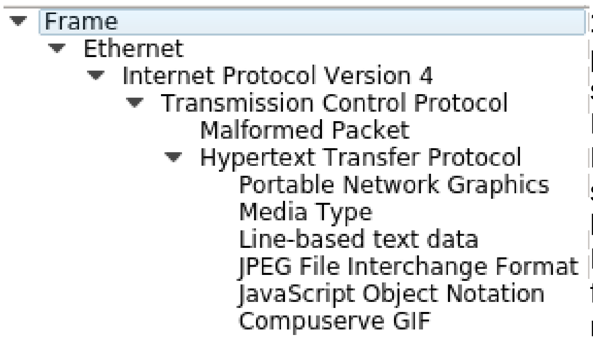
**Note –** As my project was on-line game and it was getting difficult to filter out the packets properly due to various applications using the net while I was playing the game so I created a software level network interface veth-b and routed all my packets though the interface and ran wireshark and firefox only on that. The commands used by me to create the interface are mentioned in a separate file included in the drive link which also contains all the traces.

**Note 2 –** By doing the above thing I was able to get only the packets used by firefox but still I was getting the packets by google adservices and other advertisements in my traces. To filter them out I used the filter **http.host = games.disney.ph** and then **followed its TCP stream** and got the trace correlated with the game only and used the data for all of my experiments. The trace submitted although consists of all the data including the one from the adservices **but any further screen shot/data in the assignment is data collected from the followed stream after using the mentioned filter.**

**Drive link -** https://drive.google.com/open?id=1r21_8AWc43bhl1tXRCzVNULckW2oBOVu

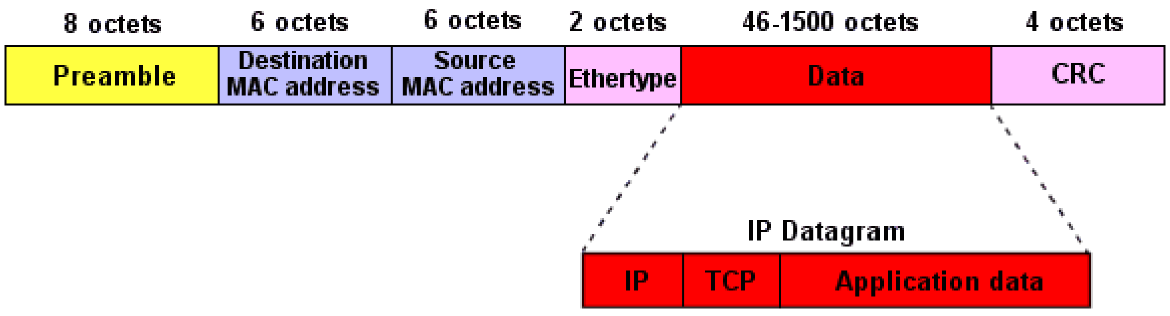
**Online Game Played -** http://games.disney.ph/danger-mouse-dash

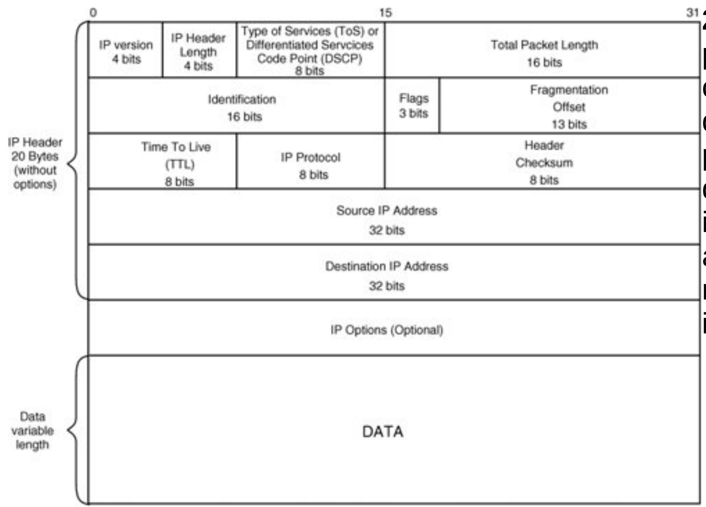
**Answer 1) Each field of the protocol explained properly in Answer 2) , here brief description given.**
The following protocols were being used by the online game as found from wireshark by checking the protocols on the TCP stream used by the game -



