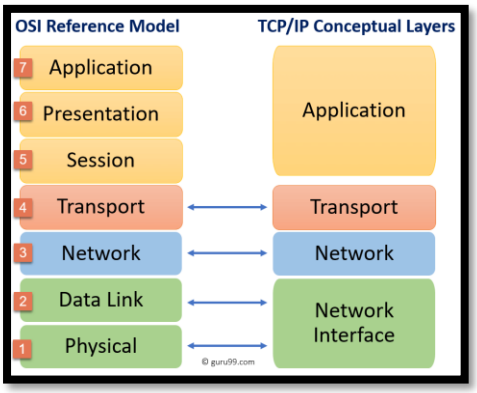
ASSIGNMENT -2

CS 342: Networks Lab (September – November2020)

Name: Vaibhav Kumar Singh

Roll No.: 180101086

Q1



The various protocols used by GitHub Desktop application are explained below in the respective layers that they belong to.

1. Application layer

(i) **TLS** (Transport Layer Security): It is a cryptographic protocol designed to provide communications security over a computer network.

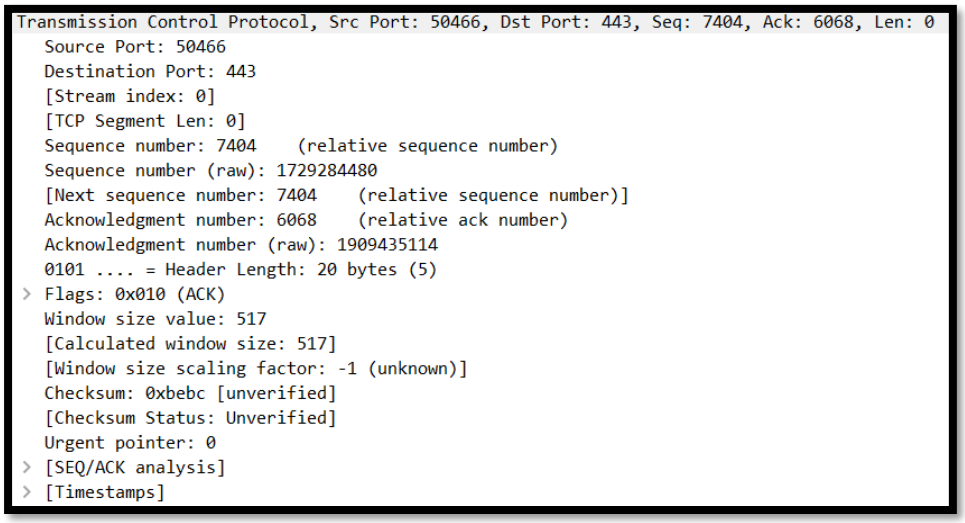
```
Transport Layer Security
▾ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 6062
    Encrypted Application Data: 0000000000000006ba4d33b319c49b31b15f560e11b7e8fe…
```

Application Data Protocol	http-over-tls	The protocol ensures secure communication over the network by encrypting the data and sending it through http requests.
Content Type	Application Data (23)	Indicates the type of content being transferred. In this case, 23 is the identifier which tells that Application Data will be transferred
Version	TLS 1.2 (0x0303)	Indicates the version of the protocol that is used
Length	6062	The length of the application data being transferred. It includes the MAC and padding trailers and excludes the protocol headers
Encrypted Application Data	0000…	The encrypted form of the data transmitted over the network.

2. Transport layer

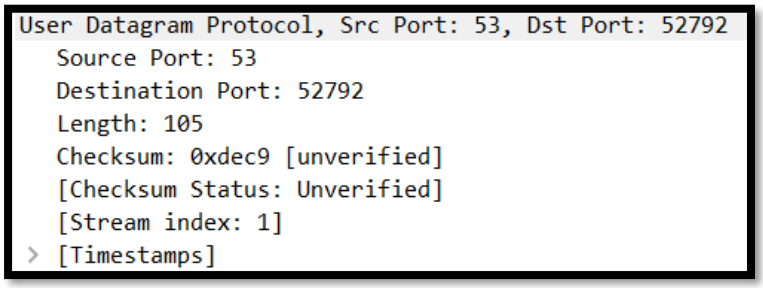
(i) **TCP** (Transmission Control Protocol): It is a standard that defines how to establish and maintain a network conversation through which application

programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.



Source Port	50466	The source port number is used by the sending host to help keep track of new incoming connections and existing data streams.
Destination Port	443	Similar to the source port, the destination port is used by the receiver to keep track of new incoming connections.
Sequence number	7404	The number assigned to the packet relative to the advent of the TCP connection.
Acknowledgement number	6068	It is the sequence number of the next byte the receiver expects to receive
Urgent Pointer	0	It is used to point to data that is urgently required. Here there is no such requirement and so its value is set to 0.

