CS349 Networks Lab - Assignment 2

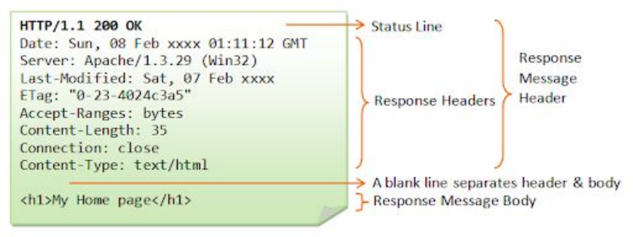
Inderpreet Singh Chera (160101035) - LIVE SPORTS STREAMING

Captured Data: https://drive.google.com/open?id=1Mx8FzrjPiLLfFYJaI2ajLhpeQrfemgDA

Answer 1. Packet format of protocols used by application are briefly explained below.

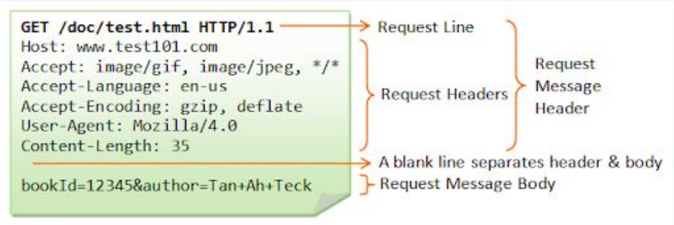
A. Application Layer

- **HTTP**: HTTP messages are generally of 2 types: requests and responses. The generic *start line* that begins all HTTP request messages is called a *request line* whereas in the case of HTTP response messages it is *status line*. The formal syntax for the **request line** is: <METHOD> <request-uri> <HTTP-VERSION>. The formal syntax of **status line** is: <HTTP-VERSION> <status-code> <reason-phrase>. They are followed by headers which are grouped according to their contexts as *General Headers*, *Request Headers*, *Entity Headers, Response Header*. Headers can also be grouped according to how proxies handle them: End-to-end headers and Hop-by-hop headers. Headers are followed by the message body. Headers contain the information like host name, user-agent, content length, content type, cookies requests, proxy-authentication requests, etc.



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

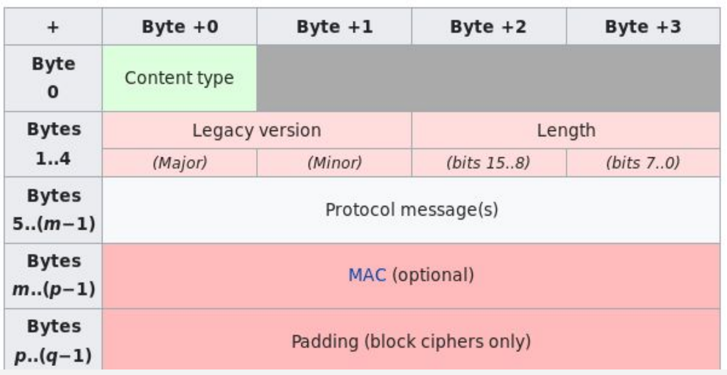


```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

B. Session Layer

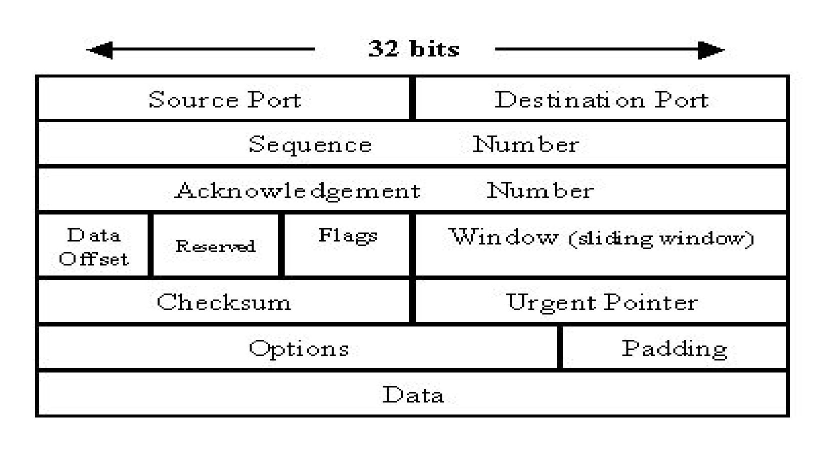
- **TLS**: The TLS protocol sits between the Application Layer and the Transport Layer. It provides security in communication between 2 hosts. The general format of all TLS records is as given. Each record consists of a header, followed by the data. Header has **content type** which is one from ChangeCipherSpec, Alert, Handshake, Application and Heartbeat. Therefore **protocol message** changes according to the content type. **Legacy version** identifies the major and minor version of TLS prior to TLS 1.3. **Length** tells the length of application data, excluding protocol header & including MAC & padding trailers.



