I have chosen AmongUs version 2020.9.9s for network protocol analysis using wireshark.
Traces used in the report are present on this drive link:
https://drive.google.com/drive/folders/1eQex3aQjP8hBjFegQgu-6EjqjyL8vbLO?usp=sharing

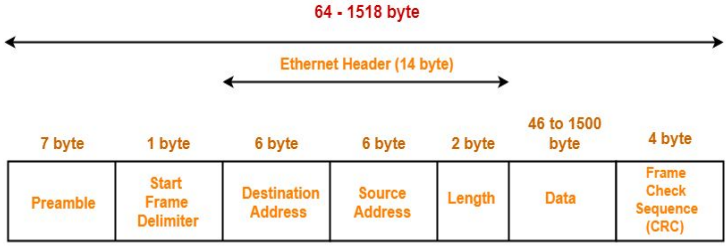
Question 1:

List out all the protocols used by the application at different layers (only those which you can figure out from traces). Study and briefly describe their packet formats. Mention and explain the observed values for at least 5 fields of the packets of each layer.

Solution:

1. The following protocols were used by the game at different layers. Since there were many applications running on the PC at the same time, many protocols correspond to the irrelevant apps and are therefore of no use to us.(Assumption: The values given for each protocol is after applying the filters of IP addresses and we have randomly chosen the packet corresponding to each protocol so the values may differ with other packets.)

A. Link Layer: Ethernet
B. Network Layer: IPv4(Internet Protocol Version 4), IPv6(Internet Protocol Version 6)
C. Transport Layer: TCP, UDP
D. Application Layer: TLS(Transport layer Security), Data Protocol, DNS Protocol.



IEEE 802.3 Ethernet Frame Format

Ethernet Protocol Packet format: (Link Layer)

The Ethernet frame begins with a Preamble and SFD both of which work at the physical layer. The **preamble** consists of a 56-bit (seven-byte) pattern of alternating 1 and 0 bits, allowing devices on the network to synchronize their receiver clocks, providing bit-level synchronization, **SFD** is the eight-bit (one-byte) value that marks the end of the preamble. The **Destination** and **Source MAC addresses** are MAC addresses of the receiving and the sending machine respectively, the **EtherType** field is the only one which diers between 802.3 and Ethernet II. **Data** is the data to be sent and CRC is a **CRC** polynomial code for error detection.

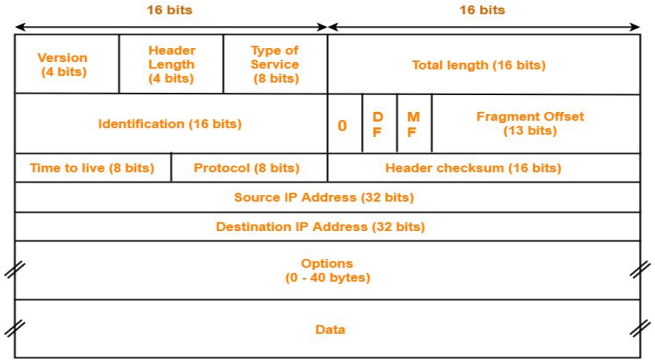
In the traces, every packet was following the ethernet protocol. This is the instance of a single packet(other packets may have swapped values of Src and Dest). The value of the **preamble** is fixed to 28 unset bits followed by 28 set bits. The value of **SFD** is set to 10101011.

```
Ethernet II, Src: IntelCor_7c:6a:e6 (00:bb:60:7c:6a:e6), Dst: Arcadyan_10:32:7e (64:cc:22:10:32:7e)
  > Destination: Arcadyan_10:32:7e (64:cc:22:10:32:7e)
  > Source: IntelCor_7c:6a:e6 (00:bb:60:7c:6a:e6)
    Type: IPv4 (0x0800)
```

IPv4 Packet format: (Network Layer)

Total length is the length of the IPv4 datagram in bytes. **Time to live** is the maximum number of remaining hops a packet is allowed to take on the network before its destination is reached.

```
Internet Protocol Version 4, Src: 40.67.188.15, Dst: 192.168.29.47
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x36f3 (14067)
> Flags: 0x4000, Don't fragment
  Fragment offset: 0
  Time to live: 105
  Protocol: TCP (6)
  Header checksum: 0x12ff [validation disabled]
  [Header checksum status: Unverified]
  Source: 40.67.188.15
  Destination: 192.168.29.47
```



