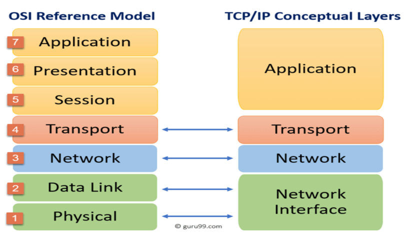
CS342: Networks Lab
Assignment 2

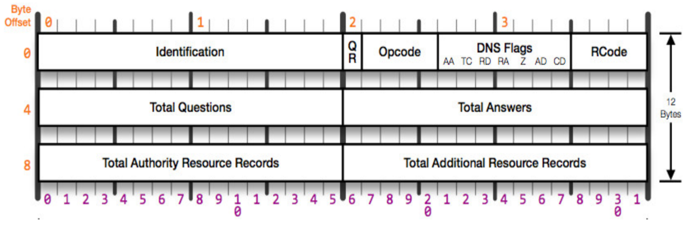
Tejas Khairnar
180101081
Application Assigned: Github Desktop
Traces: https://bit.ly/3kT3S3J

Question 1: Protocols used by the application at different layers



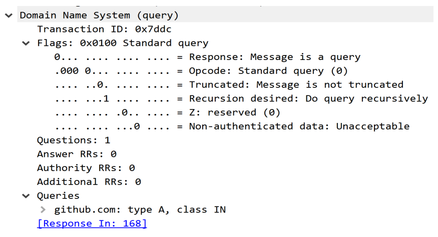
To classify the protocols used by the application **OSI model** is going to be used. **OSI model** has **7 layers** whereas **TCP/IP** has **4 layers**.
The **OSI model** is a logical and conceptual model that defines network communication used by systems open to interconnection and communication with other systems.
On the other hand, **TCP/IP** helps to determine how a specific computer should be connected to the internet.

1. Application layer



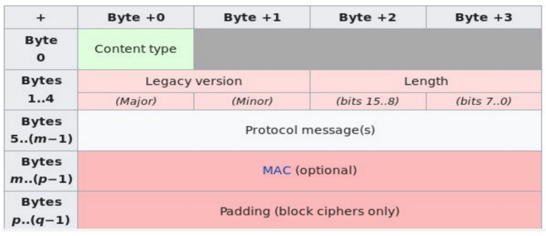
DNS: The **Domain Name System** is a host name to IP address translation service. DNS is a distributed database implemented in a hierarchy of name servers. It is used for message exchange between clients and servers.The DNS packet header consists of the following fields:
1) Identification(16bits): Used to match requests/reply packets.
2) QR,Query/Response(1bit): **QR=0** means a query, **QR=1** means a response.
3) Opcode(4bits): Used to specify the type of query or response.
4) Flags(7bits): AA:Authoritative Answer, TC:Truncated, RD:Recursion Denied, RA:Recursion Available, AD:Authenticated Data, CD:Checking Disabled. (Each having a bit to state on or off)
5) Rcode(4bits): The code returned to a query or response.
6) Total Questions(16bits): Number of entries in the question list that were returned.
7) Total Answers(16bits): Number of entries in the answer resource record list that were returned.
8) Total Authority: Number of entries in the authority resource record list that were returned.
9) Total Additional: Number of entries in the additional resource record list that were returned.

Identification(Transaction ID)	0x77dc
Opcode	0000: A standard query
Rcode(response)	0 : A query as a response.
Total questions	1 : 1 question was returned
Total Answers	0 : No answer was returned



```
⌄ Domain Name System (query)
    Transaction ID: 0x7ddc
  ⌄ Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⌄ Queries
    > github.com: type A, class IN
    [Response In: 168]
```

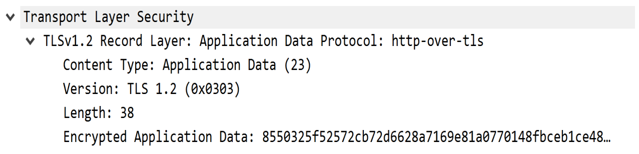
2. Session layer



+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	Legacy version		Length	
	(Major)	(Minor)	(bits 15..8)	(bits 7..0)
Bytes 5..(m−1)	Protocol message(s)			
Bytes m..(p−1)	MAC (optional)			
Bytes p..(q−1)	Padding (block ciphers only)			