+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	Legacy version		Length	
	(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Bytes 5..(*m*−1)	Protocol message(s)			
Bytes *m*..(*p*−1)	MAC (optional)			
Bytes *p*..(*q*−1)	Padding (block ciphers only)			

C. Transport Layer

- **TCP**: The TCP packet format consists of these fields: **Source Port** and **Destination Port** (16 bits each) identify the end points of the connection. **Sequence Number** (32 bits) specifies the number assigned to the first byte of data in the current message. **Acknowledgement Number** (32 bits) contains the value of the next sequence number that the sender of the segment is expecting to receive, if the ACK control bit is set. **Data Offset** (4 bits): Specifies the size of the TCP header in 32-bit words. **Reserved data** (3 bits) in TCP headers always has a value of zero. This field serves the purpose of aligning the total header size as a multiple of four bytes. **Flags field** (6 bits) contains the various flags(eg., URG, ACK, PSH, RST, SYN, FIN). **Window Size** (16 bits): The size of the receive window, which specifies the number of window size units (by default, bytes) (beyond the segment identified by the sequence number in the acknowledgment field) that the sender of this segment is currently willing to receive. **Checksum**(16 bits) is used to check if the header was damaged in transit. **Urgent pointer field** (16 bits) if the URG flag is set, then this field is an offset from the sequence number indicating the last urgent data byte. **Options** have up to three fields: Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable). The TCP header **padding** is used to ensure that the TCP header ends, and data begins, on a 32 bit boundary. **Data field** (variable length) contains upper-layer information.

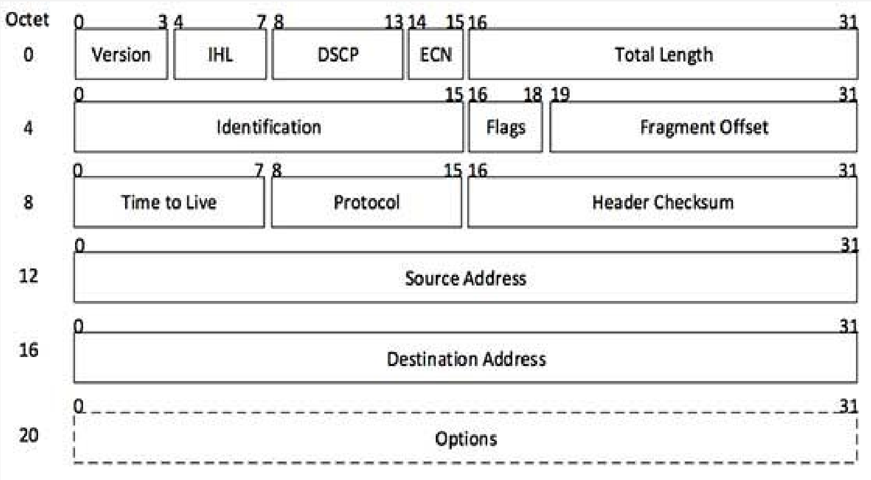


32 bits

Source Port		Destination Port	
Sequence Number			
Acknowledgement Number			
Data Offset	Reserved	Flags	Window (sliding window)
Checksum		Urgent Pointer	
Options			Padding
Data			

D. Network Layer

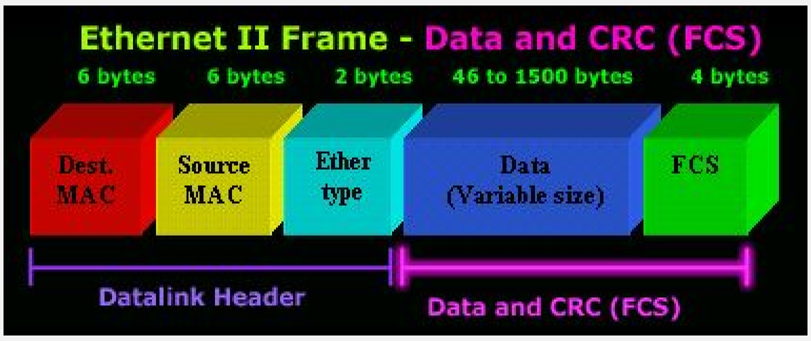
- **IPv4**: An Internet Protocol version 4 packet header (IPv4 packet header) contains application information, including usage and source/destination addresses. IPv4 packet headers contain 20 bytes of data and are normally 32 bits long. Detailed Header field description is:



[Image: IP Header]

Version (4 bits): This contains the Internet header format and uses only four packet header bits. **Internet header length (IHL)** (4 bits): This field specifies the size of of the header in the multiples of 32 bits The minimum is 5 which means length of 5 x 32 bits=160 bits. **Differentiated Services Code Point** (6 bits): this is Type of Service. **Explicit Congestion Notification** (2 bit): It carries information about the congestion seen in the route. **Total Length** (16 bits): defines entire packet size.**Identification** (16 bits): For uniquely identifying the group of fragments of a single IP datagram.**Flags** (3 bits): used to control or identify fragments. **Fragment offset** (13 bits): This offset tells the exact position of the fragment in the original IP Packet. **Time to Live (TTL)** (8 bits): This contains the total number of routers allowing packet pass-through. **Protocol** (8 bits): Tells the Network layer at the destination host, to which Protocol this packet belongs. **Header checksum** (16 bits): It checks and monitors communication errors. **Source and destination address** (32 bits each): It stores source and destination IP address respectively. **Options**: This is the last packet header field and is used for additional information. When it is used, the header length is greater than 32 bits.

