CS 349 Assignment-2 (Online Game)
Saurabh Bazari - 160101061

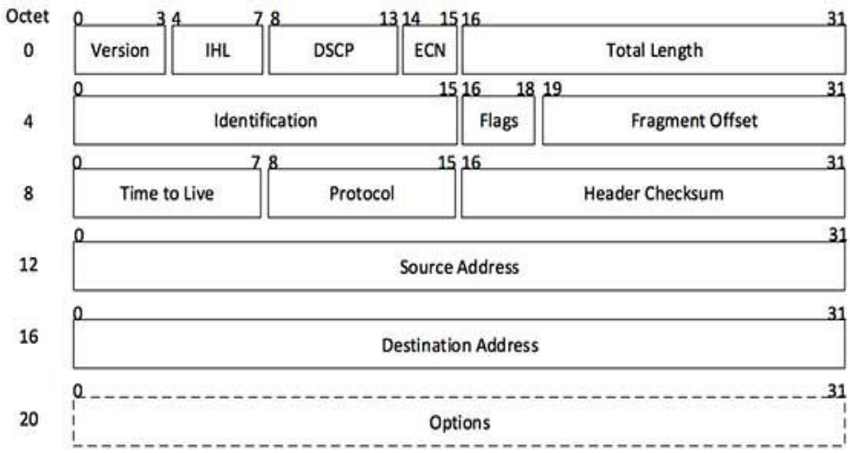
Game link: https://supermarioemulator.com/ 217.160.0.48
Drive link: https://drive.google.com/open?id=1ZqPLnDzxn56mLOL03gS9ul1OKrXTqjFN

1.

Link Layer: Ethernet(II)

Ethernet(II) is the most common local area networking technology. It has Preamble which is 56 bits of alternating 1's and 0's, **Destination** MAC Address, **Source** MAC Address, Type that identifies an upper layer protocol encapsulated by the frame data, Length of frame and Frame Checksum for error detection.

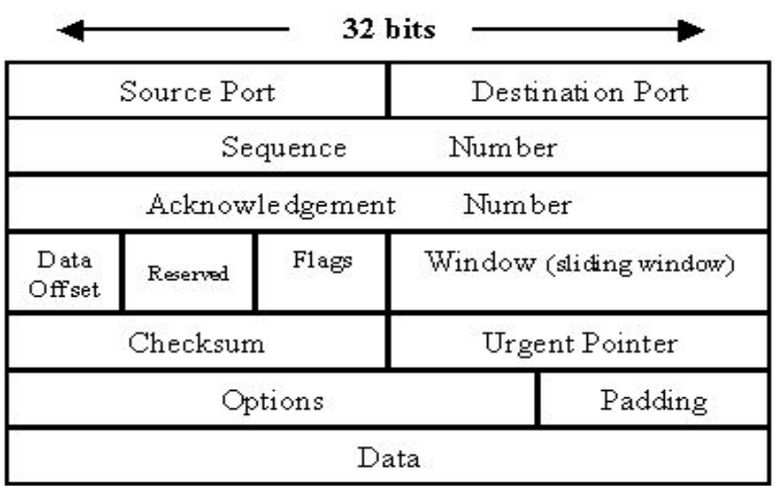
Network layer: IPv4



[Image: IP Header]

IPv4 is a connectionless protocol used in packet-switched layer networks. It provides a logical connection between network devices by providing identification for each device. **Version** - Version no. of Internet Protocol used. **Header length** - 4 bits field stores the number of 32-bit words in the entire IP header. **Differentiated Services Code Point** - this is Type of Service. **Explicit Congestion Notification** - It allows end-to-end notification of network congestion without dropping packets. **Total Length** - Length of entire IP Packet. **Identification** - Unique Packet Id for identifying the group of fragments of a single IP packet. **Flags** - 3 flags of 1 bit each: **reserved bit** (must be zero), **do not fragment** flag, **more fragments** flag (same order). **Fragment Offset** - This offset tells the exact position of the fragment in the original IP Packet. **Time to Live** - packet's lifetime (8 bits), It prevents the packet to loop through the network. **Protocol** - Name of the protocol to which the data is to be passed. **Header Checksum** - 16 bits header checksum for checking errors in the packet header. **Source** - 32-bit address of the Sender (or source) of the packet. **Destination** - 32-bit address of the Receiver (or destination) of the packet. **Options** - Optional information such as source route, record route.

Transport Layer: TCP



TCP header has ten required fields totaling 20 bytes (160 bits) in size. **Source** and **destination** TCP port numbers are the sources and destination port number. Message senders use **sequence numbers** to mark the ordering of a group of messages. Both senders and receivers use the **acknowledgment numbers** field to communicate the sequence numbers of messages that are either recently received or expected to be sent.