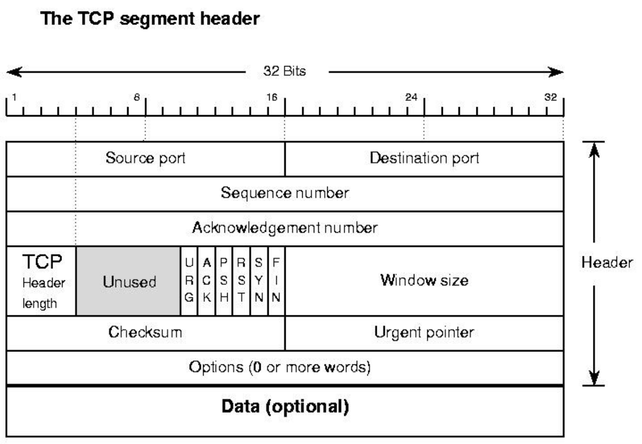
```
▼ Frame
    ▼ Ethernet
        ▼ Internet Protocol Version 4
            ▼ Transmission Control Protocol
                Malformed Packet
                ▼ Hypertext Transfer Protocol
                    Portable Network Graphics
                    Media Type
                    Line-based text data
                    JPEG File Interchange Format
                    JavaScript Object Notation
                    Compuserve GIF
```

**1) Ethernet** - An Ethernet frame is preceded by a preamble and start frame delimiter (SFD), to identify the start of the frame. Each Ethernet frame starts with an Ethernet header, which contains destination and source MAC addresses as its first two fields. The middle section of the frame is payload data including any headers for other protocols (for example, Internet Protocol) carried in the frame. The frame ends with a frame check sequence (FCS), which is a 32-bit cyclic redundancy check used to detect any in-transit corruption of data.

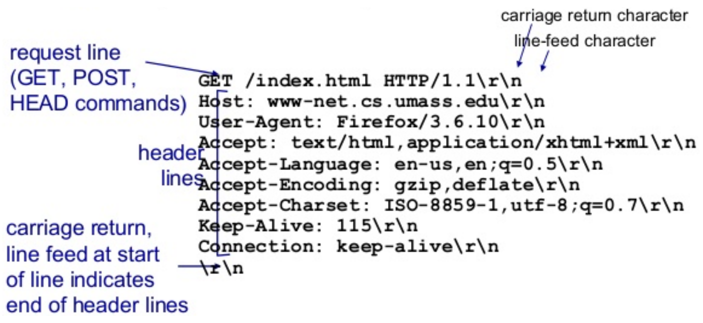



0
15
31
IP version 4 bits
IP Header Length 4 bits
Type of Services (ToS) or Differentiated Servcices Code Point (DSCP) 8 bits
Total Packet Length 16 bits
Identification 16 bits
Flags 3 bits
Fragmentation Offset 13 bits
Time To Live (TTL) 8 bits
IP Protocol 8 bits
Header Checksum 8 bits
Source IP Address 32 bits
Destination IP Address 32 bits
IP Options (Optional)
DATA
IP Header 20 Bytes (without options)
Data variable length

**2) Internet Protocol –** IPv4 is a connectionless protocol for use on packet-switched networks. It operates on a best effort delivery model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. This protocol has the responsibility of identifying hosts based upon their logical addresses and to route data among them over the underlying network. 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.



**3) Transmission Control Protocol -** The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application. TCP is optimized for accurate delivery rather than timely delivery and can incur relatively long delays (on the order of seconds) while waiting for out-of-order messages or re-transmissions of lost messages. 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 ack. Number and checksum.