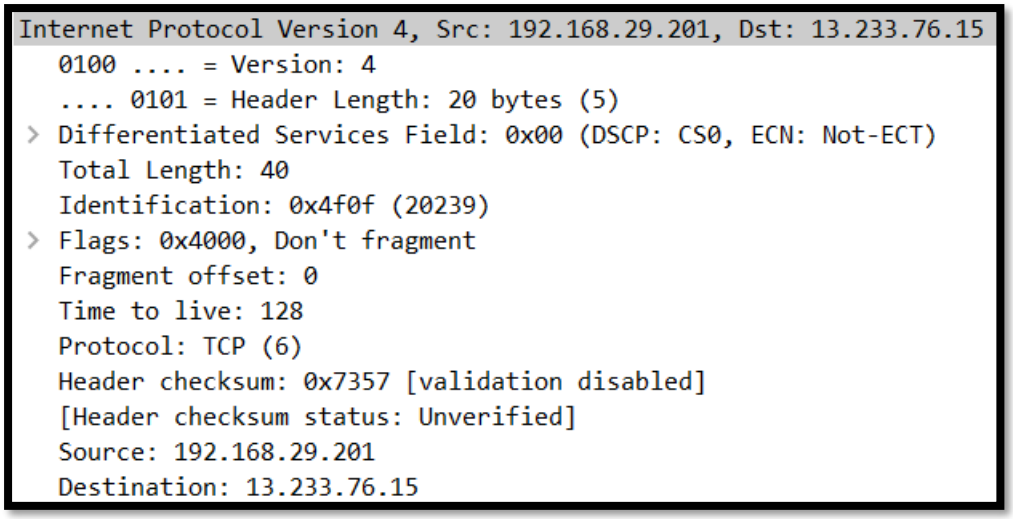
(ii) **UDP** (User Datagram Protocol): It is the simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.



Source Port	53	It is a 16-bit field and identifies the port of sender application.
Destination Port	52792	It identifies the port of receiver application.
Length	105	It identifies the combined length of UDP Header and Encapsulated data.
Checksum	0xdec9	It is calculated on UDP Header, encapsulated data and IP pseudo-header and used for error control.

3. Network layer

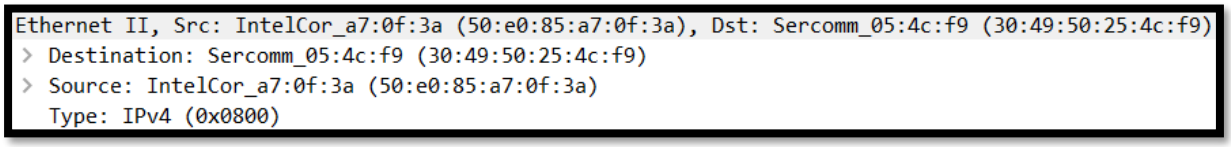
(i) **IPv4** (Internet Protocol version 4): It is the fourth version of the Internet Protocol (IP). It is one of the core protocols of standards-based internetworking methods in the Internet and other packet-switched networks.



Version	4	Indicates the IP version used.
Header Length	20 bytes (5)	Contains the length of the IP header.
Source	192.168.29.201	The IP address of the sender
Destination	13.233.76.15	The IP address of the receiver
Time to live	128	It indicates the maximum number of hops a datagram can take to reach the destination.

4. Link Layer

(i) **Ethernet II**: It is a data link layer protocol data unit and uses the underlying Ethernet physical layer transport mechanisms. In other words, a data unit on an Ethernet link transports an Ethernet frame as its payload.



Destination	30:49:50:25:4c:f9	Refers to the MAC address of the destination server
Source	50:e0:85:a7:0f:3a	Refers to the MAC address of the source
Type	IPv4 (0x0800)	Means that the upper layer protocol used is IPv4

Q2

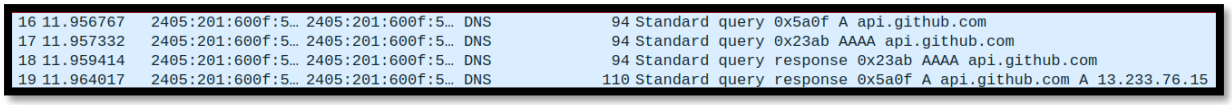
The important functionalities of GitHub Desktop Application are given below:

(i) Cloning a repository from the internet
(ii) Adding a local repository to GitHub
(iii) Pushing a repo onto GitHub server
(iv) Pulling a repo from GitHub server
(v) Branching a repository