Data offset - field stores the total size of a TCP header in multiples of four bytes. **Flags** - URG, ACK, PSH, RST, SYN, FIN. **Window** size to regulate how much data they send to a receiver before requiring an acknowledgment in return. **Urgent pointer** points to the sequence number of the octet following the urgent data. **Optional** TCP data can be used to include support for special acknowledgment and window scaling algorithms.

Application Layer:

- Hypertext Transfer Protocol (HTTP)

Host - The domain name of the server (for virtual hosting), and the TCP port number on which the server is listening. **Connection** - Control options for the current connection and list of hop-by-hop request fields. **User-Agent** -The user agent string of the user agent. **Accept** - Media type that is(/are) acceptable for the response. **Referer** - This is the address of the previous web page from which a link to the currently requested page was followed. **Accept-Language**:-List of acceptable human languages for a response. **Accept-Encoding**:-List of acceptable encodings.

- Secure Sockets Layer(SSL) - TLSv1.2 Record Layer

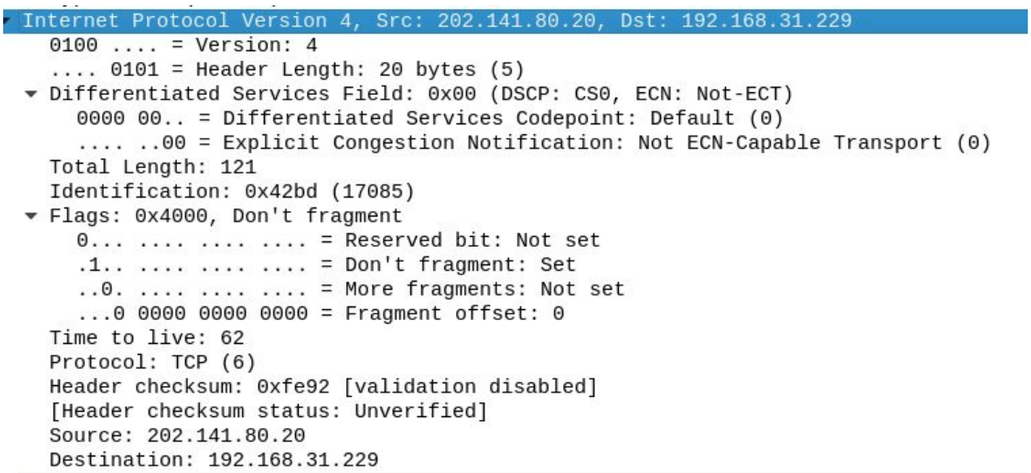
It can be used with any protocol that uses TCP as the transport layer. The basic unit of data in SSL is a record. Each record consists of a five-byte record header, followed by data. The header contains – **Record Type** can be of four types(Handshake, Change Cipher Spec, Alert, Application Data), **Record Version** is 16- byte value formatted in network order, **Record Length** is a 16-byte value of the entire header. **Encrypted Application Data:** original data that encrypted in another for security.

2. Observed Values :

- **Data link: Ethernet(II)**

The source is my router and destination is my laptop. The source is always Unicast.
In this case, the destination is also Unicast. In both of them, it is Globally Unique Address.

- **Network Layer: IPv4**



A version is always 4, Header length = 20B. No ECN. Total length is 121 bytes. Identification - 0X42bd (Unique ID). A packet is not fragmented because don't fragment bit is set. TTL = 62. Protocol - TCP. Checksum = 0xfe92. Source IP is my IP address(202.141.80.20) and a destination is 192.168.31.229.

- **Transport layer: TCP**

Source Port:- 3128 (One of the well-known ports). Destination Port:- 43896 (Port specific to the server hosting the site). Sequence Number:- 255. Acknowledgment Number:- 1360. Header Length:- 32 bytes. Flags:- 0x018 (Only ACK and PUSH flags are set). Window Size Value:- 139 (Size of the receiving window of the sender). Checksum:- 0xf9ad (unverified). Urgent Pointer:- 0 (If the value is greater than 0 the packet is prioritized over other packets).

```
Transmission Control Protocol, Src Port: 3128, Dst Port: 43896, Seq: 255, Ack: 1360, Len: 38
  Source Port: 3128
  Destination Port: 43896
  [Stream index: 10]
  [TCP Segment Len: 38]
  Sequence number: 255    (relative sequence number)
  [Next sequence number: 293    (relative sequence number)]
  Acknowledgment number: 1360    (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
▾ 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: 139
  [Calculated window size: 17792]
  [Window size scaling factor: 128]
  Checksum: 0xf9ad [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▸ [SEQ/ACK analysis]
▸ [Timestamps]
  TCP payload (38 bytes)
```