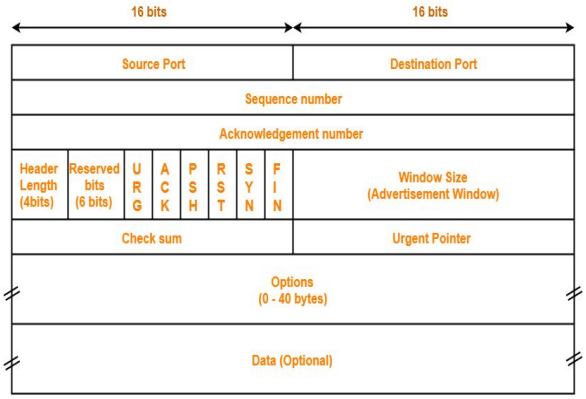
IPv4 Header

Header Checksum is a 16 bit value used to detect corruption in IPv4 header. **Source** is the IP Address of the source device. **Destination** is the IP Address of the destination device.

TCP Packet format: (Transport Layer)

Source Port is the port number associated with the process of sending a packet from source. **Destination Port** is the port number associated with the process of receiving a packet at the destination. **Sequence Number** is the byte number of the first byte of data in the TCP segment. **Window Size** is the value sent to tell the other device how much data to send before expecting an acknowledgement. It is the size of the receive buffer. **Acknowledgement number** is the sequence number of the next packet the sender of the packet expects to receive.

```
Transmission Control Protocol, Src Port: 443, Dst Port: 61646, Seq: 4452, Ack: 541, Len: 0
   Source Port: 443
   Destination Port: 61646
   [Stream index: 2]
   [TCP Segment Len: 0]
   Sequence number: 4452    (relative sequence number)
   Sequence number (raw): 1472914885
   [Next sequence number: 4452    (relative sequence number)]
   Acknowledgment number: 541    (relative ack number)
   Acknowledgment number (raw): 1855109836
   0101 .... = Header Length: 20 bytes (5)
 > Flags: 0x010 (ACK)
   Window size value: 261
```



TCP Header

UDP Packet format: (Transport Layer)

Source Port is the port number associated with the process of sending a packet from source. **Destination Port** is the port number associated with the process of receiving a packet at the destination. **Length** is the length of the UDP segment. **iv. Checksum** is the field used for error checking of header and data.

```
User Datagram Protocol, Src Port: 50043, Dst Port: 22023
    Source Port: 50043
    Destination Port: 22023
    Length: 11
    Checksum: 0xb485 [unverified]
```

UDP Header

TLSv1.2 Packet format: (Application Layer)

Content Type indicates which type of data is being contained in the packet. Version denotes the TLS version used. Length is the length of the data being sent. Encrypted Data is the data sent in encrypted form.

```
Transport Layer Security
 v TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 146
    Encrypted Application Data: 00000000000000022539a3d25060df1cf032
```

Byte	+0	+1	+2	+3
0	Content type			
1..4	Version		Length	
5..n	*Payload*			
n..m	MAC			
m..p	Padding (block ciphers only)			

DNS Packet format: (Application Layer)

The header describes the type of packet and which fields are contained in the packet. Following the header are a **number of questions, answers, authority records, and additional records. Question** contain information about the query that is being made. 1 indicates that the query type is a question. **Answer** contains the resource records for the name that was originally queried. **Authority** contains records of other authoritative servers. **Additional** contains other helpful records.

```
Domain Name System (response)
   Transaction ID: 0x7971
 > Flags: 0x8180 Standard query response
   Questions: 1
   Answer RRs: 3
   Authority RRs: 0
   Additional RRs: 0
```

```
+----------------------+
|        Header        |
+----------------------+
|       Question       |
+----------------------+
|        Answer        |
+----------------------+
|      Authority       |
+----------------------+
|      Additional      |
+----------------------+
```

Question 2:

Mention the important functionalities of the application as many as you can discover. (Two example functionalities for each application is given in Table 1). Explain which protocols are being used by which functionalities of the application. Give reason why those protocols are used for the functionalities.

Solution:

All the packets follow Ethernet (II) protocol at the data link layer. It is one of the most widely used and reliable link layer protocols. It has a well defined preamble for synchronization and CRC field for error detection. It ensures reliable data transfer between 2 network devices on the path of the packet. It gives collision free interconnection of multiple devices via a common bus.