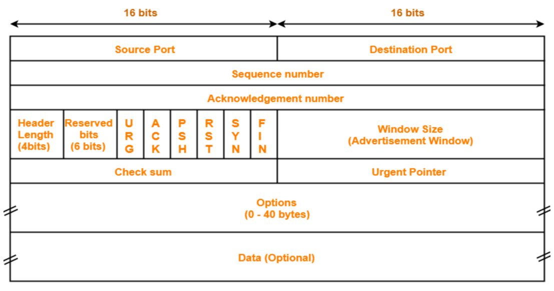
TLSv1.2(SSL): The **Transport layer security** sits between the **Application layer and the Transport layer** and provides security in transmission by encryption of data to be sent.The basic unit of SSL(Secure Socket Layer) is a **Record**.Each record has a **5 byte record header** followed by a data.The Header has **Content Type** which is one from : *Handshake, Application Data, Alert, Change Cipher Spec and Heartbeat.* Therefore protocol message changes according to the content type. **Legacy version** identifies the major and minor version of TLS prior to TLS 1.3. **Length** tells the length of application data, excluding protocol header & including MAC & padding trailers.

Content Type	Application Data (23 is the identifier)
Legacy Version	TLS 1.2 (prior to TLS 1.3)
Length	38: Length of application data
Encrypted data	Encrypted form of sent data



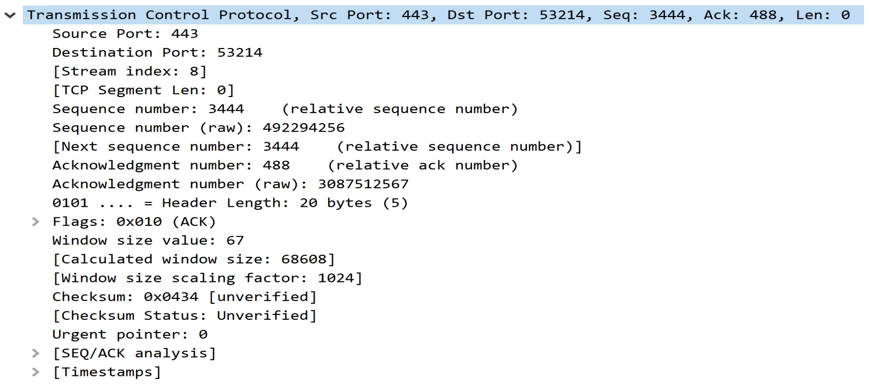
```
v Transport Layer Security
  v TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 38
      Encrypted Application Data: 8550325f52572cb72d6628a7169e81a0770148fbceb1ce48…
```

3. Transport layer



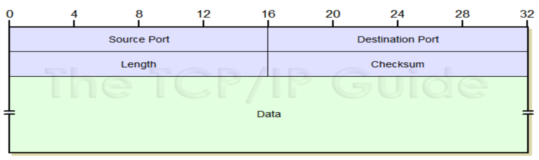
TCP Header

TCP: The **Transmission Control protocol(TCP)** packet format consists of these fields: **Source Port and Destination Port** (16 bits each) identify the end points of the connection. **Sequence Number** (32 bits) specifies the number assigned to the first byte of data in the current message. **Acknowledgement Number** (32 bits) contains the value of the next sequence number that the sender of the segment is expecting to receive, if the ACK control bit is set. **Header Length** (4 bits): Specifies the size of the TCP header in 32-bit words. **Reserved data** (3 bits) in TCP headers always has a value of zero. This field serves the purpose of aligning the total header size as a multiple of four bytes. **Flags field** (6 bits) contains the various **flags (eg., URG, ACK, PSH, RST, SYN, FIN)**. **Window Size** (16 bits): The size of the receive window, which specifies the number of window size units (by default, bytes) (beyond the segment identified by the sequence number in the acknowledgment field) that the sender of this segment is currently willing to receive. **Checksum** (16 bits) is used to check if the header was damaged in transit. **Urgent pointer field** (16 bits) if the URG flag is set, then this field is an offset from the sequence number indicating the last urgent data byte. **Options** have up to three fields: **Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable)**. The TCP header **padding** is used to ensure that the TCP header ends, and data begins, on a 32 bit boundary. **Data field** (variable length) contains upper-layer information.