- **Application Layer: SSL**

```
▾ Hypertext Transfer Protocol
    [Proxy-Connect-Hostname: supermarioemulator.com]
    [Proxy-Connect-Port: 443]
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Application Data Protocol: http2
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 33
      Encrypted Application Data: cc3bf6c3253de5f9957cc4134dbcac607987edcc92a5f8ba...
```

HTTP part has the hostname and port and SSL part has a version = TLS 1.2 Record Layer, Application data protocol = http2, length = 33, content type = Application Data and encrypted application data is encrypted original data.

- **Application Layer: HTTP**

```
▾ Hypertext Transfer Protocol
  ▾ CONNECT supermarioemulator.com:443 HTTP/1.1\r\n
    ▾ [Expert Info (Chat/Sequence): CONNECT supermarioemulator.com:443 HTTP/1.1\r\n]
        [CONNECT supermarioemulator.com:443 HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: CONNECT
      Request URI: supermarioemulator.com:443
      Request Version: HTTP/1.1
  Host: supermarioemulator.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/71.0.3578.98 Safari/537.36\r\n
▸ Proxy-Authorization: Basic c2F1cmFiaC5iYXphcmk6cy5iYXphcmk=\r\n
  \r\n
  [Full request URI: supermarioemulator.com:443]
  [HTTP request 1/1]
  [Response in frame: 57]
```

The request method is connected to the request URL, which is a game website and the requested version is HTTP 1.1. The host is the game link. Proxy connection keeps alive and credentials like username and password are also added. User Agent - one of from Mozilla, AppleWebkit, Chrome, Safari. Wireshark also shows frame number having a response to this frame.

3. Different sequences of message exchanges by application:-

Functions: Various functions like **PLAY, Pause, Up, Down, Left, Right** (Arrow keys) also **level change** etc are available in the game. After repeating the experiments several times and carefully observing the requests. It was found that any functionalities do not create any kind of network requests. In approx. 40-45 seconds that the client sends request Keep-Alive. If the client ends the game then it requests FIN to the server. The reason is the following:- In case of some online gaming platforms to maintain the flow of the game or to avoid the lag in game, it downloads a javascript corresponding to those actions and whenever such action is performed that javascript is run internally. As the javascript runs internally it does not create any networking requests. Only the interaction with the other (online) players create network requests.

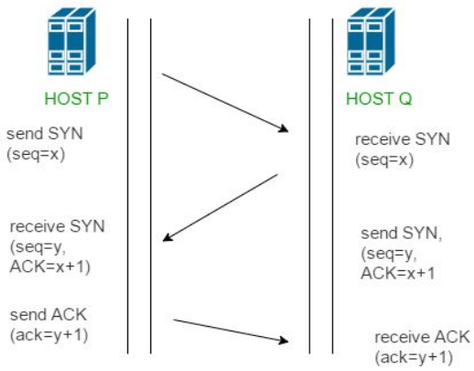
```
23.317476150  217.160.0.48   10.8.0.2          TCP      1396 443 → 48388 [ACK] Seq=1717 Ack
23.317515899  217.160.0.48   10.8.0.2          TLSv1.2   196 Application Data
23.317607858  10.8.0.2       217.160.0.48      TCP        52 48388 → 443 [ACK] Seq=217 Ack=
69.265697100  10.8.0.2       217.160.0.48      TCP        52 [TCP Keep-Alive] 48388 → 443 [
69.523703579  217.160.0.48   10.8.0.2          TCP        52 [TCP Keep-Alive ACK] 443 → 483
114.525696419 10.8.0.2       217.160.0.48      TCP        52 [TCP Keep-Alive] 48388 → 443 [
114.714959390 217.160.0.48   10.8.0.2          TCP        52 [TCP Keep-Alive ACK] 443 → 483
161.425741885 10.8.0.2       217.160.0.48      TCP        52 [TCP Keep-Alive] 48388 → 443 [
161.622888733 217.160.0.48   10.8.0.2          TCP        52 [TCP Keep-Alive ACK] 443 → 483
203.312844434 217.160.0.48   10.8.0.2          TLSv1.2    98 Application Data
203.313271703 10.8.0.2       217.160.0.48      TCP        52 48388 → 443 [FIN, ACK] Seq=217
```

While Loading the game:

1. Initialize the TCP Connection -

```
2.302900162   192.168.31.178   202.141.80.20     TCP      74 47558 → 3128 [SYN] Seq=0 Win=29200 Len=0 MSS=14
2.306485485   202.141.80.20    192.168.31.178    TCP      74 3128 → 47558 [SYN, ACK] Seq=0 Ack=1 Win=14480 l
2.306572953   192.168.31.178   202.141.80.20     TCP      66 47558 → 3128 [ACK] Seq=1 Ack=1 Win=29312 Len=0
```

First Browser initiates a TCP connection with the server by performing 3-way Handshaking. 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.



