CS349: NETWORKS LAB

ASSIGNMENT 2

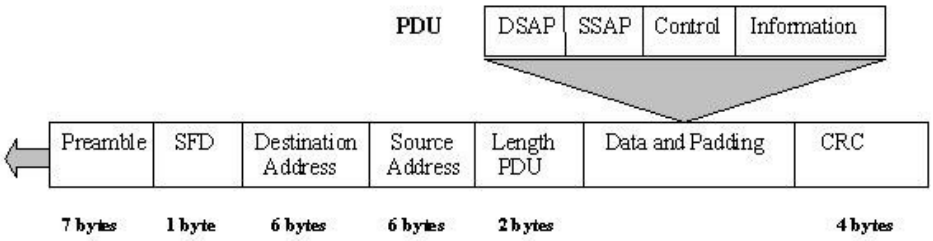
Kartik Sethi 170123057
3rd February 2020

One drive link for trace records:

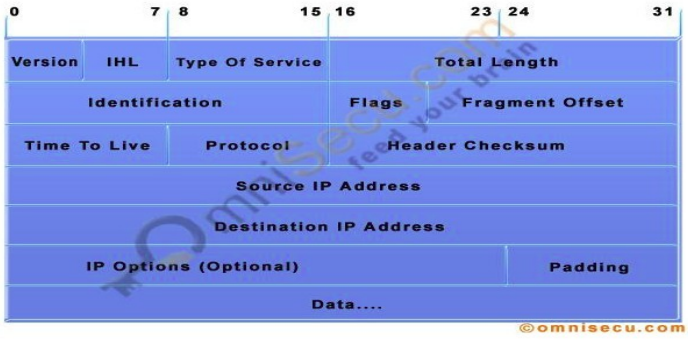
https://iitgoffice-my.sharepoint.com/:f:/g/personal/sethi170121021_iitg_ac_in/Es2ouWHg7IBBvC2gv657EScBIlYYfnd14Jjf9krj0XKezw?e=P5cKO0

Q1

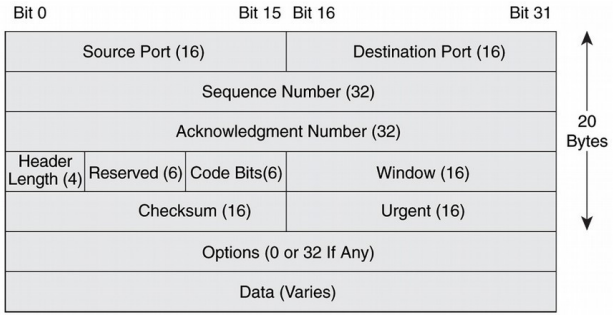
a) **Ethernet (Physical and Data Link layers)**: The Ethernet frame starts with a Preamble and a SFD (start frame delimiter), both of which work at the physical layer. The header contains both the source as well as destination MAC addresses, after which the payload of the frame is present. The last field corresponds to the Frame Check Sequence which is basically a Cyclic Redundancy Check (CRC) for the detection of errors.



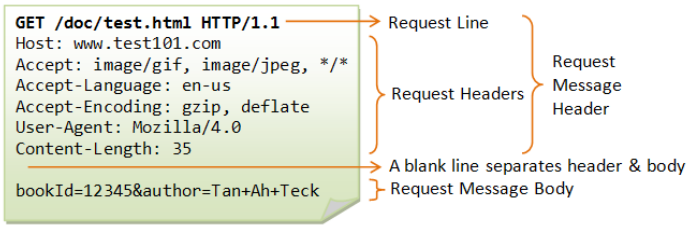
b) **Internet Protocol Version 4 (Network layer)**: It provides the logical connection between network devices by providing identification for each device. IP provides a mechanism to uniquely identify hosts by an IP addressing scheme which consists of 32-bit logical addresses. The IPv4 packet header consists of 14 fields, of which 13 are required. The 14th field is optional and aptly named: options. It has relevant information including version number (in this case 4), total length of the entire packet, TTL, protocol, source and destination address and so on.



c) **Transmission Control Protocol (Transport layer)**: TCP is a connection-oriented Layer 4 protocol that provides full-duplex, acknowledged, and flow-controlled service to upper-layer protocols. It moves data in a continuous, unstructured byte stream. Sequence numbers identify bytes within that stream. TCP can also support numerous simultaneous upper-layer conversations. A TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields and an optional extension field. It includes a number of fields including source and destination addresses, sequence, and acknowledgment number and checksum.



d) **Hypertext Transfer Protocol (Application layer)**: HTTP is based on the client-server architecture model and is a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection. An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS, etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages. As seen from the following figure, there exists many fields like Accept which accept various types of multimedia, Accept-Language, Accept-Encoding, User-Agent and Content-Length.



