3128 → 49882 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=128
 3 0.005555   192.168.0.103   202.141.80.24   TCP      49882 → 3128 [ACK] Seq=1 Ack=1 Win=65536 Len=0
 4 0.005973   192.168.0.103   202.141.80.24   HTTP     CONNECT www.gstatic.com:443 HTTP/1.1
 5 0.007955   202.141.80.24   192.168.0.103   TCP      3128 → 49882 [ACK] Seq=1 Ack=287 Win=15744 Len=0
 6 0.062751   202.141.80.24   192.168.0.103   HTTP     HTTP/1.1 200 Connection established
 7 0.063261   192.168.0.103   202.141.80.24   TLSv1.2  Client Hello
 8 0.068759   202.141.80.24   192.168.0.103   TCP      3128 → 49882 [ACK] Seq=40 Ack=851 Win=16896 Len=0
 9 0.122832   202.141.80.24   192.168.0.103   TLSv1.2  Server Hello, Change Cipher Spec, Encrypted Handshake Message
10 0.123450   192.168.0.103   202.141.80.24   TLSv1.2  Change Cipher Spec, Encrypted Handshake Message, Encrypted Handshake Message
11 0.129869   202.141.80.24   192.168.0.103   TCP      3128 → 49882 [ACK] Seq=200 Ack=1067 Win=18048 Len=0
12 0.182652   202.141.80.24   192.168.0.103   TLSv1.2  Application Data
13 0.223213   192.168.0.103   202.141.80.24   TCP      49882 → 3128 [ACK] Seq=1067 Ack=269 Win=65280 Len=0
```

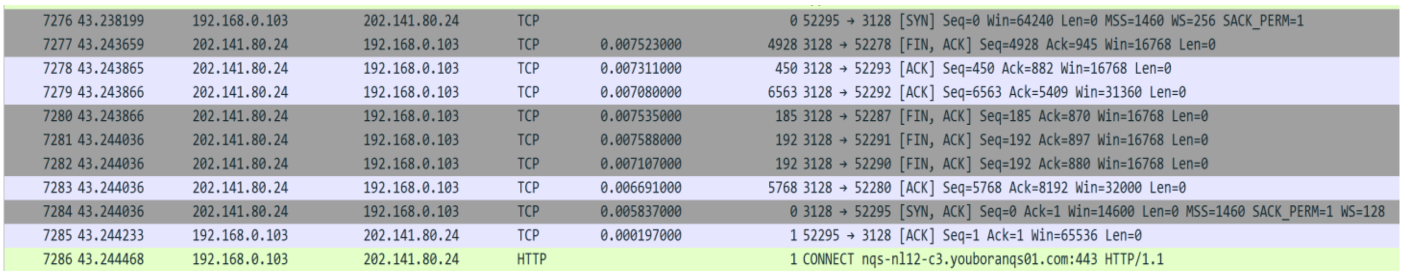
NOTE: video was paused at 36 seconds and played again at 43 seconds. Trace is present in file named "data_5(play_pause_play)"

- **Pausing a video**
  - When a playing live video stream was paused, the current ongoing TCP connection was closed and we received a FIN packet from server.