Handshaking

Step 1 (SYN): Client wants to establish a connection with the server, so it sends a segment with SYN(Random value x) which informs the server that the client is likely to start communication.

Step 2 (SYN + ACK): Server responds to the client request with SYN-ACK signal bits set. ACK (value = x+1) signifies the response of segment it received and SYN signifies with what sequence number(Random value = y) it is likely to start the segments.

Step 3 (ACK): In the final part, the client acknowledges the response of server and they both establish a reliable connection with which they will start actual data transfer.

2. Initialize the TLS Connection -

```
2.550555034   192.168.31.178   202.141.80.20     TCP        66 47558 → 3128 [ACK] Seq=291 Ack=40 Win=29312 Len=0 TSval=1838818110 TSecr=3656387920
2.551083034   192.168.31.178   202.141.80.20     TLSv1.2   583 Client Hello
2.552240188   202.141.80.20    192.168.31.178    TCP        66 3128 → 47558 [ACK] Seq=40 Ack=808 Win=16640 Len=0 TSval=3656387923 TSecr=1838818111
2.798288425   202.141.80.20    192.168.31.178    TLSv1.2  1514 Server Hello
2.799796151   202.141.80.20    192.168.31.178    TLSv1.2  1514 Certificate [TCP segment of a reassembled PDU]
2.799831289   192.168.31.178   202.141.80.20     TCP        66 47558 → 3128 [ACK] Seq=808 Ack=2936 Win=35072 Len=0 TSval=1838818359 TSecr=365638811
2.799887592   202.141.80.20    192.168.31.178    TLSv1.2   269 Server Key Exchange, Server Hello Done
2.801154266   192.168.31.178   202.141.80.20     TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
2.809548821   192.168.31.178   202.141.80.20     TLSv1.2   421 Application Data, Application Data
2.810107531   202.141.80.20    192.168.31.178    TCP        66 3128 → 47558 [ACK] Seq=3139 Ack=934 Win=16640 Len=0 TSval=3656388181 TSecr=183881830
2.810355723   202.141.80.20    192.168.31.178    TCP        66 3128 → 47558 [ACK] Seq=3139 Ack=1289 Win=17792 Len=0 TSval=3656388181 TSecr=1838818
3.052738360   202.141.80.20    192.168.31.178    TLSv1.2   340 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
```

Handshaking

Step 1: The client sends a "Client hello" message to the server, along with the ACK client's random value and

Step 2: The server responds by sending a "Server hello" message to the client, along with the ACK server's random value.

Step 3: 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.

Step 4: If the server has requested a certificate from the client, the client sends it.

Step 5: 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.

Step 6: 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.

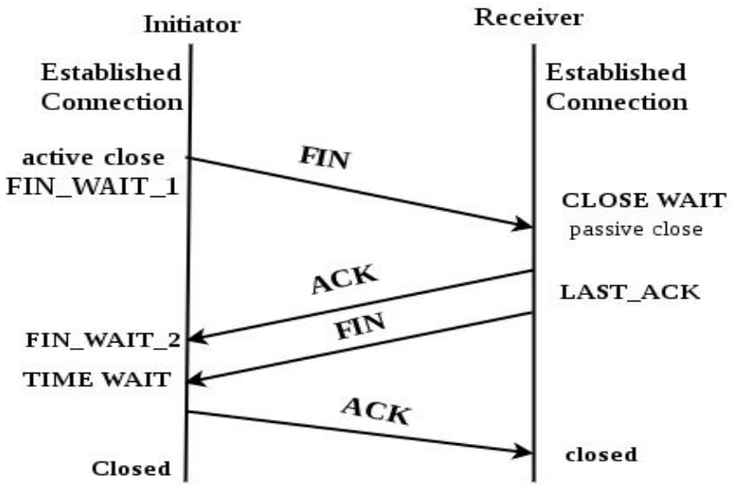
Step 7: The client sends "Change cipher spec" notification to the server to indicate that the client will start using the new session keys for hashing and encrypting messages. The client also sends "Client finished" message.

Step 8: Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption using the session keys. The server sends the "Server finished" message to the client.

Step 9: 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.