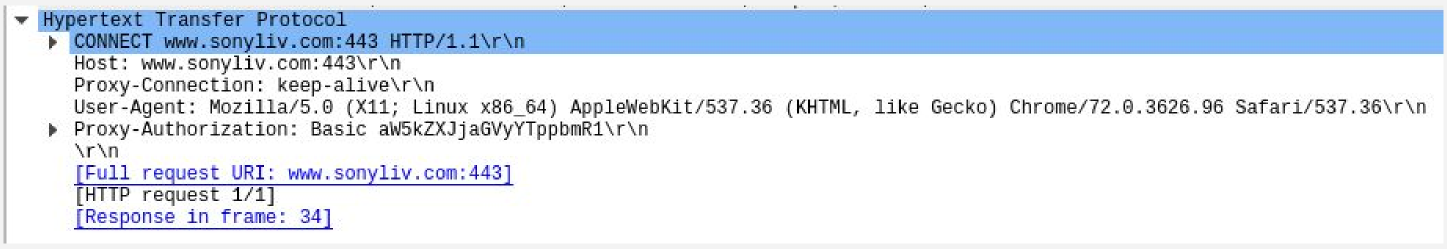
E. Data Link Layer



- **Ethernet 2**: Data link Protocols are point to point protocols. Ethernet 2 frame format is described below. (It is little different from Ethernet Frame Format): **Destination Address** (first 6 bytes): specifies to which adapter the data frame is being sent. **Source Address** (6 bytes): specifies from which adapter the message originated. **Ethertype** (2 byte): identifies upper layer protocol encapsulated by frame data. **Data** (46-1500 byes): consists of upper layer headers such as TCP/IP or IPX and then the actual user data. **FCS** (last 4 bytes): Frame Check Sequence or CRC is used to check if there is an error in received packet.

Answer 2. Most of the fields are explained in previous questions. Fields left are explained in this question.

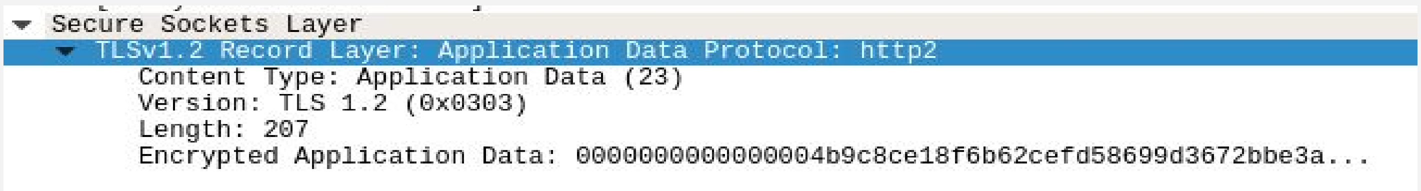
A. HTTP:



```
▼ Hypertext Transfer Protocol
  ▶ CONNECT www.sonyliv.com:443 HTTP/1.1\r\n
    Host: www.sonyliv.com:443\r\n
    Proxy-Connection: keep-alive\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36\r\n
  ▶ Proxy-Authorization: Basic aW5kZXJjaGVyYTppbmR1\r\n
    \r\n
    [Full request URI: www.sonyliv.com:443]
    [HTTP request 1/1]
    [Response in frame: 34]
```

Request Method	HTTP CONNECT (client asks an HTTP Proxy server to tunnel the TCP connection to the desired destination)
HOST	www.sonyliv.com:443 (domain name of the server (for virtual hosting), and the TCP port number on which the server is listening)
Proxy-Connection	Keep-alive (Control options for the current proxy connection)
User-Agent	Contains a characteristic string that allows the network protocol peers to identify the application type, OS, software vendor or software version of the requesting software user agent
Proxy-Authorization	Contains the credentials to authenticate a user agent to a proxy server.