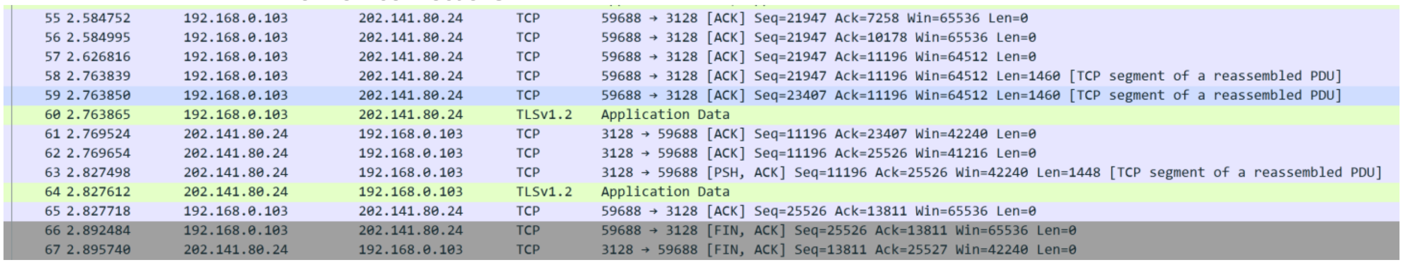
```
7261 36.290398   202.141.80.24   192.168.0.103   TCP                  449 3128 → 52293 [FIN, ACK] Seq=449 Ack=881 Win=16768 Len=0
7262 36.290502   192.168.0.103   202.141.80.24   TCP   0.000104000    881 52293 → 3128 [ACK] Seq=881 Ack=450 Win=65024 Len=0
```

- **Playing a paused video**
  - On playing the video again after pause, all the old tcp connection were terminated which can be seen from FIN packages received from the server.
  - A new TCP connection (3-way connection) is made which is evident from sequence of SYN, SYN-ACK and ACK(in similar way as explained above).



```
7276 43.238199   192.168.0.103   202.141.80.24   TCP                     0 52295 → 3128 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
7277 43.243659   202.141.80.24   192.168.0.103   TCP    0.007523000   4928 3128 → 52278 [FIN, ACK] Seq=4928 Ack=945 Win=16768 Len=0
7278 43.243865   202.141.80.24   192.168.0.103   TCP    0.007311000    450 3128 → 52293 [ACK] Seq=450 Ack=882 Win=16768 Len=0
7279 43.243866   202.141.80.24   192.168.0.103   TCP    0.007080000   6563 3128 → 52292 [ACK] Seq=6563 Ack=5409 Win=31360 Len=0
7280 43.243866   202.141.80.24   192.168.0.103   TCP    0.007535000    185 3128 → 52287 [FIN, ACK] Seq=185 Ack=870 Win=16768 Len=0
7281 43.244036   202.141.80.24   192.168.0.103   TCP    0.007588000    192 3128 → 52291 [FIN, ACK] Seq=192 Ack=897 Win=16768 Len=0
7282 43.244036   202.141.80.24   192.168.0.103   TCP    0.007107000    192 3128 → 52290 [FIN, ACK] Seq=192 Ack=880 Win=16768 Len=0
7283 43.244036   202.141.80.24   192.168.0.103   TCP    0.006691000   5768 3128 → 52280 [ACK] Seq=5768 Ack=8192 Win=32000 Len=0
7284 43.244036   202.141.80.24   192.168.0.103   TCP    0.005837000      0 3128 → 52295 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=128
7285 43.244233   192.168.0.103   202.141.80.24   TCP    0.000197000      1 52295 → 3128 [ACK] Seq=1 Ack=1 Win=65536 Len=0
7286 43.244468   192.168.0.103   202.141.80.24   HTTP                    1 CONNECT nqs-nl12-c3.youboranqs01.com:443 HTTP/1.1
```

- **TCP connection disconnection:**
  - **ON CLOSING CHROME TAB**
    - The connection termination phase uses a four-way handshake, with both side of the connection terminating independently.
    - When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. The other endpoint also sends a FIN and expect a ACK from the first endpoint.