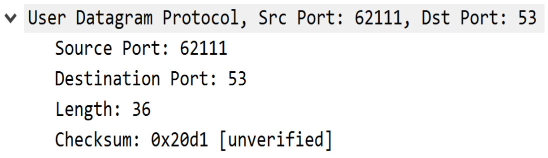
```
v Transmission Control Protocol, Src Port: 443, Dst Port: 53214, Seq: 3444, Ack: 488, Len: 0
    Source Port: 443
    Destination Port: 53214
    [Stream index: 8]
    [TCP Segment Len: 0]
    Sequence number: 3444    (relative sequence number)
    Sequence number (raw): 492294256
    [Next sequence number: 3444    (relative sequence number)]
    Acknowledgment number: 488    (relative ack number)
    Acknowledgment number (raw): 3087512567
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window size value: 67
    [Calculated window size: 68608]
    [Window size scaling factor: 1024]
    Checksum: 0x0434 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
```

Source Port	443	Destination Port	53214
Urgent Pointer	0 (i.e. Not set)	Flag	ACK is set to 488
Seq number	3444	Window Size	67 bytes
Acknowledgement Number	488	Header length	5 (length of packet 20 bytes)
Checksum	0x0434(for error correction)		



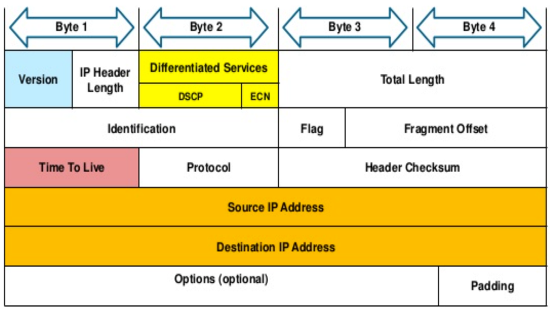
UDP: The **User Datagram Protocol** uses a simple connectionless communication model with a minimum of protocol mechanisms.**Destination and Source Ports** are both 16 bits long used to indentify the port of the destined packet and the source.**Length** is a 16 bit field which identifies the length of UDP including the header and the data.To check whether the end to end data is not corrupted **Checksum** is used.

Source Port	62111 is the source port
Destination Port	53 is the destination port
Length	36 i.e. including the header length
Checksum	0x20d1 (unverifed)

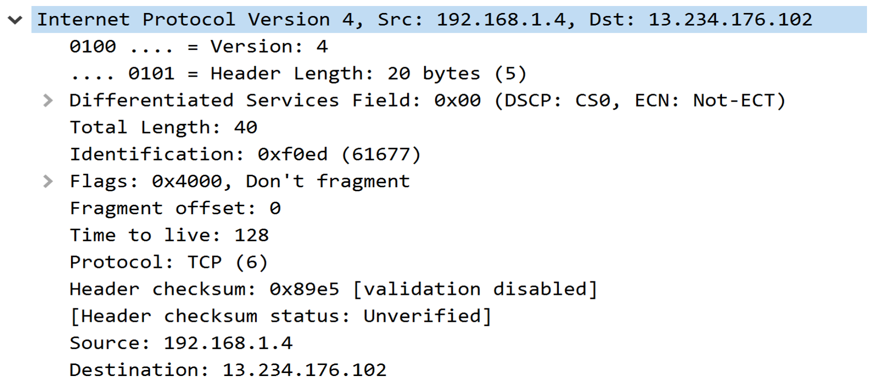


```
v User Datagram Protocol, Src Port: 62111, Dst Port: 53
    Source Port: 62111
    Destination Port: 53
    Length: 36
    Checksum: 0x20d1 [unverified]
```