B. TLS:



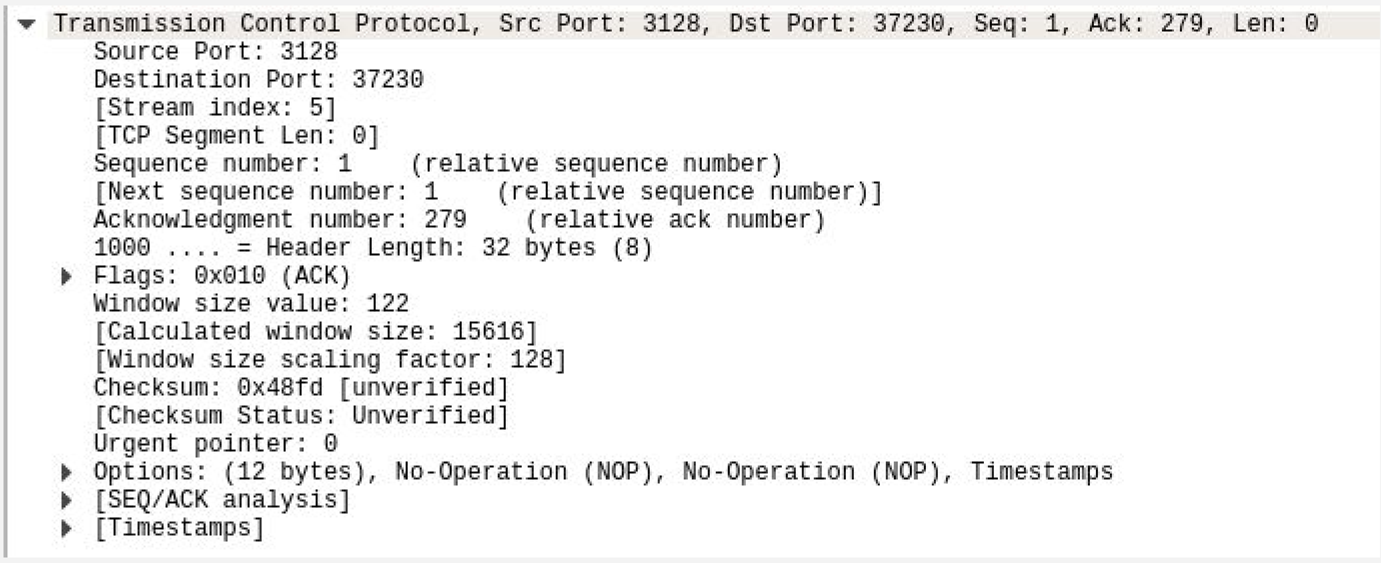
```
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Application Data Protocol: http2
        Content Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 207
        Encrypted Application Data: 00000000000000004b9c8ce18f6b62cefd58699d3672bbe3a...
```

Content Type	Application Data (Protocol type contained in this record - 23 is identifier for same)
Legacy Version	1.2 (version of TLS prior to TLS 1.3)
Length	207 (Length of application data, excluding protocol header & including MAC & padding trailers)
Encrypted Data	Encrypted form of the data that is being send.