- DNS protocol is used by all the above functionalities since resolving IP Address of GitHub server is a common step in all of them. DNS uses UDP for message smaller than 512 bytes (common requests and responses) and hence the packets in my experiment were also UDP packets.
- Similarly, TLS is also common to all the above functionalities in order to establish a secure connection between client and server.
- Similarly, IPv4 is used at network layer for all the functionalities. This is mainly due to the wide support for IPv4 protocol.
- Application data is transmitted by TLS for all the functionalities.
- TCP protocol is used in the transport layer for all the functionalities. This is because TCP is a reliable communication protocol which is a necessary attribute for the given functionalities.

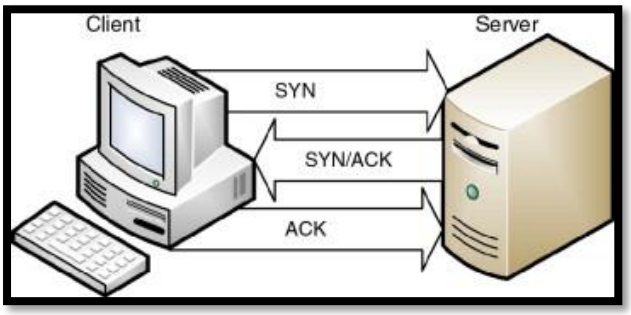
Q3

DNS Query: The first operation for any functionality is to resolve the IP Address of the GitHub server using a DNS Query. My PC sends a DNS Query to **"`api.github.com`"** which in turn sends a response to my request.



TCP 3-Way Handshake: A connection between the client and the server is established in 3 steps. As we can see in the image below, the connection is established using client port 49893 and server port 443.

20	11.965419	192.168.29.201	13.233.76.15	TCP	66 49893 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
21	12.006238	13.233.76.15	192.168.29.201	TCP	66 443 → 49893 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1436 SACK_PERM=1 WS=1024
22	12.006424	192.168.29.201	13.233.76.15	TCP	54 49893 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0



First, the client chooses an initial sequence number, set in the first SYN packet. Then, the server also chooses its own initial sequence number, set in the SYN/ACK packet shown in figure. Each side acknowledges each other's sequence number by incrementing it; this is the acknowledgement number. The use of sequence and acknowledgment numbers allows both sides to detect missing or out-of-order segments.

Once a connection is established, ACKs typically follow for each segment. The connection will eventually end with an RST (reset or tear down the connection) or FIN (gracefully end the connection).

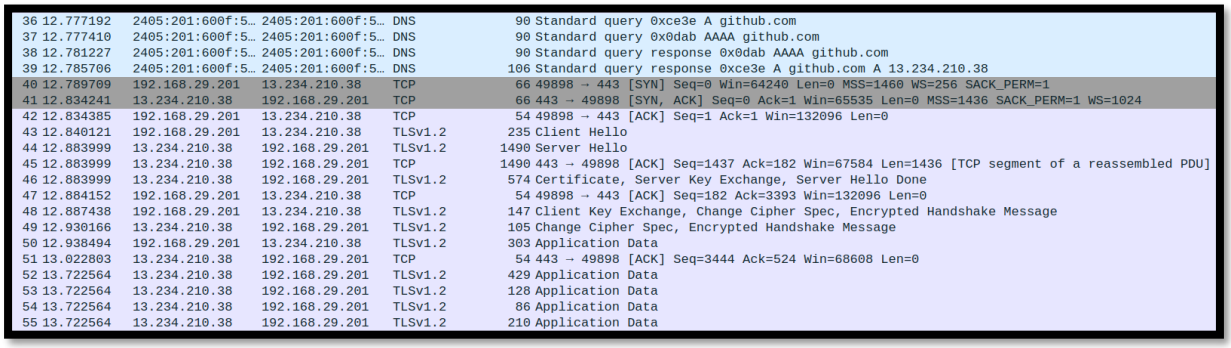
TLS Handshake: A TLS handshake is the process that kicks off a communication session that uses TLS encryption. During a TLS handshake, the two communicating sides exchange messages to acknowledge each other, verify each other, establish the encryption algorithms they will use, and agree on session keys. The first message in the TLS Handshake is the Client Hello which is sent by the client to initiate a session with the server. In return, the server responds with Server Hello and the Server Certificate (for authentication) along with a Server Key, which is used by the client to encrypt Client Key Exchange later in the process. Server Hello Done is an indication that the server is done and is now awaiting the client's response. The client responds with the Client Key and is issued a New Session Ticket. The TLS session is now established, and application data can be exchanged between both the server and the client.