All the packets except some DNS packets follow Internet Protocol Version 4 protocol at the network layer. This protocol is used because it lays the foundation of rules on which the internet is based upon. Also, the AmongUs server faces compatibility issues with IPv6. IPv4 uses a logical addressing system and performs routing, which is the forwarding of packets from a source host to the next router that is one hop closer to the intended destination host on another network.
Now, we will focus on the **Application Layer** and **Transport Layer** protocols.

TCP is a connection-oriented reliable data transfer protocol and involves handshaking between the client and the server. It also ensures proper error handling and flow control mechanisms to minimize the error loss rate which is needed in my application as while playing the game, we cannot afford in between data loss.
UDP is an alternative communications protocol to TCP used primarily for establishing low-latency and loss-tolerating connections between applications on the internet.
DNS is used to translate domain names into IP Addresses, which computers can understand. It responds with the IP address of the server. It allows users to have human-readable domains while ensuring a mapping to the IP addresses corresponding to the domain names.
TLSv1.2 protocol helps your system (clients and servers) communicate over the secured layer where data travels over the wire in encrypted format which could be understood only by the involved parties not by intrusions or outside audience.

The important functionalities that I discovered are-:

1. Starting(initially loading) of the game-:

Transport Layer Protocols-: TCP
Application Layer Protocols-: TLSv1.2, DNS
Reason-: We require **TCP** since we need to establish a reliable connection to log the user into the gamefield server. **TLSv1.2 protocol** is used here to ensure that the player's private data is encrypted and is safe from other users. This protocol basically provides security to the application. **DNS protocol** is basically used to return the IP address of the AmongUs server because this is the first time we are establishing connection to that server.

2. Moving the player in the map-:

Transport Layer Protocols-: UDP
Application Layer Protocols-: TLSv1.2
Reason-: Player movements in the game are done very frequently and every movement needs to be taken care of. Our main priority is speed. Hence, **UDP** packets are used. Even if some packets are corrupted or delivered out of order or simply missed, some states of movement would be missed but the final positions would be the same. Security is not as important because players move around very frequently and decoding the exact position using game coordinates is not needed. **TLSv1.2** is used to ensure that the activity of the player is isolated and does not get mixed with activity of other players.

3. Chat during emergency meeting-:

Transport Layer Protocols-: TCP, UDP
Application Layer Protocols-: TLSv1.2
Reason-: Players are conversing via text messages in the chat box. They are just sending short commands and plans. Accuracy is more important than speed. Hence, **TCP** is used for this game's chat. Also, text messages are sent using **TLSv1.2** application layer protocol. **UDP** packets are coming because the game is active in the background.

4. Completing a mission/task-:

Transport Layer Protocols-: TCP, UDP
Application Layer Protocols-: TLSv1.2
Reason-: TCP packets are used here since completing a mission involves complex tasks which need

reliable transfer of information to the game's server. The server needs to update the completed tasks information as well as check whether the tasks have been completed or not or whether an emergency meeting or sabotage has been called or not. **UDP packets** are used because the game is running for other players at the same time. **TLSv1.2** provides isolation to the player by securing its data.

5. Ending the game-:

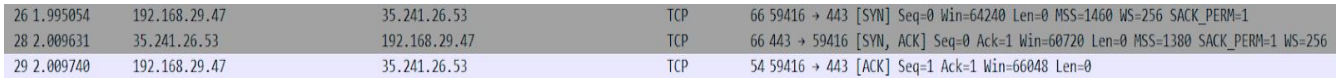
Transport Layer Protocols-: **TCP**
Application Layer Protocols-: **TLSv1.2**
Reason-: When we exit, the AmongUs server is notified of the exit with an encrypted message. This is a major game statistics and this information needs to be passed reliably, thus TCP is used. TLSv1.2 is used in the application layer so the opponents are notified that the particular player has left the game and they act accordingly.

Question 3:

For any two functionalities of the application (mentioned in question 2), show the sequence of messages (attach screenshot) exchanged to achieve those functionalities. Explain those message sequences. Check whether there are any handshaking sequences in the messages, and briefly explain the reason.