C. TCP:

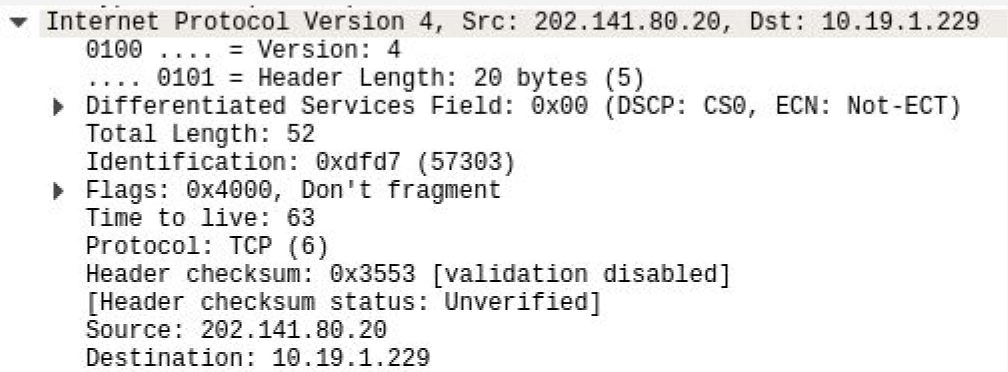
Source Port:	3128 (Proxy Server Port)	Urgent Pointer:	0 (i.e., not set)
Destination Port:	37230 (Local port opened by browser)	Flag:	ACK flag is set
Seq Number:	1 (Relative to TCP connection)	Window Size:	122 bytes

Acknowledgement Number:	279
Header Length (in multiples of 4):	8 (length of packet: 32 bytes)
CHECKSUM:	0x48fd (for error correction)



```
▼ Transmission Control Protocol, Src Port: 3128, Dst Port: 37230, Seq: 1, Ack: 279, Len: 0
     Source Port: 3128
     Destination Port: 37230
     [Stream index: 5]
     [TCP Segment Len: 0]
     Sequence number: 1    (relative sequence number)
     [Next sequence number: 1    (relative sequence number)]
     Acknowledgment number: 279    (relative ack number)
     1000 .... = Header Length: 32 bytes (8)
   ▶ Flags: 0x010 (ACK)
     Window size value: 122
     [Calculated window size: 15616]
     [Window size scaling factor: 128]
     Checksum: 0x48fd [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
   ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
   ▶ [SEQ/ACK analysis]
   ▶ [Timestamps]
```

D. IPv4:

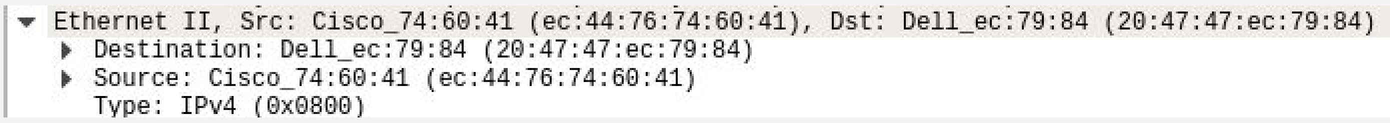


```
▼ Internet Protocol Version 4, Src: 202.141.80.20, Dst: 10.19.1.229
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 52
     Identification: 0xdfd7 (57303)
   ▶ Flags: 0x4000, Don't fragment
     Time to live: 63
     Protocol: TCP (6)
     Header checksum: 0x3553 [validation disabled]
     [Header checksum status: Unverified]
     Source: 202.141.80.20
     Destination: 10.19.1.229
```

Version	4 (IPv4)
Header Length (in multiples of 4 bytes)	5 (and hence size is 5*4=20 bytes)
Total Length	52 bytes
CHECKSUM	0x3553 (to check errors)
Protocol	TCP (protocol is used in above layer)

DSCP	CS0 (meaning standard service and have undifferentiated applications)
ECN	Not-ECN Capable Transport (Not-ECT)
TTL	63 (This packet can go at max these many hops)
Flags	Don't Fragment (If the DF flag is set, and fragmentation is required to route the packet, then the packet is dropped.)
Source	202.141.80.20 (Proxy server's IP)
Destination	10.19.1.229 (My IP)

E. Ethernet 2:

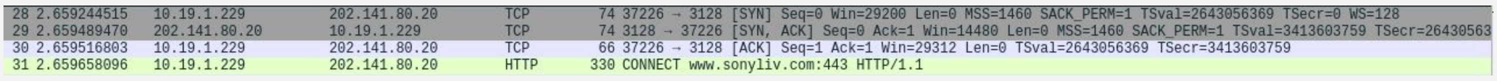


```
▼ Ethernet II, Src: Cisco_74:60:41 (ec:44:76:74:60:41), Dst: Dell_ec:79:84 (20:47:47:ec:79:84)
   ▶ Destination: Dell_ec:79:84 (20:47:47:ec:79:84)
   ▶ Source: Cisco_74:60:41 (ec:44:76:74:60:41)
     Type: IPv4 (0x0800)
```

Destination MAC	20:47:47:ec:79:84 (MAC address of my ethernet card by manufacturer DELL)
Source MAC	ec:44:76:74:60:41 (MAC address of source)
Type	IPv4 (specifies upper layer protocol used is IPv4)

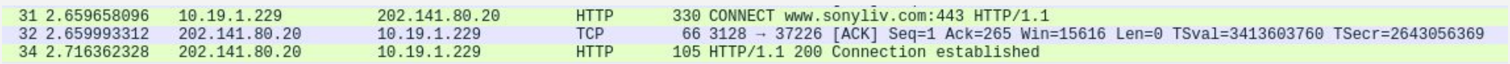
Answer 3. The following exchange of messages were observed in different cases:

- **Connecting Website** (www.sonyliv.com):
  TCP Connection Establishment: To connect to the website, it requires TCP connection. For connection establishment TCP does 3 way handshake with the destination server as described. The client first sends packet with SYN flag set requesting the server to synchronize with provided sequence number. Then server (through our proxy server) sends packet with SYN and ACK flag set having the acknowledgement number one more than the sequence number sent by the client (as it represents the next packet number it is expecting) and having some random sequence number. Finally, client sends back packet with ACK and SYN flags set to the server having sequence number set to received ack value and ack number set to one more than the received sequence number.