43	12.840121	192.168.29.201	13.234.210.38	TLSv1.2	235 Client Hello
44	12.883999	13.234.210.38	192.168.29.201	TLSv1.2	1490 Server Hello
45	12.883999	13.234.210.38	192.168.29.201	TCP	1490 443 → 49898 [ACK] Seq=1437 Ack=182 Win=67584 Len=1436 [TCP segment of a reassembled PDU]
46	12.883999	13.234.210.38	192.168.29.201	TLSv1.2	574 Certificate, Server Key Exchange, Server Hello Done
47	12.884152	192.168.29.201	13.234.210.38	TCP	54 49898 → 443 [ACK] Seq=182 Ack=3393 Win=132096 Len=0
48	12.887438	192.168.29.201	13.234.210.38	TLSv1.2	147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
49	12.930166	13.234.210.38	192.168.29.201	TLSv1.2	105 Change Cipher Spec, Encrypted Handshake Message

Sending of Resources: After the handshaking is over, the exchange of information between client and server takes place via packets.

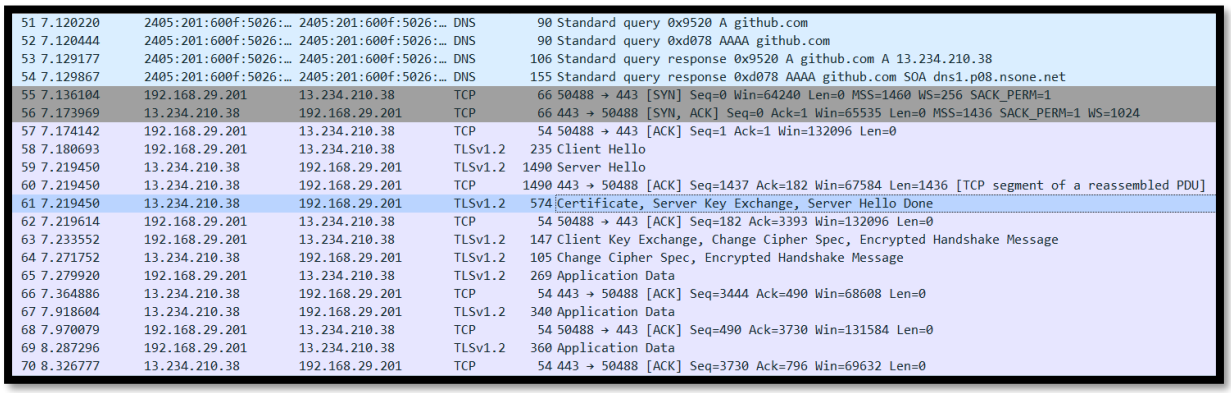
Two functionalities of GitHub Desktop application along with the sequence of messages exchanged is given below:

1. Cloning a repository from the internet



36 12.777192	2405:201:600f:5…	2405:201:600f:5…	DNS	90	Standard query 0xce3e A github.com
37 12.777410	2405:201:600f:5…	2405:201:600f:5…	DNS	90	Standard query 0x0dab AAAA github.com
38 12.781227	2405:201:600f:5…	2405:201:600f:5…	DNS	90	Standard query response 0x0dab AAAA github.com
39 12.785706	2405:201:600f:5…	2405:201:600f:5…	DNS	106	Standard query response 0xce3e A github.com A 13.234.210.38
40 12.789709	192.168.29.201	13.234.210.38	TCP	66	49898 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
41 12.834241	13.234.210.38	192.168.29.201	TCP	66	443 → 49898 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1436 SACK_PERM=1 WS=1024
42 12.834385	192.168.29.201	13.234.210.38	TCP	54	49898 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
43 12.840121	192.168.29.201	13.234.210.38	TLSv1.2	235	Client Hello
44 12.883999	13.234.210.38	192.168.29.201	TLSv1.2	1490	Server Hello
45 12.883999	13.234.210.38	192.168.29.201	TCP	1490	443 → 49898 [ACK] Seq=1437 Ack=182 Win=67584 Len=1436 [TCP segment of a reassembled PDU]
46 12.883999	13.234.210.38	192.168.29.201	TLSv1.2	574	Certificate, Server Key Exchange, Server Hello Done
47 12.884152	192.168.29.201	13.234.210.38	TCP	54	49898 → 443 [ACK] Seq=182 Ack=3393 Win=132096 Len=0
48 12.887438	192.168.29.201	13.234.210.38	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
49 12.930166	13.234.210.38	192.168.29.201	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
50 12.938494	192.168.29.201	13.234.210.38	TLSv1.2	303	Application Data
51 13.022803	13.234.210.38	192.168.29.201	TCP	54	443 → 49898 [ACK] Seq=3444 Ack=524 Win=68608 Len=0
52 13.722564	13.234.210.38	192.168.29.201	TLSv1.2	429	Application Data
53 13.722564	13.234.210.38	192.168.29.201	TLSv1.2	128	Application Data
54 13.722564	13.234.210.38	192.168.29.201	TLSv1.2	86	Application Data
55 13.722564	13.234.210.38	192.168.29.201	TLSv1.2	210	Application Data