Q2

a) **APPLICATION LAYER**: for HTTP we can see below

```
Hypertext Transfer Protocol
  [truncated]GET /v1/segment/CpQE9bCpStKdsQYrK1ejItxallNlfZLFjO65hBIr1OmWxdBhzrcmfaAJKJw18kCsqM4kmOoDhCe6M6dSo_kPbq28lw
  Host: video-edge-c55be4.sin01.abs.hls.ttvnw.net\r\n
  Connection: keep-alive\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36\
  Accept: */*\r\n
  Origin: https://www.twitch.tv\r\n
  Sec-Fetch-Site: cross-site\r\n
  Sec-Fetch-Mode: cors\r\n
```

Request method: CONNECT
Request URL: www.twitch.tv (Port 443 as is usually used for SSL)
User-Agent: Mozilla Firefox browser running on Ubuntu (Linux)
Connection: Keep-alive denotes persistence of TCP connection used by HTML.

For TLS v1.3 we can see the image below

```
Transport Layer Security
  TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
```

b) **TRANSPORT LAYER**
Source Port: Port number of source device
Destination Port: Port number of destination device
Length: Size of TCP Packet
Sequence number : Used to correctly order the packets
ACK number : Expected next

```
Transmission Control Protocol, Src Port: 50968, Dst Port: 443, Seq: 50020, Ack: 27947879, Len: 1177
  Source Port: 50968
  Destination Port: 443
  [Stream index: 319]
  [TCP Segment Len: 1177]
  Sequence number: 50020    (relative sequence number)
  [Next sequence number: 51197    (relative sequence number)]
  Acknowledgment number: 27947879    (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
  Window size value: 7722
  [Calculated window size: 988416]
  [Window size scaling factor: 128]
  Checksum: 0x5999 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
```

packet number
Flags: 0x010 here denotes that this is an ACK flag
Window size: Number of packets sent before waiting for ACK
Checksum: The value 0xe780 is used for error correction and detection.
c) **NETWORK LAYER**

```
Internet Protocol Version 4, Src: rust-bucket.local (192.168.43.196), Dst: video-edge-c55be4.sin01.abs.hls.ttvnw.net (45.113.129.
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1229
  Identification: 0x3ffe (16382)
▸ Flags: 0x4000, Don't fragment
  ...0 0000 0000 0000 = Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0x5aee [validation disabled]
  [Header checksum status: Unverified]
  Source: rust-bucket.local (192.168.43.196)
  Destination: video-edge-c55be4.sin01.abs.hls.ttvnw.net (45.113.129.97)
```

Header length: Number of 4 byte words in the header is 5 (20 byte header)
Total length: Size of the packet is 528 bytes.
Identification: Value (0x6551) is used in cases the datagram fragments to transmit
Flag: (0x4000) means Don’t Fragment is set and all nodes through which the datagram passes are asked not to fragment it.
TTL: Maximum number of hops (63) the packet can make before dying out.
Header checksum: Used for error detection and correction
Source IP: IP of the sender, here it is 202.141.80.20 (ProxyServer of IITG)
Destination IP: IP of the receiver, here it is 10.11.12.13 (my device’s IP).
d) **ETHERNERT LAYER**

```
▾ Ethernet II, Src: rust-bucket.local (94:65:9c:ae:19:83), Dst: _gateway (5e:89:90:ad:b3:99)
  ▸ Destination: _gateway (5e:89:90:ad:b3:99)
  ▸ Source: rust-bucket.local (94:65:9c:ae:19:83)
    Type: IPv4 (0x0800)
```

Source and Destination MAC addresses uniquely (globally unique) identify the Network Interface Controllers present in the devices.Source device is my Acer Laptop(Rust-bucket) and the MAC address reflects the router where the packet is headed originating from my laptop.