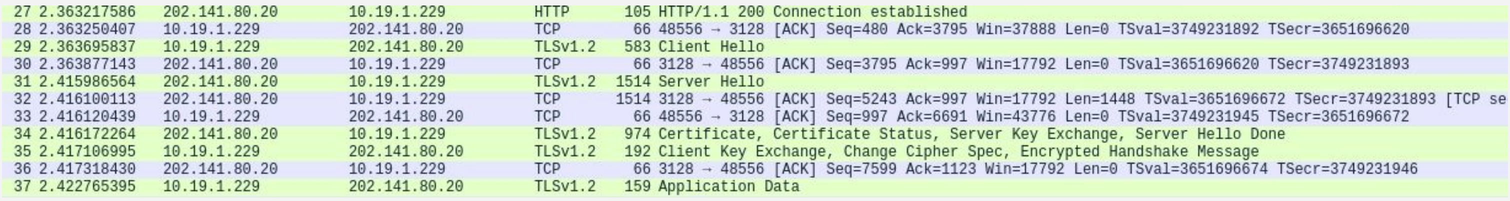
```
28 2.659244515   10.19.1.229        202.141.80.20      TCP       74 37226 → 3128 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2643056369 TSecr=0 WS=128
29 2.659489470   202.141.80.20      10.19.1.229        TCP       74 3128 → 37226 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=3413603759 TSecr=26430563
30 2.659516803   10.19.1.229        202.141.80.20      TCP       66 37226 → 3128 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2643056369 TSecr=3413603759
31 2.659658096   10.19.1.229        202.141.80.20      HTTP     330 CONNECT www.sonyliv.com:443 HTTP/1.1
```

  HTTP Connection Establishment: HTTP connection request is send as soon as TCP connection takes place. The client sends its requests and then wait for server to respond. The server processes the request and sends back its acknowledgement with the status code of connection and appropriate data.



```
31 2.659658096   10.19.1.229        202.141.80.20      HTTP     330 CONNECT www.sonyliv.com:443 HTTP/1.1
32 2.659993312   202.141.80.20      10.19.1.229        TCP       66 3128 → 37226 [ACK] Seq=1 Ack=265 Win=15616 Len=0 TSval=3413603760 TSecr=2643056369
34 2.716362328   202.141.80.20      10.19.1.229        HTTP     105 HTTP/1.1 200 Connection established
```

- **Handshaking after HTTP connection (TLS handshaking)**:
  After the HTTP connection is established, TLS handshake enables the TLS client and server to establish the secret keys with which they communicate. They are many ways to do the same. Example TLS handshake is described below (it may be change depending on various conditions.):
  - The client sends a "client hello" message that lists cryptographic information such as the TLS version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations.
  - The server responds with a "server hello" message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string.
  - The server may sends its Certificate message and ServerKeyExchange message.
  - The server sends a CertificateRequest message, to request a certificate from the client so that the connection can be mutually authenticated. Then server sends .ServerHelloDone message, indicating its done with handshake negotiation.
  - The client responds with ClientKeyExchange, which may contain a PreMasterSecret, public key, or nothing.
  - he client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret".
  - The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." Client sends an authenticated and encrypted Finished message.
  - Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)." The server sends its authenticated and encrypted Finished message.
  - At this point, the "handshake" is complete and the application protocol is enabled. Application messages exchanged between client and server will also be authenticated and optionally encrypted like in their finished message.