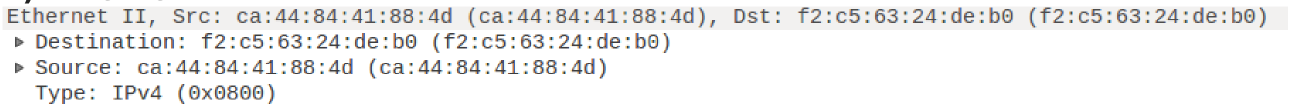


carriage return character
line-feed character
request line (GET, POST, HEAD commands)
header lines
carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

**4) HTTP -** A typical HTTP request contains many fields like Accept (to specify media types), accept-charset, accept-encoding and accept language. It also contains fields for proxy authorization, range, host and User-agent. An ‘expires’ field is also present which specifies the time after which the response is considered stale. HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client.

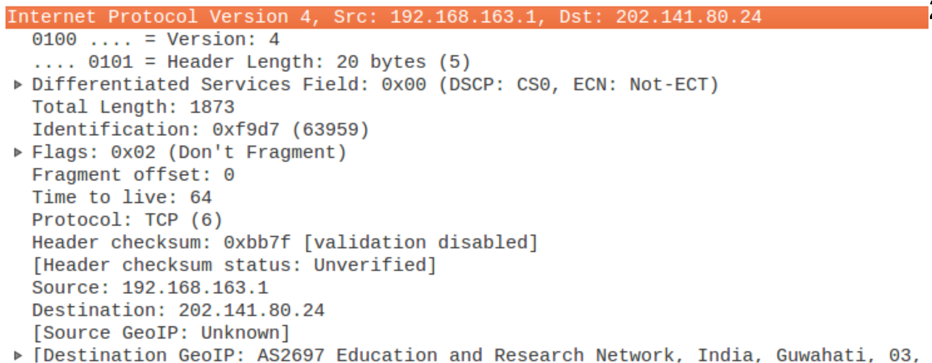
**Answer 2)**
**1) Ethernet II**



```
Ethernet II, Src: ca:44:84:41:88:4d (ca:44:84:41:88:4d), Dst: f2:c5:63:24:de:b0 (f2:c5:63:24:de:b0)
▸ Destination: f2:c5:63:24:de:b0 (f2:c5:63:24:de:b0)
▸ Source: ca:44:84:41:88:4d (ca:44:84:41:88:4d)
  Type: IPv4 (0x0800)
```

➔ **Destination:** gives the destination device’s adapter address.
➔ **Source:** gives the source’s device adapter address.
➔ **Type:** Includes a 16-bit type field to indicate what upper layer protocol should be used.

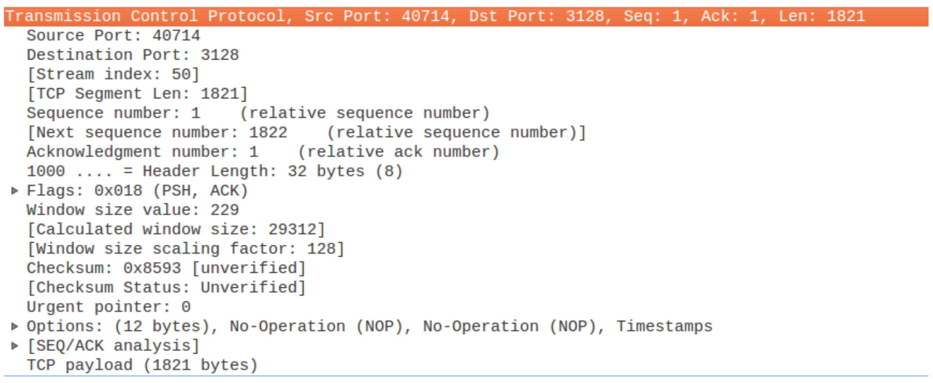
**2) IPV4**



```
Internet Protocol Version 4, Src: 192.168.163.1, Dst: 202.141.80.24
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1873
  Identification: 0xf9d7 (63959)
▸ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0xbb7f [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.163.1
  Destination: 202.141.80.24
  [Source GeoIP: Unknown]
▸ [Destination GeoIP: AS2697 Education and Research Network, India, Guwahati, 03,
```

