Siddharth Charan **Roll No. : 190101085**

CS344
Assignment #2

.

Question 1

I identified the following protocols used in different layers by performing the traces:

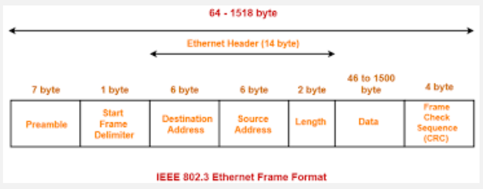
1. Data Link Layer:

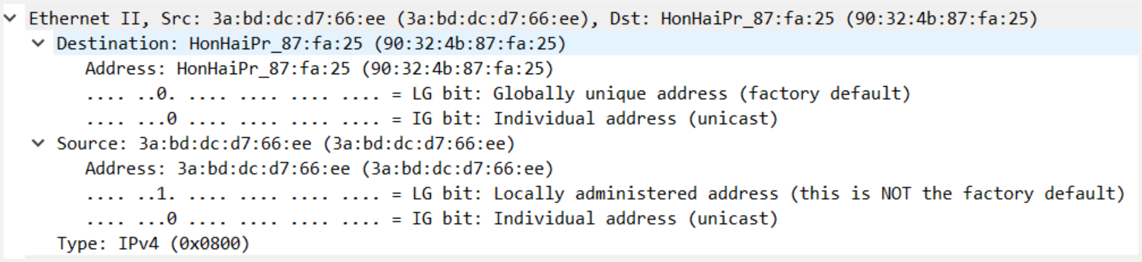
The data link layer is responsible for the node to node delivery of the message. The main function of this layer is to make sure data transfer is error-free from one node to another, over the physical layer.

EthernetII:

Ethernet is a widely used LAN technology. The frame of the ethernet packet starts with a preamble which enables the receiver to synchronise and know that a data frame is about to be sent. There is also the SFD (Start Frame Delimiter) which indicates the start of the frame. The preamble takes up 8 bytes. The Destination Address part gives the station MAC address where the packet is to intended to be sent. The first bit indicates whether it is an individual address or a group address. The source address consists of six bytes, and it is used to identify the sending station. Type field is the one which differentiates between the type of ethernet connection. User Data block contains the data to be sent and it may be up to 1500 bytes long. FCS contains Cyclic Redundancy Check (CRC) for error detection and analysis.