Q3

Application Layer : HTTP and TLSv1.2 (SSL).
HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted. TLSv1.2 is a Secure Sockets Layer protocol used to encrypt the application data and prevent hackers who snoop packets to gain access to any confidential information such as login credentials for a user.
Transport Layer : TCP
TCP is a connection-oriented reliable data transfer protocol and involves server-client handshaking mechanism. It also helps in proper error handling and flow control which is important for a video streaming website such as Twitch.
Network Layer : IPv4
IPv4 is a connectionless protocol for use in a packet-switched network. It operates on best-effort delivery model - neither does it guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. These aspects, including data integrity are ensured by TCP.
Link Layer : Ethernet II
Ethernet II is a reliable link layer protocol with a well defined preamble for synchronization and

CRC field for error detection and handling. It ensures reliable data transfer between the network devices (i.e on a link) on path of the packet.

Q4

DNS Query: First, when the site is loaded, DNS querying is done by the browser. A series of messages are exchanged (Query and query responses), so that the browser may learn the IP address of twitch.tv (and also twitch CDN).

190	3.100325922	rust-bucket.local	_gateway	DNS	69	Standard query 0xfd54 A twitch.tv
194	3.174069517	_gateway	rust-bucket.local	DNS	133	Standard query response 0xfd54 A twitch.tv A 151.101.130.167 A 151.101…

TCP Handshake Protocol: To establish a TCP Connection, the client sends a SYN to the server,which returns an ACK for it and SYN to connect to the client and then client acknowledges SYN of server. This is also known as the three-way TCP Connection Handshake.

1178…	4830.7744209…	rust-bucket.local	usher.ttvnw.net	TCP	74	33044 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=37…
1178…	4831.0234513…	rust-bucket.local	usher.ttvnw.net	TCP	74	33046 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=37…
1178…	4831.0252190…	usher.ttvnw.net	rust-bucket.local	TCP	74	443 → 33044 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1400 SACK_PERM…

Application Layer Handshake: Application layer handshake begins with the client sending a “Client Hello” to the server to which server responds with a “Server Hello” to complete the handshake. This happens when we first connect to Twitch.

223	3.366725714	rust-bucket.local	twitch.tv	TLSv1.2	583	Client Hello
255	3.584874541	twitch.tv	rust-bucket.local	TCP	66	443 → 47816 [ACK] Seq=1 Ack=518 Win=29696 Len=0 TSval=199505273 TSecr…
266	3.605449622	twitch.tv	rust-bucket.local	TLSv1.2	1454	Server Hello

Live Streaming Video: On starting a live stream, our PC makes a HTTP GET request for a live stream, each of which is ACKnowledged from my PC. Since the video data is big in size it is broken in to multiple TCP segments, indicated by the TCP and TLS segments (of a reassembled PDU), which are ACKs but with non-zero lengths, indicating that they are carrying application data. Once all the packets arrive the segments are reassembled and fed to the application layer, indicated by the HTTP message sent to our PC with the live stream as payload (application/vnd.apple.mpegurl indicates the HLS protocol).

460	10.852646198	rust-bucket.local	usher.ttvnw.net	HTTP	354	GET /api/channel/hls/dreamleague.m3u8?allow_source=true&fast_bread=tr…
492	11.453925965	usher.ttvnw.net	rust-bucket.local	HTTP	97	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
529	12.215872591	rust-bucket.local	video-weaver.sin01.…	HTTP	1334	GET /v1/playlist/CpwEm0c8yz6UxSv22kxPaMV7oMU-xW-lSU96xL1m4IxcP_fixmgx…
591	12.698580100	rust-bucket.local	video-weaver.sin01.…	HTTP	1331	GET /v1/playlist/CpoEJDg5Z9G56af-wGuwe94cdlqAeGXBnroyo0zYbgNtN1FfnUSX…
625	13.386129878	rust-bucket.local	video-edge-c55be4.s…	HTTP	1243	GET /v1/segment/CpQEqN3qT_I-5YhO8d0uPspTwbC6w66n30Jh9zrruOpaWV5nTIIU_…
724	14.295285803	video-edge-c55be4.s…	rust-bucket.local	TLSv1.3	275	HTTP/1.1 200 OK
726	14.335487116	rust-bucket.local	video-edge-c55be4.s…	HTTP	1243	GET /v1/segment/CpQEDYUmh4_8FEX_2V8jhBNMnVCN8ohmpgB23U0dHXxxN_SbFdj3k…
728	14.690188689	rust-bucket.local	video-weaver.sin01.…	HTTP	1331	GET /v1/playlist/CpoEJDg5Z9G56af-wGuwe94cdlqAeGXBnroyo0zYbgNtN1FfnUSX…
792	14.795078707	video-edge-c55be4.s…	rust-bucket.local	TLSv1.3	770	HTTP/1.1 200 OK