    - After both FIN/ACK exchanges are concluded, the side which sent the first FIN before receiving one waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections



```
55 2.584752   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=21947 Ack=7258 Win=65536 Len=0
56 2.584995   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=21947 Ack=10178 Win=65536 Len=0
57 2.626816   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=21947 Ack=11196 Win=64512 Len=0
58 2.763839   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=21947 Ack=11196 Win=64512 Len=1460 [TCP segment of a reassembled PDU]
59 2.763850   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=23407 Ack=11196 Win=64512 Len=1460 [TCP segment of a reassembled PDU]
60 2.763865   192.168.0.103   202.141.80.24   TLSv1.2  Application Data
61 2.769524   202.141.80.24   192.168.0.103   TCP      3128 → 59688 [ACK] Seq=11196 Ack=23407 Win=42240 Len=0
62 2.769654   202.141.80.24   192.168.0.103   TCP      3128 → 59688 [ACK] Seq=11196 Ack=25526 Win=41216 Len=0
63 2.827498   202.141.80.24   192.168.0.103   TCP      3128 → 59688 [PSH, ACK] Seq=11196 Ack=25526 Win=42240 Len=1448 [TCP segment of a reassembled PDU]
64 2.827612   202.141.80.24   192.168.0.103   TLSv1.2  Application Data
65 2.827718   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [ACK] Seq=25526 Ack=13811 Win=65536 Len=0
66 2.892484   192.168.0.103   202.141.80.24   TCP      59688 → 3128 [FIN, ACK] Seq=25526 Ack=13811 Win=65536 Len=0
67 2.895740   202.141.80.24   192.168.0.103   TCP      3128 → 59688 [FIN, ACK] Seq=13811 Ack=25527 Win=42240 Len=0
```

Question 4

Relevance of Protocols used by the application:

- **TLSv1.2**:
  - It is used to secure (Encrypt) the data to and from the site to clients.
  - The security protocol protects the integrity of the website by helping to prevent intruders tampering with communications between the site and the visitors browsing (a common tactic here is injecting malware) as well as safeguarding privacy and security.
  - Every unprotected HTTP request can potentially reveal information about the behaviors and identities. Thus Login and credentials details are compromised.
  - HTTPS helps SEO(Search Engine Optimizers) by improving your site's ranking in search engine results.
- **TCP:**
  - This was the majorly used protocol in communication with www.sonyliv.com .

- It is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data.
- This handles all the packets relating to TCP 3-way connection and TLS handshaking protocol.
- TCP being in transport layer is a end to end connection and is abstracted over what is in the lower layers(namely Internet, data-link and physical layer).Thus TCP will use IP(either IPv4 / IPv6 and Ethernet in data link layer).

- **UDP:**
  - **UDP packets were not seen while communicating with www.sonyliv.com .**
  - UDP uses connectionless minimalistic communication model. It uses checksums (data integrity) and port numbers but has no handshaking dialogues(unreliability) and no guarantee of delivery, ordering, or duplicate protection.
  - UDP just sends the packets, which means that it has much lower bandwidth overhead and latency. Since no acks and retransmission is present.
  - From theory we have learnt that UDP is preferred in Live video streaming. Since in video streaming if a packet is lost, there is no point in asking for same packet again since once that moment is gone and if we wait for that packet to be transmitted again we wouldn't be live anymore.
- **TCP vs UDP:**
  - TCP was being used instead of UDP on sonyliv.
  - Sonyliv gave the option of playing and pausing the live video and even allowed to rewind the video and is thus highly likely that video is buffered at the sonyliv servers.
  - Also Prefetching was in action (some seconds of video is buffered in local device) thus explaining the fact that TCP can do the job also.
- **IPv4**:
  - The IP has the job of delivering packets using the IP headers from the source to the destination.
  - Existing in the network layer, IPv4 connection is a hop to hop connection.
- **Ethernet:**
  - This contains information about source and destination MAC address found in its header.
  - Ethernet lying in data link layer is also responsible for error detection and correction along with flow control. It exists as a point to point connection.