➔ **Version** 4 bits : 4 (0100) .
➔ **Header length** denotes the length of entire IP header .
➔ **Differentiated Services Code Point (DSCP)** is a packet header value that can be used to indicate the level of service requested for traffic, such as high priority or best effort delivery. **ECN** is Explicit Congestion Notification .
➔ **Total length** is the length of IP packet = header + payload.
➔ **Identifications** number is different for each IP packet so that if an IP packet is fragmented during transmission it can be uniquely identified to the other fragments of the packet .
➔ **Flags** : Tells whether the packet is fragmented or not and if yes then is it the last fragment or not.
➔ **Fragment offset** tells the offset of the fragment in terms of 8 bytes.
➔ **Time to live (TTL)** is the maximum hops allowed to the packet after which it is dropped.
➔ **Protocol** refers to which protocol the packet belongs .
➔ **Header checksum** verifies if the header received is error free or not .
➔ **Source** address is the address of the source machine , in this case the ip that I have set up for veth b ie 192.168.163.1
➔ **Destination address** is the address of the proxy server.

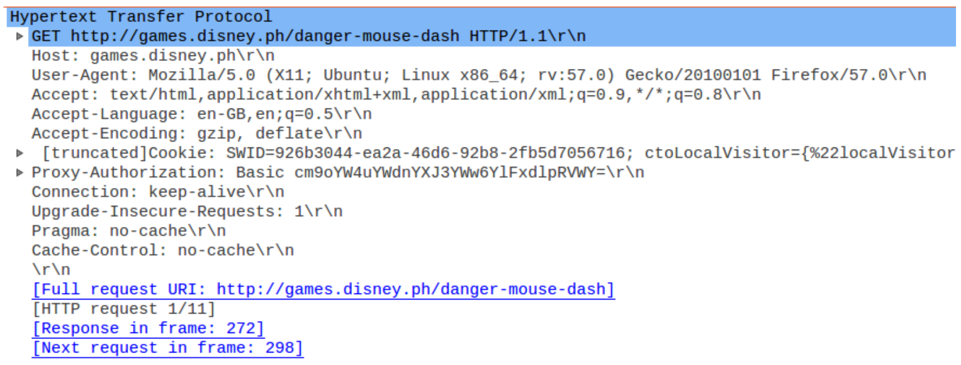
## 3) TCP



```
Transmission Control Protocol, Src Port: 40714, Dst Port: 3128, Seq: 1, Ack: 1, Len: 1821
    Source Port: 40714
    Destination Port: 3128
    [Stream index: 50]
    [TCP Segment Len: 1821]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1822    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0x8593 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
    TCP payload (1821 bytes)
```

➔ **Source port:** gives the port number of the host .
➔ **Destination port:** gives the port number of the receiver end .
➔ **Sequence number:** When data is sent from one end to another end it is divided into IP packets which have a definite sequence number relative to the other packets .
➔ **Acknowledgement number:** This shows the piggy banking of Acknowledgements.
➔ **Header length:** as is other protocols is the length of the TCP header.
➔ **Flags**: Contains one of the 9 flags which indicates the current scenario .
➔ **Window size:** Indicate how much buffer space is available on A for receiving packets.
➔ **CRC** is the checksum.
➔ **The Urgent Pointer:** is used when some information has to reach the server ASAP.
➔ **Options:** This field is optional and represents additional instructions not covered in the other TCP fields.

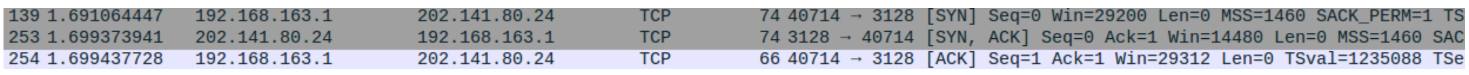
## 4) HTTP



```
Hypertext Transfer Protocol
  ▸ GET http://games.disney.ph/danger-mouse-dash HTTP/1.1\r\n
    Host: games.disney.ph\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-GB,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
  ▸  [truncated]Cookie: SWID=926b3044-ea2a-46d6-92b8-2fb5d7056716; ctoLocalVisitor={%22localVisitor
  ▸ Proxy-Authorization: Basic cm9oYW4uYWdnYXJ3YWw6YlFxdlpRVWY=\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
    \r\n
    [Full request URI: http://games.disney.ph/danger-mouse-dash]
    [HTTP request 1/11]
    [Response in frame: 272]
    [Next request in frame: 298]
```