Figure 1: Ethernet Frame Format





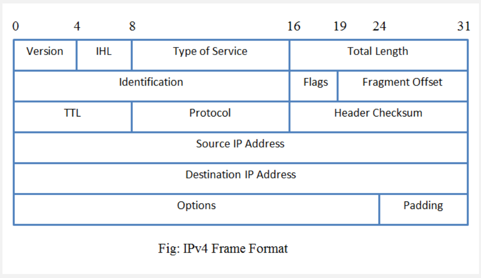
```
v Ethernet II, Src: 3a:bd:dc:d7:66:ee (3a:bd:dc:d7:66:ee), Dst: HonHaiPr_87:fa:25 (90:32:4b:87:fa:25)
  v Destination: HonHaiPr_87:fa:25 (90:32:4b:87:fa:25)
      Address: HonHaiPr_87:fa:25 (90:32:4b:87:fa:25)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: 3a:bd:dc:d7:66:ee (3a:bd:dc:d7:66:ee)
      Address: 3a:bd:dc:d7:66:ee (3a:bd:dc:d7:66:ee)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

2. Network Layer

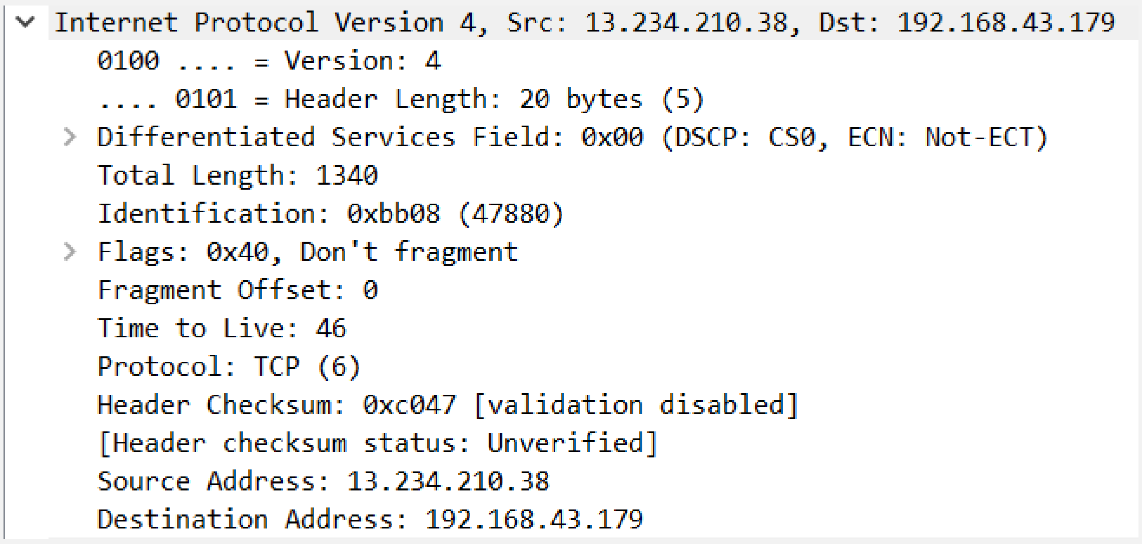
Network layer works for the transmission of data from one host to the other located in different networks. The following protocols were observed in the network layer:

Internet Protocol Version 4:

Figure 2: IPV4 Frame Format



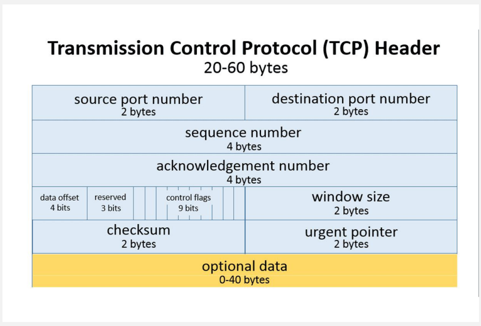
IP is responsible for transferring data packets from the source to the host. The packet header contains many fields. Version indicates the version of the IP used, in this case, it is version-4. The header length specifies the length of IP header. Total length field determines the entire packet size in bytes, including header and data. The identification field is primarily used for uniquely identifying the group of fragments of a single IP datagram. Fragment offset specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram.



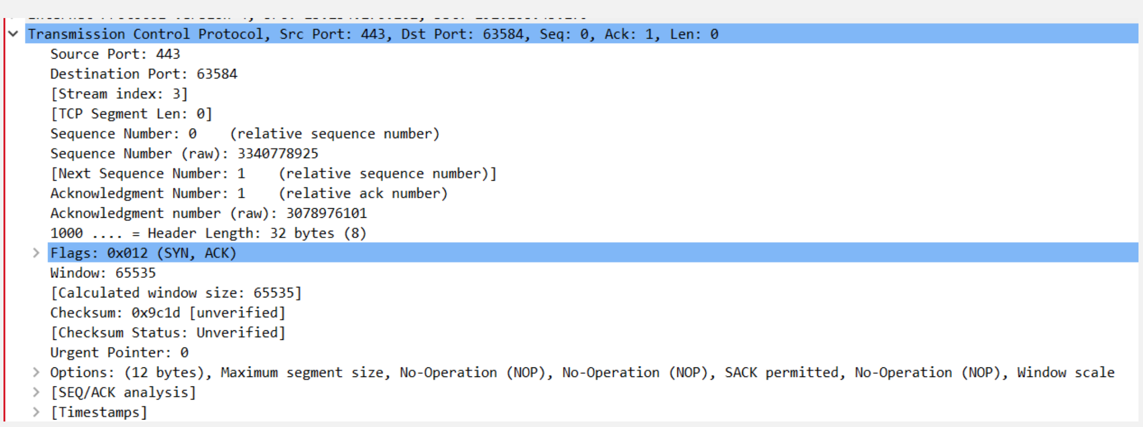
```
Internet Protocol Version 4, Src: 13.234.210.38, Dst: 192.168.43.179
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1340