Question 5

Statistics from traces:

Property	**Time 1(Room Night)**	**Time 2(Room Evening)**	**Time 3(In Lab)**	**Time 3(On mobile data hotspot)**
Host A	192.168.0.103	192.168.0.103	172.16.114.149	192.168.43.118
Host B	202.141.80.24:3128	202.141.80.24:3128	202.141.80.24:3128	Dynamic IPs
Throughput (A -> B)(kbps)	20	24	16	15
Throughput (B -> A)(kbps)	1233	1337	1217	576
RTT (in ms)	0.644	0.685	0.577	44.074
Avg. Packet Size	1080.5	1052.5	2478.5	855.5
No. of packets lost	28 out of 27633 (0.103%)	18 out of 15467 (0.116%)	0 out of 10668 (0%)	27 out of 25955 (0.104%)
TCP packets (A->B)	8025	4757	4590	9204
TCP packets (B->A)	19608	10710	6078	16751
UDP packets	0	0	0	0
Responses recd. wrt per request sent (packets B->A /packets A->B)	2.44	2.25	1.33	1.82

Observation:

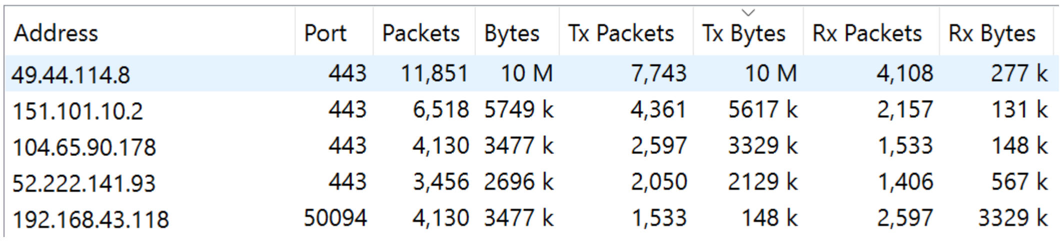
- When taking data on mobile data, we can see the throughput is pretty less and rtt is very high due to slower connection speed.

Note:

- The data was taken after filtering packet related to my application only.
- UDP packets was not used by my application and hence was 0 always
- RTT is calculated by adding a custom field "tcp.analysis.ack_rtt" to the logs and then taking average.
- No of packets lost is taken by filtering packets with "tcp.analysis.lost_segment" which represent that some segment was lost in transmission and hence equal to packets lost.
- Throughput is calculated from 'conversation' option of wireshark in 'statistics' tab. Similarly Avg packets size, No of packets lost , TCP /UDP is taken from various option of wireshark in 'statistics' tab.
- Screenshots of Throughput and RTT are present in "images" folder.

Question 6

- **On Campus Network(Proxy authenticated)**
  - In our campus, we are behind a proxy authenticated network (my proxy server being : 202.141.80.24:3128).
  - Thus all the request sent by my machine will have an destination IP as : 202.141.80.24 and destination port of 3128. Similarly all requests coming to my machine will have an source IP and port as 202.141.80.24 and 3128 respectively.
  - Thus using proxy network I was unable to check if there are connections with multiple ip.
- **On mobile network**
  - **Yes.** sonyliv transmitted tcp packets from **multiple** IP's namely **(49.44.114.8 & 151.101.10.2 & 104.65.90.178 & 52.222.141.93)** as evident from packets in below screenshot.



Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
49.44.114.8	443	11,851	10 M	7,743	10 M	4,108	277 k
151.101.10.2	443	6,518	5749 k	4,361	5617 k	2,157	131 k
104.65.90.178	443	4,130	3477 k	2,597	3329 k	1,533	148 k
52.222.141.93	443	3,456	2696 k	2,050	2129 k	1,406	567 k
192.168.43.118	50094	4,130	3477 k	1,533	148 k	2,597	3329 k

- **Reason for Multiple IP:**
  - Since modern day websites deploy **load balancing on servers**, data is sent to clients from various IP in most efficient manner.
  - **Round Robin DNS** mechanism for faster fetching of relevant pages by balancing the page requests across many servers may lead to multiple IP.
  - It may be possible that video is being streamed from one IP and other site data is being loaded from other server with different IP. Since video and html data might be present on different server and hence different IP.