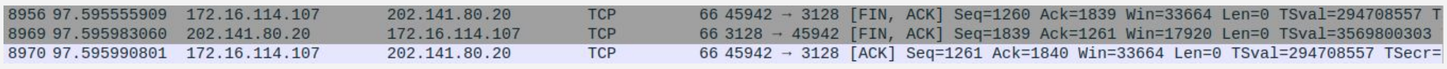


```
27 2.363217586   202.141.80.20      10.19.1.229        HTTP      105 HTTP/1.1 200 Connection established
28 2.363250407   10.19.1.229        202.141.80.20      TCP        66 48556 → 3128 [ACK] Seq=480 Ack=3795 Win=37888 Len=0 TSval=3749231892 TSecr=3651696620
29 2.363695837   10.19.1.229        202.141.80.20      TLSv1.2   583 Client Hello
30 2.363877143   202.141.80.20      10.19.1.229        TCP        66 3128 → 48556 [ACK] Seq=3795 Ack=997 Win=17792 Len=0 TSval=3651696620 TSecr=3749231893
31 2.415986564   202.141.80.20      10.19.1.229        TLSv1.2  1514 Server Hello
32 2.416100113   202.141.80.20      10.19.1.229        TCP      1514 3128 → 48556 [ACK] Seq=5243 Ack=997 Win=17792 Len=1448 TSval=3651696672 TSecr=3749231893 [TCP se
33 2.416120439   10.19.1.229        202.141.80.20      TCP        66 48556 → 3128 [ACK] Seq=997 Ack=6691 Win=43776 Len=0 TSval=3749231945 TSecr=3651696672
34 2.416172264   202.141.80.20      10.19.1.229        TLSv1.2   974 Certificate, Certificate Status, Server Key Exchange, Server Hello Done
35 2.417106995   10.19.1.229        202.141.80.20      TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
36 2.417318430   202.141.80.20      10.19.1.229        TCP        66 3128 → 48556 [ACK] Seq=7599 Ack=1123 Win=17792 Len=0 TSval=3651696674 TSecr=3749231946
37 2.422765395   10.19.1.229        202.141.80.20      TLSv1.2   159 Application Data
```

- **Play and Pause of live stream video**:
  Pausing live video had **no handshaking sequence**. In this case, server continuously sends data to client irrespective of whether live streaming is paused or not. And it is the job of browser to not show the data that are arriving. And hence when we play the paused video, it doesn't continue from where we paused. It shows live video. This confirms our observation that live streaming doesn't matter if the video is paused. Note: Further confirmation of same was done by inspecting elements on sonyliv website and checking data under networks tab. We could see source ip from where video was arriving. Then data was taken by vpn service so that we could see actual ips in wireshark from where data is arriving. Then source ip was filtered and checked if there was some change on pause and play of video.

Moreover, in the networks tab of inspect element we could still see that data was arriving from source ip even when video was paused.

- **Connection Termination**:
  On closing the browser, **4 way handshaking** occurs. First, the client transmits FIN packet, which is acknowledged by the server. Then server also sends the FIN packet that client acknowledges.



```
8956 97.595555909  172.16.114.107       202.141.80.20        TCP        66 45942 → 3128 [FIN, ACK] Seq=1260 Ack=1839 Win=33664 Len=0 TSval=294708557 T
8969 97.595983060  202.141.80.20        172.16.114.107       TCP        66 3128 → 45942 [FIN, ACK] Seq=1839 Ack=1261 Win=17920 Len=0 TSval=3569800303
8970 97.595990801  172.16.114.107       202.141.80.20        TCP        66 45942 → 3128 [ACK] Seq=1261 Ack=1840 Win=33664 Len=0 TSval=294708557 TSecr=
```

Answer 4.Reasons for protocols used by my application:

- **HTTP**:
  - HTTP functions as a request–response protocol in the client–server computing model.
  - HTTP (Hyper Text Transfer Protocol), is used for transferring information like document, file, image, video between computers over internet. Here, it is used to stream live sports match.
  - Its platform independent, which allows straight cross platform porting. Hence, we us this application in phones, tablets, laptops, etc.
  - HTTP support data compression using different algorithms, hence saving large amount of data used for streaming.
- **TLS**:
  - TLS is used for prevention of unwanted eavesdropping and modification on internet traffic.
  - As, we are using HTTP protocol, it has a disadvantage that it is not secure. Anyone, can see what is transferred in the network. To prevent this, TLS protocol is used as it encrypts data to and from the site to clients.
  - This also protects the integrity of the website by helping to prevent intruders tampering between the site and client browsing.
  - User's login and credentials details are protected as they are encrypted when they are send hence safeguarding privacy and security. If it was only for HTTP protocol then anyone eavesdropping our network can see our credential details.
- **TCP**:
  - It was majorly used protocol in communication with www.sonyliv.com.
  - TCP always guarantees that data reaches its destination and it reaches there without duplication.
  - It guarantees reliable data transfer by having handshaking protocol on connection establishment and connection termination.
  - It is interoperable, i.e., it allows cross-platform communications among heterogeneous networks.
  - It uses flow control, Error control and congestion control mechanisms.

- **Why use TCP over UDP?**:
  - TCP was being used instead of UDP on sonyliv.