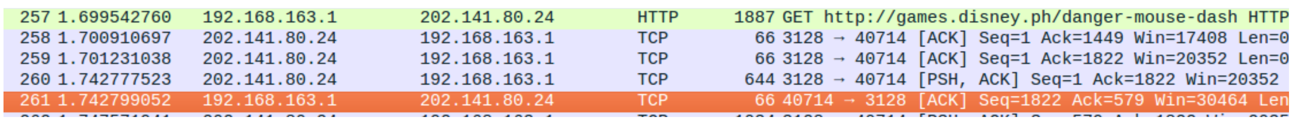
- ➔ **Request Method** currently is GET
- ➔ **Host –** The domain name of the server and TCP port no on which server is listening.
- ➔ **User-Agent –** Allows the application protocol to identify the application type,operating system etc.
- ➔ **Accept –** Media types that are acceptable for the response
- ➔ **Accept-Language –** List of acceptable human languages
- ➔ **Accept Encoding –** List of acceptable encodings
- ➔ **Cookie** - An HTTP cookie previously sent by the server with Set-Cookie.
- ➔ **Proxy_Authorization –** Authorization credentials for connecting to a proxy.
- ➔ **Connection –** Control options for the current connection and list of hop by hop request fields.
- ➔ **Cache-Control -** Used to specify directives that must be obeyed by all caching mechanisms along the request-response chain.
- ➔ **Upgrade-Insecure-Requests –** Tells the server that the client would prefer redirection to HTTPS.
- ➔ **Pragma -** Implementation-specific fields that may have various effects anywhere along the request-response chain.

**Answer 3)** The following sequence of message exchanges are observed in different cases:-
1) Initially loading of the game :-
1a) Initial establishment of the connection :- The browser initiates a TCP connection by performing a **3 way handshake** with the destination server which works as follows.



```
139 1.691064447    192.168.163.1          202.141.80.24          TCP          74 40714 → 3128 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TS
253 1.699373941    202.141.80.24          192.168.163.1          TCP          74 3128 → 40714 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SAC
254 1.699437728    192.168.163.1          202.141.80.24          TCP          66 40714 → 3128 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1235088 TSe
```

The client 192.168.163.1 sends a [SYN] frame for requesting the server to synchronize. Then the server (through our proxy server)  sends the [SYN,ACK] frame that is for acknowledging the request for synchronization that was sent by the client and at the same time sends request to the client for synchronizing its sequence number. Then the client then sends an [ACK] acknowledging the synchronization request from the server.
1b) Sending of Http data -



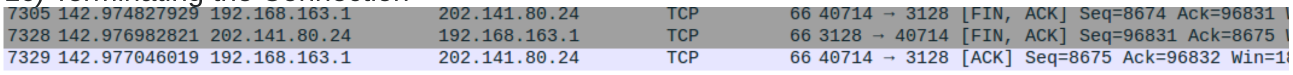
```
257 1.699542760    192.168.163.1          202.141.80.24          HTTP       1887 GET http://games.disney.ph/danger-mouse-dash HTTP
258 1.700910697    202.141.80.24          192.168.163.1          TCP          66 3128 → 40714 [ACK] Seq=1 Ack=1449 Win=17408 Len=0
259 1.701231038    202.141.80.24          192.168.163.1          TCP          66 3128 → 40714 [ACK] Seq=1 Ack=1822 Win=20352 Len=0
260 1.742777523    202.141.80.24          192.168.163.1          TCP         644 3128 → 40714 [PSH, ACK] Seq=1 Ack=1822 Win=20352
261 1.742799052    192.168.163.1          202.141.80.24          TCP          66 40714 → 3128 [ACK] Seq=1822 Ack=579 Win=30464 Len
```

The client using a GET request requests the HTML content After that data exchange happens using TCP protocol with the use of various ACKs and [PSH,ACK]s.
 TCP's push capability accomplishes two things:

- The sending application informs TCP that data should be sent immediately.
- The PSH flag in the TCP header informs the receiving host that the data should be pushed up to the receiving application immediately.

So the server tells the client at various intervals that it has no more data to send and requests an acknowledgement immediately to which the client also replies with an ACK.