3. Terminating the Connection -

```
25.569867562  192.168.31.178   202.141.80.20    TCP   66 47558 → 3128 [FIN, ACK] Seq=9715 Ack=17703077 Win=1194112 Le
25.572865042  202.141.80.20    192.168.31.178   TCP   66 3128 → 47558 [FIN, ACK] Seq=17703077 Ack=9716 Win=23168 Len=
25.572886482  192.168.31.178   202.141.80.20    TCP   66 47558 → 3128 [ACK] Seq=9716 Ack=17703078 Win=1194112 Len=0 T
```



Handshaking

Step 1: Client application decides it wants to close the connection. The client sends a TCP segment with the FIN bit set to 1 to the server. Now, the client waits for a TCP segment from the server with an ACK.

Step 2: Server received FIN bit segment from Client, Server Immediately sends ACK segment along with FIN bit segment to the Client.

Step 3: Client receives a FIN bit segment from the Server, the client acknowledges the server's segment and enters the TIME_WAIT state. Client resends the final acknowledgment in case the ACK is lost. And then the server receives the ACK and the connection is shut-down.

4.

- **Ethernet(II)**

This protocol is the most common local area network technology and works at the Data Link layer. A frame header contains source and destination MAC addresses that indicate which device originated the frame and which device is expected to receive and process it. It also includes mechanisms to detect and even recover from transmission errors. System communicating over Ethernet divide a stream of data into frames as was visible through Wireshark.

- **Internet Protocol Version 4 (IPV4)**

The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking and essentially establishes the Internet. The Internet Protocol is responsible for addressing hosts, encapsulating data into datagrams (including fragmentation and reassembly) and routing datagrams from a source host to a destination host across one or more IP networks. For these purposes, the Internet Protocol defines the format of packets and provides an addressing system. Hence, in the current online gaming application to motivate the packet exchange between the source and host addresses, the IP protocol is used. This makes sure it is received and interpreted by the intended recipient. IP provides the fundamental mechanism using which data is delivered between devices which may or may not be in the same network which is the most fundamental concept in today's internet.

- **Transmission Control Protocol (TCP)**

In the current gaming application, TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. The game uses 3-way handshaking for establishment while 3 packets are used for termination. TCP provides host to host connectivity and control. If the packet is lost, it requests for retransmission. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and—because it is meant to provide error-free data transmission—handles retransmission of dropped or garbled packets as well as acknowledgment of all packets that arrive. It also uses the positive acknowledgment technique to guarantee the reliability of packet transfers. Generally, Online Multiplayer games prefer to use UDP as a faster response is required. But here very less data is being communicated to the server constantly as most of the game files are loaded on the client which is evident from the fact that while playing a

level no TCP packets are exchanged. Thus with the connection speeds offered today and also the nature of the game makes TCP also viable as it guarantees packet delivery.

- **Hypertext Transfer Protocol (HTTP)**

HTTP is an application layer protocol which uses a request-response protocol in the client-server computing model. HTTP is a protocol designed to transfer information (text, graphic images, sound, video, and other multimedia files) between computers over WWW (World Wide Web). HTTP pages are stored on your computer and internet caches. The pages load faster, but they are stored on systems that you potentially don't have control over e.g.: ISP's caching proxy. The current online gaming application is hosted by a server which is split into many domains to benefit the users in different ways. To handle the queries made by my machine and to interpret the queries sent by the server efficiently HTTP protocol is used. Our entire game data is also transferred through HTTP only initially as shown earlier in the game. Like in my game once a game javascript file is saved in Cookie then it cannot again request for the javascript file.

5.

Following are the statistics obtained at different times of the day.

Time	Throughput (Bytes/sec)	RTT (ms)	Avg. Packet Size (Bytes)	No. of Packets loss	No. UDP Packets	No. TCP Packets	Avg response wrt 1 request
11:00 AM	28K	194	903	3	136	25789	3.0687
05:00 PM	811K	0.292	2065	16	1901	46992	1.5699
11:00 PM	751	0.956	154	3	9	178	1.2733

6.

If we use IITG proxy server we cannot find IP where we access the game. If we use a VPN then we can find IP when we access the game. After many tries, I was only able to trace one IP because the game is a single player game, so it gets a javascript file and then runs internally.

In my **game**: **Super Mario** (https://supermarioemulator.com/) **IP**: 217.160.0.48 (Unique IP Address)

If the game is multiplayer then many users connect and server limits can be reached, your site cannot grow more without splitting it between servers or making it more efficient. Also, using two or more server rather than one super server is more efficient. If by chance the server fails then other servers can handle the situation but in case of one server, it goes offline. Also, uptime and reliability come into play.