    Identification: 0xbb08 (47880)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 46
    Protocol: TCP (6)
    Header Checksum: 0xc047 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 13.234.210.38
    Destination Address: 192.168.43.179
```

3. Transport layer:
This layer is responsible for establishment of connection, maintenance of sessions, authentication and also ensures security.
Transmission Control Protocol(TCP):

Figure 3: TCP header format

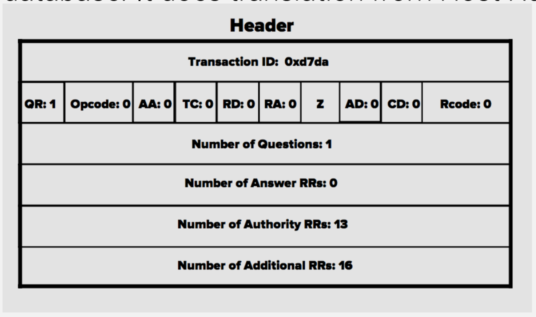


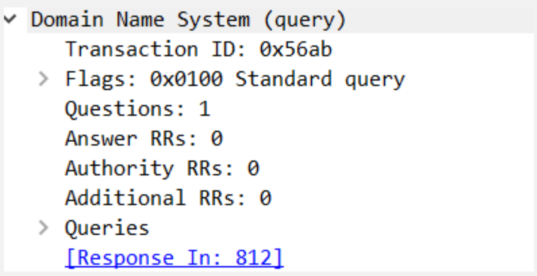
TCP is used for organizing data in a way that ensures the secure transmission between the server and client. Source Port and Destination Port indicates the port of the sending and receiving application. Sequence Number contains the sequence number of the first data byte. Acknowledgement Number field (32 bits) contains the sequence number of the data byte that receiver expects to receive next from the sender. Header Length specifies the length of the TCP header. There is a total of 6 types of Flags of 1 bit each. Some of them are ACK, PSH and SYN. Checksum is used to verify the integrity of data in the TCP payload. Window Size contains the size of the receiving window of the sender. It advertises how much data (in bytes) the sender can receive without acknowledgement. Urgent Pointer indicates how much data in the current segment counting from the first data byte is urgent. Options are used for different purposes like timestamp, window size extension, parameter negotiation, padding.