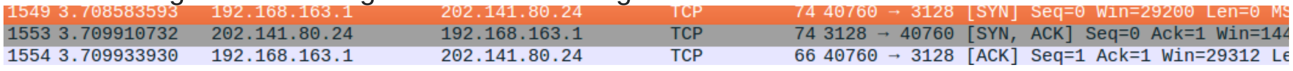
1c) Terminating the Connection -



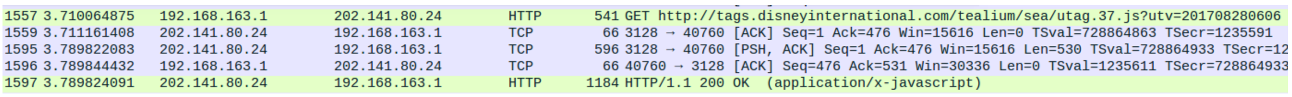
```
7305 142.974827929 192.168.163.1          202.141.80.24          TCP          66 40714 → 3128 [FIN, ACK] Seq=8674 Ack=96831
7328 142.976982821 202.141.80.24          192.168.163.1          TCP          66 3128 → 40714 [FIN, ACK] Seq=96831 Ack=8675
7329 142.977046019 192.168.163.1          202.141.80.24          TCP          66 40714 → 3128 [ACK] Seq=8675 Ack=96832 Win=1
```

1. The client sends a FIN along with an ACK. The FIN indicates to the server that it has no more data to send
whereas the ACK identifies the connection they have established.
2. Then the server acknowledges the FIN of client by sending an ACK.
3. Since the connections are independent, the server must now send a FIN to the client.
4. Finally the client acknowledges the FIN of server by an ACK to gracefully terminate the connection.

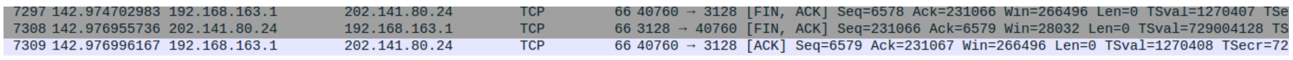
After the loading of the game no more live exchange of the data happens when the player is playing the game but whenever actions like **RESTART** , **PAUSE** , and **LEVEL CHANGE** actions are taken again the server is sent data and is informed about the actions that the user is taking. This is carried out by the same TCP connection setup , data transfer and teardown as shown above. For eg: when I changed the level of the game:-



```
1549 3.708583593   192.168.163.1     202.141.80.24     TCP        74 40760 → 3128 [SYN] Seq=0 Win=29200 Len=0 MS
1553 3.709910732   202.141.80.24     192.168.163.1     TCP        74 3128 → 40760 [SYN, ACK] Seq=0 Ack=1 Win=144
1554 3.709933930   192.168.163.1     202.141.80.24     TCP        66 40760 → 3128 [ACK] Seq=1 Ack=1 Win=29312 Le
```



```
1557 3.710064875   192.168.163.1    202.141.80.24    HTTP    541 GET http://tags.disneyinternational.com/tealium/sea/utag.37.js?utv=201708280606
1559 3.711161408   202.141.80.24    192.168.163.1    TCP      66 3128 → 40760 [ACK] Seq=1 Ack=476 Win=15616 Len=0 TSval=728864863 TSecr=1235591
1595 3.789822083   202.141.80.24    192.168.163.1    TCP     596 3128 → 40760 [PSH, ACK] Seq=1 Ack=476 Win=15616 Len=530 TSval=728864933 TSecr=12
1596 3.789844432   192.168.163.1    202.141.80.24    TCP      66 40760 → 3128 [ACK] Seq=476 Ack=531 Win=30336 Len=0 TSval=1235611 TSecr=728864933
1597 3.789824091   202.141.80.24    192.168.163.1    HTTP   1184 HTTP/1.1 200 OK  (application/x-javascript)
```



```
 7297 142.974702983 192.168.163.1     202.141.80.24     TCP       66 40760 → 3128 [FIN, ACK] Seq=6578 Ack=231066 Win=266496 Len=0 TSval=1270407 TSe
 7308 142.976955736 202.141.80.24     192.168.163.1     TCP       66 3128 → 40760 [FIN, ACK] Seq=231066 Ack=6579 Win=28032 Len=0 TSval=729004128 TS
 7309 142.976996167 192.168.163.1     202.141.80.24     TCP       66 40760 → 3128 [ACK] Seq=6579 Ack=231067 Win=266496 Len=0 TSval=1270408 TSecr=72
```

Which shows the transfer of javascript information indicated by the response.