Pausing Video: The client sends a (FIN, ACK) asking the server to stop the data transmission which is then acknowledged back by the server which in turn sends a (FIN, ACK) to the client and client acknowledges it to complete the four-way TCP Termination handshake. However, the server continues to send TCP packets like before.

246	7.018702441	rust-bucket.local	video-edge-c55be4.s…	TCP	66	52056 → 443 [ACK] Seq=4113 Ack=223449 Win=276736 Len=0 TSval=39450177…
247	7.202898708	rust-bucket.local	video-weaver.sin01.…	HTTP	1331	GET /v1/playlist/CpoEJDg5Z9G56af-wGuwe94cdlqAeGXBnroyo0zYbgNtN1FfnUSX…
248	7.260511663	rust-bucket.local	video-weaver.sin01.…	TCP	66	54150 → 443 [FIN, ACK] Seq=4368 Ack=20569 Win=64128 Len=0 TSval=15275…
249	7.465521087	rust-bucket.local	151.101.38.167	TCP	66	42968 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=2100825242 TSecr=19…
250	7.468685051	video-weaver.sin01.…	rust-bucket.local	TLSv1.2	2991	[TLS segment of a reassembled PDU]

Closing Twitch: The client sends a (FIN, ACK) asking the server to stop the data transmission which is then acknowledged back by the server which in turn sends a (FIN, ACK) to the client and client acknowledges it to complete the four-way TCP Termination handshake.

117	7.598317756	rust-bucket.local	ec2-52-89-127-159.u…	TCP	66	49348 → 443 [FIN, ACK] Seq=38 Ack=1 Win=501 Len=0 TSval=3904851525 TS…
118	7.598602063	rust-bucket.local	ec2-54-187-65-211.u…	TLSv1.2	103	Application Data
119	7.598884657	rust-bucket.local	ec2-54-187-65-211.u…	TCP	66	41276 → 443 [FIN, ACK] Seq=38 Ack=190 Win=501 Len=0 TSval=3797520298 …
120	7.608310667	rust-bucket.local	usher.ttvnw.net	TCP	66	50576 → 443 [FIN, ACK] Seq=2189 Ack=138 Win=64128 Len=0 TSval=1116147…
121	7.625188690	usher.ttvnw.net	rust-bucket.local	TCP	66	443 → 50578 [ACK] Seq=138 Ack=569 Win=30208 Len=0 TSval=1376009038 TS…
122	7.710519926	usher.ttvnw.net	rust-bucket.local	TCP	1454	443 → 50576 [ACK] Seq=138 Ack=2189 Win=35840 Len=1388 TSval=256810907…
123	7.710644267	rust-bucket.local	usher.ttvnw.net	TCP	54	50576 → 443 [RST] Seq=2189 Win=0 Len=0

Q5

	HOSTEL		
Time	00:47	10:27	5:30
# Packets Lost	0	0	0
Average packet size	714	739	766
Throughput	1791183 B	2473777 B	4358318 B
# TCP Packets	2207	2951	5401
# UDP Packets	0	387	275
# Responses per request sent	37/37	29/50	46/84
Avg. RTT (in ms)	93.59827807	109.0423395	36.1980002

Q6.

On checking the TLS packets, I found different connection with different servers
Some of them include:
1. www.google.com : Google is the default search engine on chrome. Hence, my local machine connects to Google so that it can display search suggestions while I am typing the website name in the url bar.
2. adservice.google.com: Twitch shows ads to generate revenue, and those ads are served by Google AdServices.
3. Grammarly : I've enabled the grammarly extension in the browser, and it shows up whenever text is being typed in the browser.
4. www.twitch.tv : This is the main twitch server which sends most of the data to the computer.
5. Since Twitch is primarily a video streaming website, it utilises multiple host server to serve requests. Some of those include pubsub-edge.twitch.tv, gql.twitch.tv and twitch.amazon.com. This is done for the sake of load-balancing and ensuring different points of failure in the video streaming service.