```
⌄ Transmission Control Protocol, Src Port: 443, Dst Port: 63584, Seq: 0, Ack: 1, Len: 0
    Source Port: 443
    Destination Port: 63584
    [Stream index: 3]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3340778925
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3078976101
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x9c1d [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale
  > [SEQ/ACK analysis]
  > [Timestamps]
```

4. Application layer:
Application layer is at the top of the stack. These applications produce the data, which has to be transferred over the network
Domain Name System (DNS) : This is a distributed database implemented in a hierarchy of DNS servers and an application-layer protocol that allows hosts to query the distributed database. It does translation from Host Name to IP





```
⌄ Domain Name System (query)
    Transaction ID: 0x56ab
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  > Queries
    [Response In: 812]
```

• Identification : Field is a 16-bit number that identifies the query. Flags in the flag field include query/reply flag, and authoritative flag.
• Destination : It indicates the address of the destination adapter.
• Authority : This section contains records of other authoritative servers
• QR,Query/Response : QR = 0 means a query, QR = 1 means a response.
• Rcode : Consists of 4 bits and the code returned to a query or response.
• Total Questions : Number of entries in the question list that were returned.

Question 2

I was assigned Github desktop for my assignment. This desktop application provides great interface to manage your files using github.com. You can systematically manage your documents using this application. Some of the important functions of GitHub are as follows:-
1. Creating a Repository
2. Push to a Repository(Adding new files to repository)
3. Pull a Repository
4. Cloning a Repository
5. Deleting a Repository

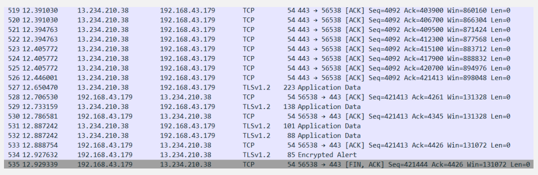
Now, we will see some important protocols used by different functionalities:
1. DNS protocol: This is an important protocol as it resolves the IP address for the github.com DNS uses UDP packets because these are fast and have low overhead and hence does not need any connection between the sever and the client. It is used by the every functionality of GitHub.
2. TCP Protocol: TCP 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 crossplatform communications among heterogeneous networks.It uses ow control, Error control and congestion control mechanisms. It is also used by all functionalities of GitHub.
3. TLS Protocol: It is also used by all the functionalities of this application.It 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.
4. UDP Protocol: The User Datagram Protocol provides faster data transfer in comparison to TCP but it lacks components such as security and reliability. Thus GitHub uses UDP only for performing DNS queries.
5. OCSP Protocol: It is used by clone and pull commands to check whether the server's certificate is revoked or not.
6. Ethernet II: It is used for error handling. It is used by all the functionalities of the application, since it has a reliability, rate of data transfer coupled with flow control.

Question 3

Part 1
We were asked to show the sequence of messages for any two functionalities and we were asked to explain them. I have worked on Git clone and Git push. Details are as given below:

A. Git Push (Adding files to a repository):
The client sends application data to the server when we create a repository in GitHub and the sever responds by sending ACK packets. No Handshaking is observed here.As shown in the above image, the packets (PDU,protocol data unit) need to be reassembled because they might arrive out of order (by using different routes, to ensure load balancing).

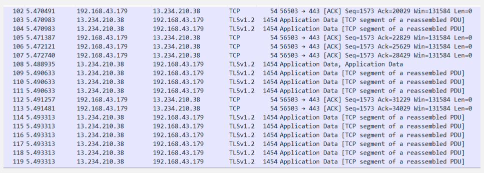
Figure 4: Ethernet Frame Format



```
519 12.391030   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=403900 Win=860160 Len=0
520 12.391030   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=406700 Win=866304 Len=0
521 12.394763   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=409500 Win=871424 Len=0
522 12.394763   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=412300 Win=877568 Len=0
523 12.405772   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=415100 Win=883712 Len=0
524 12.405772   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=417900 Win=888832 Len=0
525 12.405772   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=420700 Win=894976 Len=0
526 12.446001   13.234.210.38    192.168.43.179   TCP        54 443 → 56538 [ACK] Seq=4092 Ack=421413 Win=898048 Len=0
527 12.650470   13.234.210.38    192.168.43.179   TLSv1.2   223 Application Data
528 12.706530   192.168.43.179   13.234.210.38    TCP        54 56538 → 443 [ACK] Seq=421413 Ack=4261 Win=131328 Len=0
529 12.733159   13.234.210.38    192.168.43.179   TLSv1.2   138 Application Data
530 12.786581   192.168.43.179   13.234.210.38    TCP        54 56538 → 443 [ACK] Seq=421413 Ack=4345 Win=131328 Len=0
531 12.887242   13.234.210.38    192.168.43.179   TLSv1.2   101 Application Data
532 12.887242   13.234.210.38    192.168.43.179   TLSv1.2    88 Application Data
533 12.888754   192.168.43.179   13.234.210.38    TCP        54 56538 → 443 [ACK] Seq=421413 Ack=4426 Win=131072 Len=0
534 12.927632   192.168.43.179   13.234.210.38    TLSv1.2    85 Encrypted Alert
535 12.929339   192.168.43.179   13.234.210.38    TCP        54 56538 → 443 [FIN, ACK] Seq=421444 Ack=4426 Win=131072 Len=0
```

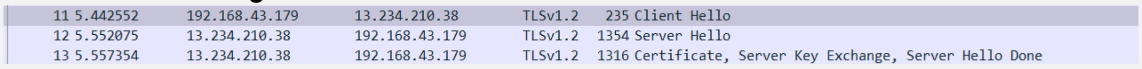
B. Git clone(copying a repository to our system):
You can see many TCP packages. But, no handshaking is observed. While clonning, the client sends the data via TCP packets and TLS ensures that the exchanged data is encrypted.The packets may arrive out of order hence the data needs to be reassembled. That is the reason you can majorly see ACK, FIN ACK and application data in messages.

Figure 5: Ethernet Frame Format