4. Network layer



IPv4: An **Internet Protocol version 4** packet header (IPv4 packet header) contains application information, including usage and source/destination addresses. IPv4 packet headers contain 20 bytes of data and are normally 32 bits long. Detailed Header field description is: **Version** (4 bits): This contains the Internet header format and uses only four packet header bits. **Internet header length (IHL)** (4 bits): This field specifies the size of of the header in the multiples of 32 bits The minimum is 5 which means length of 5 x 32 bits=160 bits. **Differentiated Services Code Point** (6 bits): this is Type of Service. **Explicit Congestion Notification** (2 bit): It carries information about the congestion seen in the route. **Total Length** (16 bits): defines entire packet size. **Identification** (16 bits): For uniquely identifying the group of fragments of a single IP datagram. **Flags** (3 bits): used to control or identify fragments. **Fragment offset** (13 bits): This offset tells the exact position of the fragment in the original IP Packet. **Time to Live** (TTL) (8 bits): This contains the total number of routers allowing packet pass-through.**Protocol** (8 bits): Tells the Network layer at the destination host, to which Protocol this packet belongs. **Header checksum** (16 bits): It checks and monitors communication errors. **Source and destination address** (32 bits each): It stores source and destination IP address respectively. **Options** : This is the last packet header field and is used for additional information. When it is used, the header length is greater than 32 bits.

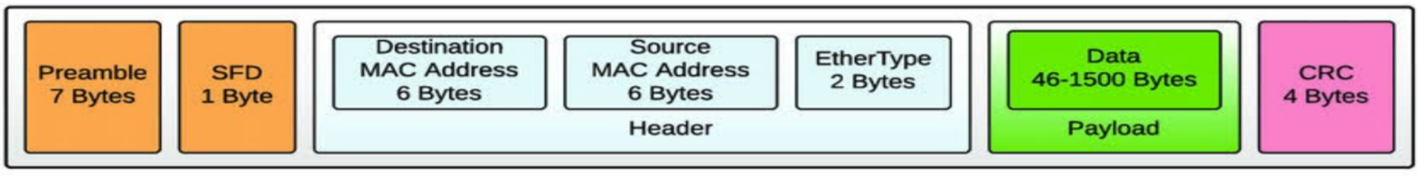


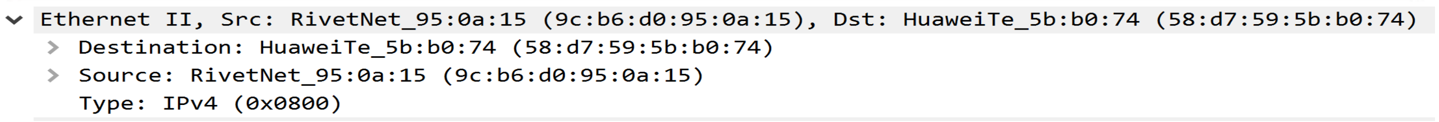
```
⌄ Internet Protocol Version 4, Src: 192.168.1.4, Dst: 13.234.176.102
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 40
     Identification: 0xf0ed (61677)
   > Flags: 0x4000, Don't fragment
     Fragment offset: 0
     Time to live: 128
     Protocol: TCP (6)
     Header checksum: 0x89e5 [validation disabled]
     [Header checksum status: Unverified]
     Source: 192.168.1.4
     Destination: 13.234.176.102
```

Version	4(IPv4)	Header Length	5 (Hence 5*4=20bytes)
Total Length	40 bytes	Checksum	0x89e5(to check errors)
Protocol	TCP(protocol used in above layer)	Time to live	128 (max hops allowed for packet)
Fragment offset	0	Flags	Dont fragment
Source	192.168.1.4 (My IP)	Destination	13.234.176.102

5. Data-link layer

Ethernet II: The Ethernet frame begins with a Preamble and SFD i.e. start frame delimiter, both of which work at the physical layer. The **preamble** consists of a **7 byte** i.e a 56 bits pattern of alternating 1 and 0 bits, allowing devices on the network to synchronize their receiver clocks, providing bit-level synchronization and **SFD** is the **8 bit** 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 differs between 802.3 and Ethernet II. **Data** is the data to be sent and **Cyclic Redundancy Check** i.e CRC, is a **CRC-32 polynomial code** for error detection.





```
⌄ Ethernet II, Src: RivetNet_95:0a:15 (9c:b6:d0:95:0a:15), Dst: HuaweiTe_5b:b0:74 (58:d7:59:5b:b0:74)
  > Destination: HuaweiTe_5b:b0:74 (58:d7:59:5b:b0:74)
  > Source: RivetNet_95:0a:15 (9c:b6:d0:95:0a:15)
    Type: IPv4 (0x0800)
```