**Answer 4)** Different protocols used by the application work at different layers of the OSI model. They perform a specific task which is necessary for the proper functioning of the application. They are discussed below:

**Hypertext Transfer Protocol (HTTP)** - HTTP is an application layer protocol which uses a request-response protocol in the client-server computing model. It is used to get various resources from the server like images,portable network graphics. It is visible from the traces that we use HTTP/1.1 which reuses a connection multiple times to transfer javascript objects which are our actions in the game and also other game related objects after the page has been delivered. Our entire game data is also transferred through HTTP only initially as shown earlier in the game.

**Transmission Control Protocol (TCP)** – It is evident from the traces that TCP layer handles all the handshaking which is done for the establishment and graceful termination of the connection. 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 also uses the positive acknowledgement technique to guarantee reliability of packet transfers. Generally games prefer to use UDP as a faster response is required in a live game. 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.

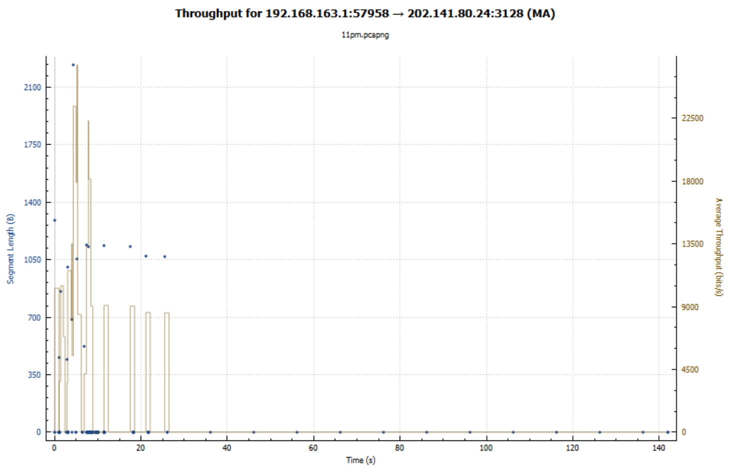
**Internet Protocol (IPv4)** – The Internet protocol or the IP handles tasks in the network layer which incorporates all the routing procedures. It has the job of delivering the packets from the source to the destination solely on the basis of the IP addresses in the packet headers. IP also provides security to networks by encapsulating the data into an IP datagram. 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.

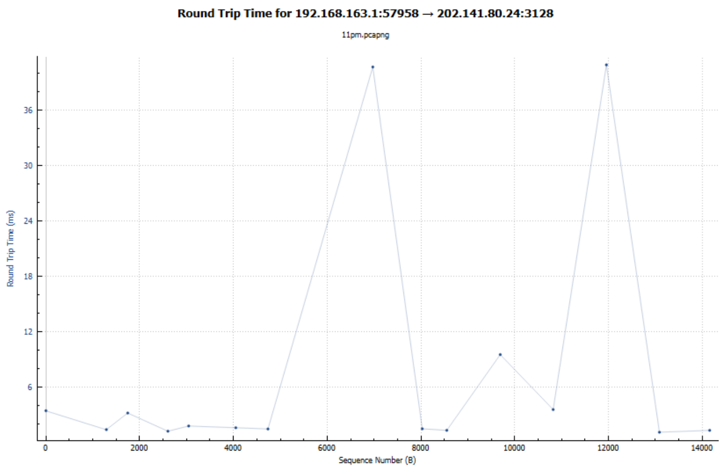
**Ethernet** – This protocol is the most common local area network technology and works at the Data Link layer. Details regarding the MAC address of the source and destination are found in Ethernet header. System communicating over Ethernet divide a stream of data into frames as was visible through Wireshark. The frame also contains error checking data so that damaged frames can be detected and discarded.