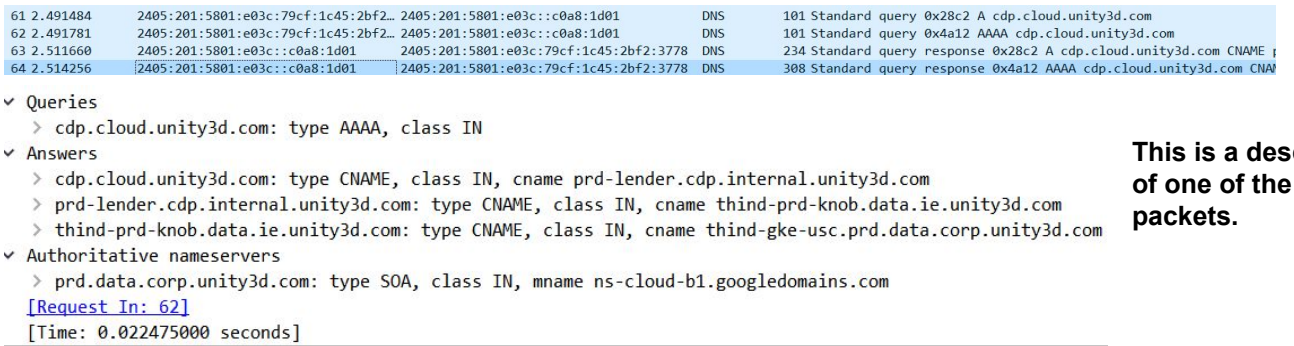
Solution:
Initial loading of the game.

A. 3- way TCP Handshaking to establish the connection:
This 3-way handshake process is designed so that both ends can initiate and negotiate separate TCP socket connections at the same time.
Step1(SYN from client to server): In the first step, the client wants to establish a connection with the server, so it sends a segment with SYN(Synchronize Sequence Number) which informs the server that the client is likely to start communication and with what sequence number it starts segments with.
Step2([SYN,ACK] from server to client): Server responds to the client request with SYN-ACK signal bits set. The ACK signifies the response of the segment it received and SYN signifies the sequence number it is likely to start the segments with.
Step3(ACK from client to server): The client acknowledges the response of the server and thus establishes a reliable connection with which they start the actual data transfer.
The steps 1, 2 establish the sequence number for one direction and it is acknowledged. With these, a full-duplex communication is thus established.



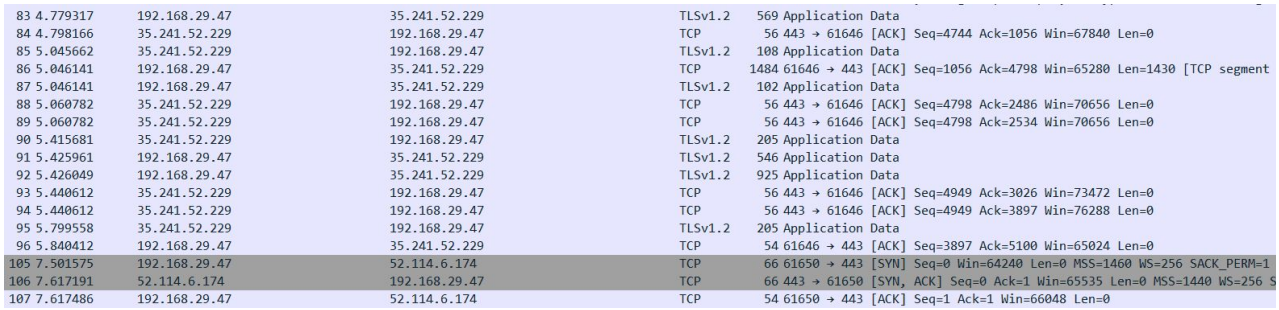
B.TLSv1.2 Handshaking(Application layer handshaking)starts with a client sending a "Client Hello" to the server to which the server responds with a "Server Hello" and "Server Certificate" to complete the handshake. The server also sends a server key which is used by the client for encryption purposes later in the process. Server is now waiting for the client's response which responds with a client key. This occurs during the initial loading and the application data can now be transferred between the client and the server.

C. DNS queries and answers: Whenever an application is opened, multiple **DNS queries are performed** to multiple DNS servers which give answers to resolve the address for game gateway and login. DNS packets contact the unity3d cloud in the case of AmongUs servers.