Destination MAC	58:d7:59:5b:b0:74 (MAC address of the wifi router in my home)
Source MAC	9c:b6:d0:95:0a:15 (MAC address of the source)
Type	IPv4 (specifies upper layer protocol used in IPv4)

Question 2

There are some very important functionalities of github namely:

1. Creating a Repository
2. Push to a Respository
3. Pull a Repository
4. Cloning a Repository
5. Creating a Branch of the repository

The **TLS protocol** is used by every functionality of github.TLS is used for prevention of unwanted eavesdropping and modification on internet traffic.TLS protocol is used as it encrypts data to and from the site to clients.This also protects the integrity of the website by helping to prevent intruders tampering between the site and client browsing.User's login and credentials details are protected as they are encrypted when they are send hence safe-guarding privacy and security.

The **DNS protocol** is also used by the every functionality of github as it is used to resolve 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.

The **TCP protocol** is used by all functionalities of the github at the transport layer.TCP always 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 cross-platform communications among heterogeneous networks.It uses flow control, Error control and congestion control mechanisms.

The **IPv4 protocol** is also used by all functionalities of github at the network layer.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.

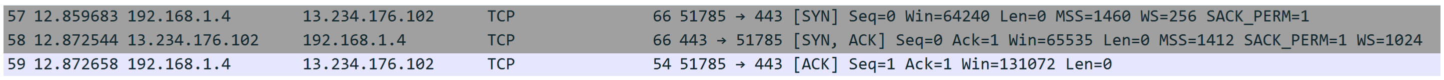
The **Ethernet II** is also used by all functionalities of github in the data link layer.This contains information about source and destination MAC address found in its header.Ethernet lying in data link layer is also responsible for error detection and correction along with flow control. It exists as a point to point connection.

To check whether the server's certificate is revoked or not while using clone or pull in github **The Online Certificate Status Protocol (OCSP)** is used.

The **User Datagram Protocol** is a connection-less protocol and provides faster data transfer than TCP but is not very reliable or secure. It is mostly used to transfer small individual packets (like DNS requests).Github uses DNS queries to fetch IP addresses, which are made using UDP.

Question 3

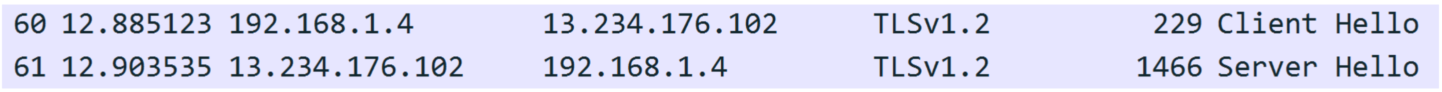
1. TCP connection establishment i.e 3 way handshake



```
57 12.859683 192.168.1.4       13.234.176.102     TCP          66 51785 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
58 12.872544 13.234.176.102    192.168.1.4        TCP          66 443 → 51785 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 SACK_PERM=1 WS=1024
59 12.872658 192.168.1.4       13.234.176.102     TCP          54 51785 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
```

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 flag set requesting the server to synchronize with provided sequence number. Then server sends packet with **SYN and ACK flag** 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** flags set to the server having sequence number set to received ACK value and ACK number set to one more than the received sequence number.