2. Adding a local repository to GitHub



51 7.120220	2405:201:600f:5026:…	2405:201:600f:5026:…	DNS	90	Standard query 0x9520 A github.com
52 7.120444	2405:201:600f:5026:…	2405:201:600f:5026:…	DNS	90	Standard query 0xd078 AAAA github.com
53 7.129177	2405:201:600f:5026:…	2405:201:600f:5026:…	DNS	106	Standard query response 0x9520 A github.com A 13.234.210.38
54 7.129867	2405:201:600f:5026:…	2405:201:600f:5026:…	DNS	155	Standard query response 0xd078 AAAA github.com SOA dns1.p08.nsone.net
55 7.136104	192.168.29.201	13.234.210.38	TCP	66	50488 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
56 7.173969	13.234.210.38	192.168.29.201	TCP	66	443 → 50488 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1436 SACK_PERM=1 WS=1024
57 7.174142	192.168.29.201	13.234.210.38	TCP	54	50488 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
58 7.180693	192.168.29.201	13.234.210.38	TLSv1.2	235	Client Hello
59 7.219450	13.234.210.38	192.168.29.201	TLSv1.2	1490	Server Hello
60 7.219450	13.234.210.38	192.168.29.201	TCP	1490	443 → 50488 [ACK] Seq=1437 Ack=182 Win=67584 Len=1436 [TCP segment of a reassembled PDU]
61 7.219450	13.234.210.38	192.168.29.201	TLSv1.2	574	Certificate, Server Key Exchange, Server Hello Done
62 7.219614	192.168.29.201	13.234.210.38	TCP	54	50488 → 443 [ACK] Seq=182 Ack=3393 Win=132096 Len=0
63 7.233552	192.168.29.201	13.234.210.38	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
64 7.271752	13.234.210.38	192.168.29.201	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
65 7.279920	192.168.29.201	13.234.210.38	TLSv1.2	269	Application Data
66 7.364886	13.234.210.38	192.168.29.201	TCP	54	443 → 50488 [ACK] Seq=3444 Ack=490 Win=68608 Len=0
67 7.918604	13.234.210.38	192.168.29.201	TLSv1.2	340	Application Data
68 7.970079	192.168.29.201	13.234.210.38	TCP	54	50488 → 443 [ACK] Seq=490 Ack=3730 Win=131584 Len=0
69 8.287296	192.168.29.201	13.234.210.38	TLSv1.2	360	Application Data
70 8.326777	13.234.210.38	192.168.29.201	TCP	54	443 → 50488 [ACK] Seq=3730 Ack=796 Win=69632 Len=0

Q4

I performed the cloning operation at 3 different times of the day. The obtained values are given in the table below and can be verified using the given trace files.

Time	Throughput	RTT	Avg. Packet Size	Packets Lost	UDP Packets	TCP Packets	Number of responses per request
10 PM	9,49,000	40.81	1,298	0	9	2,453	11.50
3 PM	14,20,000	38.64	1,301	0	30	2,442	15.50
8 PM	15,88,000	39.86	1,211	0	251	2,527	14.68

Q5

The IP address of my PC remains constant over the course of experiments since it is connected to the same Wi-Fi Router. But the server IP address was different for different experiments. For the 1st and 2nd experiments, the server IP address was 13.234.210.38 but for the 3rd experiment, it was 13.234.176.102

The possible causes could be Load Balancing and improved reliability. These are explained in detail below.

Load balancing: In computing, load balancing refers to the process of distributing a set of tasks over a set of resources (computing units), with the aim of making their overall processing more efficient. Load balancing techniques can optimize the response time for each task, avoiding unevenly overloading compute nodes while other compute nodes are left idle.

Reliability: As we know that GitHub is one of the most heavily used websites over the world, therefore for reliable communication between hosts and its servers, it has multiple servers that attend to the requests of its users. At any point of time, the server that is free attends to the incoming user request. Due to this, server IP address can vary over different times of the day.

TRACE FILE LINK

https://drive.google.com/drive/folders/1vOQVWrlisK1F-vzbkBe0GCefBEcT3TdY?usp=sharing