```
102 5.470491  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=20029 Win=131584 Len=0
103 5.470983  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
104 5.470983  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
105 5.471387  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=22829 Win=131584 Len=0
106 5.472121  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=25629 Win=131584 Len=0
107 5.472740  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=28429 Win=131584 Len=0
108 5.488935  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data, Application Data
109 5.490633  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
110 5.490633  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
111 5.490633  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
112 5.491257  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=31229 Win=131584 Len=0
113 5.491481  192.168.43.179  13.234.210.38   TCP       54 56503 → 443 [ACK] Seq=1573 Ack=34029 Win=131584 Len=0
114 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
115 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
116 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
117 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
118 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
119 5.493313  13.234.210.38   192.168.43.179  TLSv1.2 1454 Application Data [TCP segment of a reassembled PDU]
```

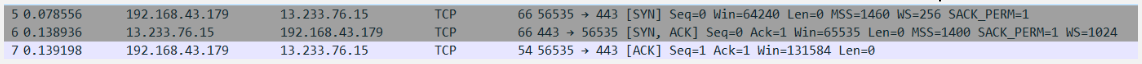
Part 2
We were also asked to check for handshaking sequences. I checked the messages which I got and I found following handshaking sequences were observed in the messages:-

1. TLS Handshaking


```
11 5.442552  192.168.43.179  13.234.210.38   TLSv1.2   235 Client Hello
12 5.552075  13.234.210.38   192.168.43.179  TLSv1.2  1354 Server Hello
13 5.557354  13.234.210.38   192.168.43.179  TLSv1.2  1316 Certificate, Server Key Exchange, Server Hello Done
```

Client sends a Client Hello and server responds with Server Hello and authentication key.

2. TCP connection establishment
To connect to the github, 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 ag set requesting the server to synchronize with provided sequence number. Then server sends packet with SYN and ACK ag 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 ags set to the server having sequence number set to received ACK value and ACK number set to one more than the received sequence number.


```
5 0.078556  192.168.43.179  13.233.76.15    TCP  66 56535 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
6 0.138936  13.233.76.15    192.168.43.179  TCP  66 443 → 56535 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PERM=1 WS=1024
7 0.139198  192.168.43.179  13.233.76.15    TCP  54 56535 → 443 [ACK] Seq=1 Ack=1 Win=131584 Len=0
```

Question 4

S.N.	Property	7AM	2pm	11pm
1	Throughput			
2	RTT			
3	packet Size			
4	Number of packets lost			
5	Number of UDP and TCP packets			
6	Responses per request sent			

Question 5

I performed my task 4, thrice during the day(in morning, evening and night). I indeed found different IP addresses of source/destination servers.
Reason: It is happening because, GitHub is a website having huge traffic at almost all times of the day. Hence, they use multiple servers to fasten up the data transfer which happens due to Load Balancing of data across servers since there is little network congestion increased reliability. Different servers are also helpful in ensuring reliability since there is no single point of failure. Even if some server experiences some issues, others can provide data to the client without any sort of interruptions. Hence, when we try to connect to the site at different time of day, we can experience difference in IP addresses.
Time in Q4 and Q5 is different as, I rechecked Q5 next day.

Time	IP
8 AM	13.233.123.47
2 PM	13.233.167.60
8 PM	13.233.167.15