2. TLS handshaking

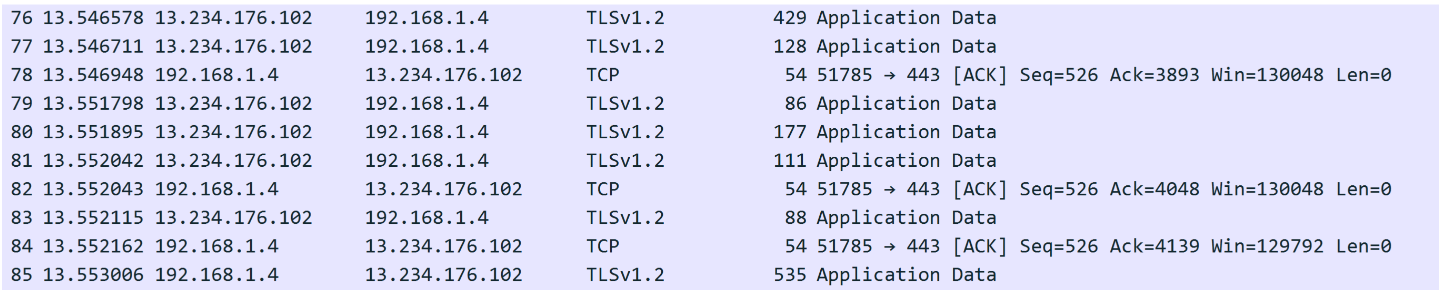


```
60 12.885123 192.168.1.4       13.234.176.102     TLSv1.2        229 Client Hello
61 12.903535 13.234.176.102    192.168.1.4        TLSv1.2       1466 Server Hello
```

TLS handshake enables the TLS **client and server to establish the secret keys** with which they communicate. They are many ways to do the same.
1. The client sends a **"Client Hello"** message that lists cryptographic information such as the TLS version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations.
2. The server responds with a **"Server Hello"** message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string.
3. The server may sends its Certificate message and ServerKeyExchange message.

NOTE : The above handshakes occur when we open github desktop application.

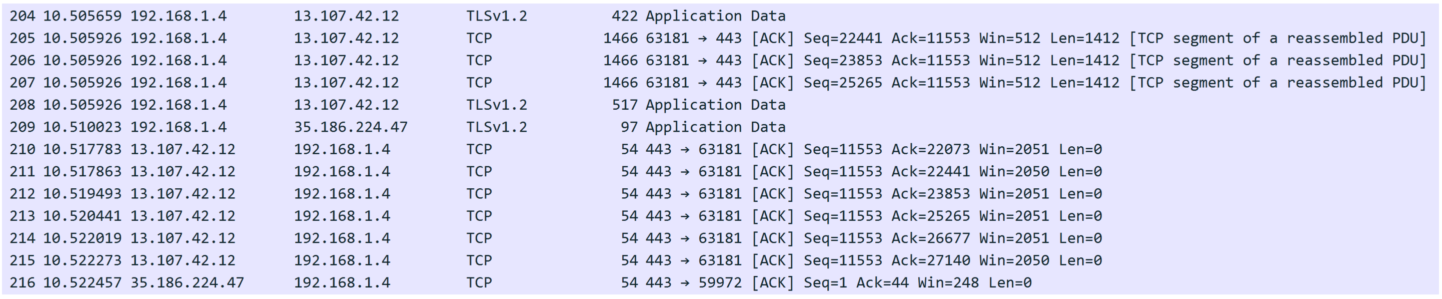
3. Cloning a repository



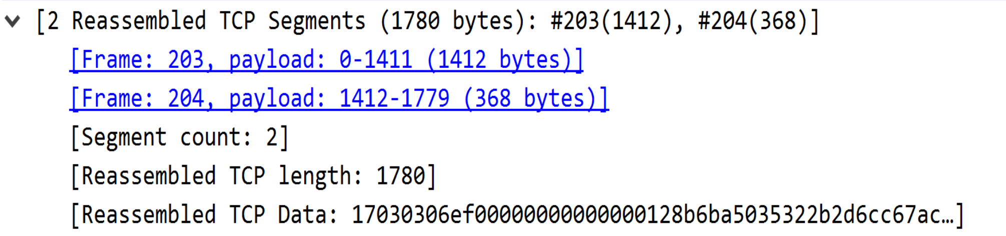
```
76 13.546578 13.234.176.102    192.168.1.4       TLSv1.2        429 Application Data
77 13.546711 13.234.176.102    192.168.1.4       TLSv1.2        128 Application Data
78 13.546948 192.168.1.4       13.234.176.102    TCP             54 51785 → 443 [ACK] Seq=526 Ack=3893 Win=130048 Len=0
79 13.551798 13.234.176.102    192.168.1.4       TLSv1.2         86 Application Data
80 13.551895 13.234.176.102    192.168.1.4       TLSv1.2        177 Application Data
81 13.552042 13.234.176.102    192.168.1.4       TLSv1.2        111 Application Data
82 13.552043 192.168.1.4       13.234.176.102    TCP             54 51785 → 443 [ACK] Seq=526 Ack=4048 Win=130048 Len=0
83 13.552115 13.234.176.102    192.168.1.4       TLSv1.2         88 Application Data
84 13.552162 192.168.1.4       13.234.176.102    TCP             54 51785 → 443 [ACK] Seq=526 Ack=4139 Win=129792 Len=0
85 13.553006 192.168.1.4       13.234.176.102    TLSv1.2        535 Application Data
```