  - UDP uses a simple connectionless communication model with a minimum of protocol mechanism.
  - Theoretically, we have learned that UDP is preferred for live video streaming as if the packet is lost there is no point for waiting to resend the packet lost because if we did that then we won't be live anymore. But, still various sites use TCP protocol for the same.
  - Firewalls (from enterprises, ISPs) often block UDP data for security reasons. Unlike TCP, UDP can relentlessly consume valuable bandwidth (whereas TCP uses algorithms for congestion control). Moreover, UDP data can't be throttled.
  - Sonyliv also provide services of live TV where one can watch Indian drama online. Now these live videos support options of playing and pausing live video and even allowed to rewind the video. This means video must be buffered giving TCP an edge. For handling UDP packets developers must program for the same whereas OS automatically handles TCP packets. Therefore, sonyliv may have use it as a common protocol for all media.
  - TCP is capable of multiplexing whereas UDP is not.
  - In order to deliver videos, platforms adopt/rent Content Delivery Networks (CDN). By checking the source ip of streaming video, we could check that the video on sonyliv was streamed from akamai which is a CDN. Most of the CDNs were originally and already configured to support web services as their primary course. Thus, streaming video over HTTP works out of the box without setting up dedicated servers, and most of the firewalls won't block HTTP traffic. Although in theory HTTP can be encapsulated in other protocols, but HTTP definition presumes and underlying and reliable transport layer protocol which again precludes UDP.
  - sonyliv provides option to change video quality. If it had used UDP as underlying protocol then it can't guarantee the quality of video displayed. To do so, we need a reliable transfer of video packets and hence favoring TCP.
- **IPv4**:
  - IPv4 is a connectionless protocol for use on packet-switched networks.
  - It delivers packets using IP headers from the source to the destination.
  - Existing in the network layer, IPv4 connection is hop to hop.

- **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.

Answer 5. Statistics from traces:

`

Property	**Time 1 (Room Night)**	**Time 2 (Room evening)**	**Time 3 (Lab)**	**Time 4 (VPN)**
Host A	10.19.1.229	10.19.1.229	172.16.114.107	10.8.0.2
Host B	202.141.80.20:3128	202.141.80.20:3128	202.141.80.20:3128	Multiple Ips
Throughput (A->B)(kbps)	54	40	39	44
Throughput (B->A)(kbps)	1382	1047	1623	983
RTT (in ms)	6.67	0.90	2.40	12
Avg. Packet Size	944.01	923.99	2262.20	762.04
No. of Packets lost	16 out of 28988 packets (0.055 %)	21 out of 26048 packets (0.08%)	6 out of 9005 packets (0.066%)	120 out of 15553 packets (0.771%)
TCP packets (A->B)	9516	8775	3961	6282
TCP packets (B->A)	19472	17273	5044	9271
UDP packets	0	0	0	0
Responses received wrt per request sent (packets B->A/packets A->B)	2.04	1.968	1.27	1.475

Observation:

- When taking data on VPN we observe more number of packets lost and also average RTT is high because connection goes through to VPN server.

Note:

- The data was taken after filtering packet related to my application only.
- Data was taken from visiting the website, opening the live video and waiting for some time till closing the browser window.
- UDP packets were not used by my application and hence count is 0.
- RTT is calculated by adding a custom field "tcp.analysis.ack_rtt" to the logs and then taking the average.
- No of packets lost is taken by filtering packets with "tcp.analysis.lost_segment" or "tcp.analysis.ack_lost_segment" which represents that some segments were lost in transmission and hence equal to packets lost.
- In case of multiple ips, throughput was calculated combined from all the ips and not only from where the video stream was coming.

Answer 6. There are 2 different scenarios in which data has been taken.

- **Behind Campus Proxy Server**:
  - When data was taken behind proxy authenticated server we could see data arriving to my pc from campus proxy server 202.141.80.20 and port number 3128. Similarly, all my requests were sent to campus proxy server 202.141.80.20 and port number 3128.
  - It is because behind a proxy server, the HTTP GET requests are first sent to the campus proxy server from where they are forwarded to destination host. Any responses received take the same reverse path.
- **On VPN**:
  - We we able to notice multiple IPs from where data was arriving to my pc.
  - It could be because they may have deployed load balancing on servers, where data is sent from various IP in most efficient manner.
  - The resources from where the data is arriving may be distributed. By looking at the hostname of IP from where data was arriving, we know that live video came from CDN servers. Hence, it is possible that video is streamed from CDN servers and other site's data arrives from other servers.
  - Round Robin DNS mechanism for faster fetching of relevant pages by balancing the page requests across many servers may lead to multiple IP.