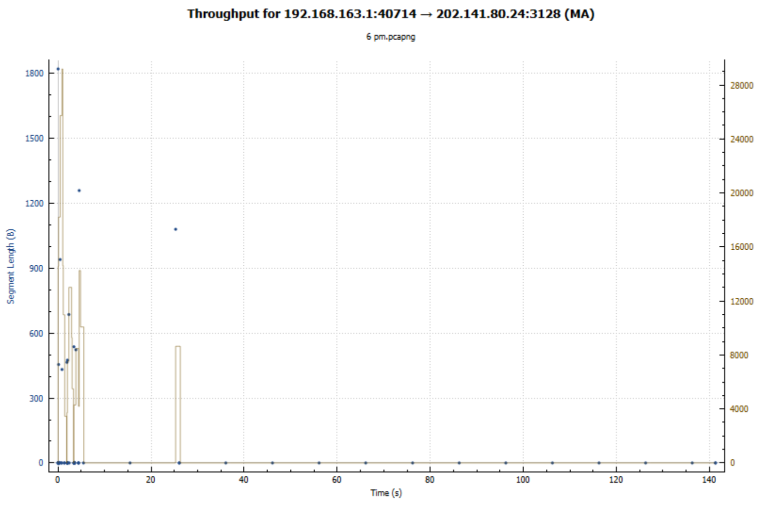
**Answer 5)**

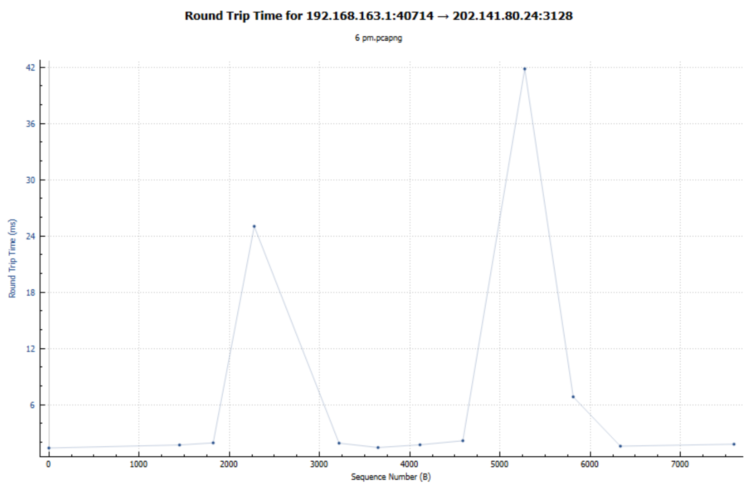
| Sample | Sample 1 | Sample 2 | Sample 3 |
|---|---|---|---|
| Time of Day | 6PM(lab) | 11PM(room) | 7:30AM(room) |
| Host A | 192.168.163.1 | 192.168.163.1 | 192.168.163.1 |
| Port A | 40714 | 57958 | 50736 |
| Host B | 202.141.80.24 | 202.141.80.24 | 202.141.80.24 |
| Port B | 3128 | 3128 | 3128 |
| Throughput A-B(bytes/s) | 791 | 1657 | 1928 |
| Throughput B-A(bytes/s) | 5783 | 28K | 45K |
| RTT(ms) | 8.3 | 19.39 | 13.9 |
| Average Packet Size(bytes) | 725 | 1219.95 | 1582.5 |

| | | | |
|---|---|---|---|
| No of packets lost | 0 | 5 | 4 |
| No of TCP packets | 160 | 441 | 503 |
| No of UDP packets | 0 | 0 | 0 |
| No of TCP packets from A to B | 80 | 207 | 240 |
| No of TCP packets from B to A | 80 | 234 | 263 |
| No of Response wrt requests sent | 1 | 1.13 | 1.096 |









**Answer 6)** Since we are using a proxy server, hence we cannot find the IP from where we are accessing the game.
It is not possible to find the IP using Wireshark but using online resources I found the IP of the host to be `2.21.246.148` using **http://ipinfo.info/html/ip_checker.php**. Now being curious I tried to repeat this experiment again and again, and got the same IPS again and again which basically means that always the request is processed by a unique server. Although in some heavy games multiple servers may be used for load balancing.