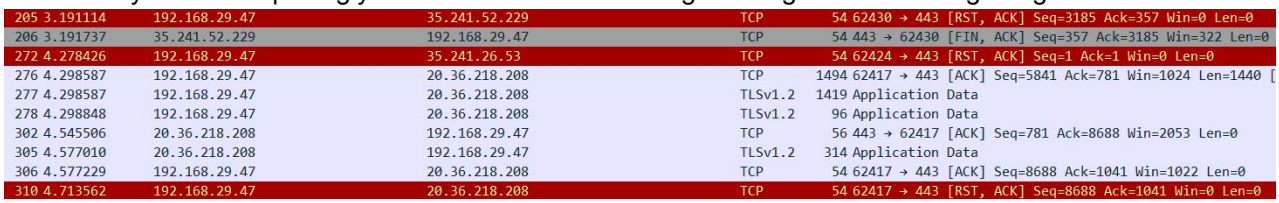


This is a description of one of the DNS packets.

Functionality I: Moving the player in the map (Playing the game)
While playing the game, we see that there is a transfer of TCP, TLSv1.2, UDP packets to and fro from my system. Application data as mentioned in the info section of the screenshot, is the main part of the packet along with TCP and Ip headers. While moving around in the game, we see the transfer of UDP packets because moving around does not require much accuracy but speed is our major priority. Different TCP segments arrive at the system(these may be in order or out-of-order). After the arrival of each segment of the packet, they are re-assembled to make them in order and then transferred to the application layer for further process(because out-of-order segments are not usable). There are no handshaking messages while playing the game.



Functionality II: Ending the game
The client sent a FIN,ACK packet to the game server(2nd line in the screenshot provided). The server then responded by sending a RST, ACK packet from its end which denoted the end of TCP session. Since the client did not send the final acknowledgment packet to the server, it denotes that the game ended abruptly and the connection was not successfully closed surprisingly. There are no handshaking messages while ending the game.



Question 4:

Calculate the following statistics from your traces while performing experiments at three different times (morning, afternoon, night) of the day: a) Throughput, b) RTT, c) Packet size, d) Number of packets lost, e) Number of UDP & TCP packets, f) Number of responses received with respect to one request sent.

	3 PM (Gameplay 1)	4 AM(Gameplay 2)	10 AM (Gameplay 3)
Throughput(Bytes/s)	873	1336	1263
RTT(ms)	58.605	22.046	77.792
Avg. Packet Size(bytes)	76	67	72
No. of packets lost	0	0	1
No. of TCP Packets	305	144	305
No. of UDP packets	5047	7808	9073
No. of responses per request sent	2.43	3.83	3.17

I applied these filters in the following gameplays:

Gameplay 1: ip.addr==40.67.188.15 || ip.addr==35.241.26.53 || ip.addr==35.241.52.229 || ip.addr==52.114.6.174 || ip.addr==52.114.36.16 || ip.addr==198.58.119.189
Gameplay 2: ip.addr==45.79.55.71 || ip.addr==52.139.168.125 || ip.addr==35.241.52.229 || ip.addr==35.241.26.53
Gameplay 3: ip.addr==35.241.26.53 || ip.addr==35.241.52.229 || ip.addr==172.105.168.113 || ip.addr==52.46.142.17 || ip.addr==40.67.188.15

Question 5:

Check whether the content is being sent/fetched by the application to/from the same or different destination(s)/source(s) during the three different times of the day used in question 4. If multiple destinations /sources exist, list out their IP addresses, and explain the reason behind this.

Solution:

I played the game three times to make contrast between the servers during different times using my Wi-Fi connection. Two server IP addresses were observed**: 35.241.26.53** and **35.241.52.229.** These two servers were constant throughout the day. Such websites use **multiple servers** as data transfer is faster because of **Load Balancing** of data across servers since there is little network congestion and **increased reliability** since there is no single point of failure. If a website uses multiple servers, it means that there are multiple ways for the user to get response to the requests, even if some server experiences some issues, others can provide the data to the client without interruption. On further analysis, while checking the TLS packets and DNS queries made during the time the game was loaded and played, I found that connections were made to many different servers while running the gaming website. These servers were not for the gaming application, but were called for certain specific additional tasks.

Some of them include:

- **fonts.googleapis.com :** While searching for the game, the device would have loaded the Google fonts to display the search results.
- **google-analytics.com:** Google Analytics is a web analytics service offered by Google that tracks and reports website traffic.