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. **No Handshaking** is observed here.

4. Creating a repository



```
204 10.505659 192.168.1.4       13.107.42.12      TLSv1.2        422 Application Data
205 10.505926 192.168.1.4       13.107.42.12      TCP           1466 63181 → 443 [ACK] Seq=22441 Ack=11553 Win=512 Len=1412 [TCP segment of a reassembled PDU]
206 10.505926 192.168.1.4       13.107.42.12      TCP           1466 63181 → 443 [ACK] Seq=23853 Ack=11553 Win=512 Len=1412 [TCP segment of a reassembled PDU]
207 10.505926 192.168.1.4       13.107.42.12      TCP           1466 63181 → 443 [ACK] Seq=25265 Ack=11553 Win=512 Len=1412 [TCP segment of a reassembled PDU]
208 10.505926 192.168.1.4       13.107.42.12      TLSv1.2        517 Application Data
209 10.510023 192.168.1.4       35.186.224.47     TLSv1.2         97 Application Data
210 10.517783 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=22073 Win=2051 Len=0
211 10.517863 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=22441 Win=2050 Len=0
212 10.519493 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=23853 Win=2051 Len=0
213 10.520441 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=25265 Win=2051 Len=0
214 10.522019 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=26677 Win=2051 Len=0
215 10.522273 13.107.42.12      192.168.1.4       TCP             54 443 → 63181 [ACK] Seq=11553 Ack=27140 Win=2050 Len=0
216 10.522457 35.186.224.47     192.168.1.4       TCP             54 443 → 59972 [ACK] Seq=1 Ack=44 Win=248 Len=0
```



```
[2 Reassembled TCP Segments (1780 bytes): #203(1412), #204(368)]
    [Frame: 203, payload: 0-1411 (1412 bytes)]
    [Frame: 204, payload: 1412-1779 (368 bytes)]
    [Segment count: 2]
    [Reassembled TCP length: 1780]
    [Reassembled TCP Data: 17030306ef0000000000000128b6ba5035322b2d6cc67ac…]
```

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).

Why Handshaking occurs?

Handshaking occurs just before a connection is established between the server and the client ie before launching the github application hence we dont see handshakes in any functionality of github if the connection between the server and client is already established.However we will find handshakes in these functionalities if a different server is used to process our requests.

Question 4

Property	Morning	Afternoon	Night
Throughput (kilobytes/sec)	249	53	29
RTT (ms)	48.2	20.3	22.5
Average packet size (bytes)	945	1784	456
No. of packets lost	0	0	0
No. of TCP packets	7522	2047	656
No. of UDP packets	6	60	47
Responses per Request sent	3985/3537=1.12	1136/925=1.22	352/306=1.15

Question 5

Different IP addresses are used by github at different times of day:
(NOTE : IP found using the filter "frame contains github")
Morning : 13.233.76.15
Afternoon : 13.234.210.38
Night : 13.234.176.38
As it has multiple servers around the world.When a request is sent to any of the servers depending on the network traffic and congestion in a manner that distributes the requests uniformly,This is known as **load balancing** and hence helps in keeping the network traffic stable.The multiple servers are being used to increase the **reliability** of the system i.e. even if some server fails other servers will provide alternative routes between two points and keep the website functioning smoothly.