SHREY VERMA
190101083
V.SHREY@IITG.AC.IN

Assignment 2,
CS342: Computer Networks Laboratory

31-08-2021

Drive Link for Traces :-

https://drive.google.com/drive/folders/1BvVLpUsTMLkKrByGtR3biLTJcmym4F_v?usp=sharing

Click Here

Question 1

1. Transport Layer

This layer is responsible for establishment of connection, maintenance of sessions, authentication and also ensures security.

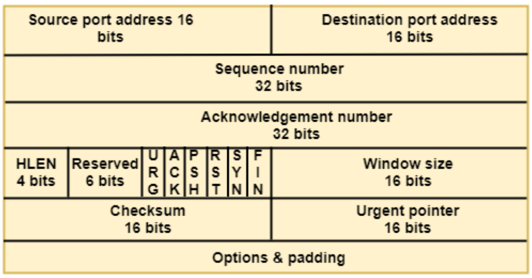
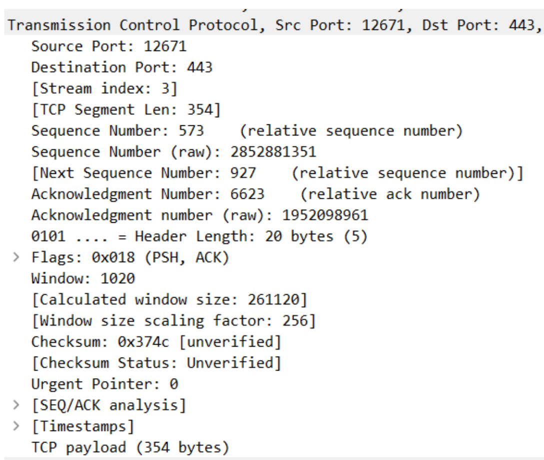


Figure: TCP Packet Format



```
Transmission Control Protocol, Src Port: 12671, Dst Port: 443,
    Source Port: 12671
    Destination Port: 443
    [Stream index: 3]
    [TCP Segment Len: 354]
    Sequence Number: 573    (relative sequence number)
    Sequence Number (raw): 2852881351
    [Next Sequence Number: 927    (relative sequence number)]
    Acknowledgment Number: 6623    (relative ack number)
    Acknowledgment number (raw): 1952098961
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 1020
    [Calculated window size: 261120]
    [Window size scaling factor: 256]
    Checksum: 0x374c [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (354 bytes)
```

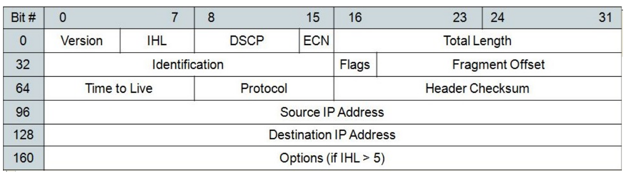
Figure: Trace of TCP packet

Transmission Control Protocol (TCP) : TCP packet consists of the following fields.

- **Header Length - 4 bits** : Specifies the size of the TCP header in 32-bit words.
- **Acknowledgment 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.
- **Flags Field - 6 bits** : Consists of the flags such as URG, ACK, PSH, RST, SYN, FIN.
- **Checksum - 16 bits** : It is used to check whether the header was damaged in transit or not.
- **Source Port and Destination Port - 16 bits each** : Identify the ports used at the endpoint nodes in the network connection.
- **Urgent pointer field - 16 bits** : if the URG flag is enabled, then this field is an offset from the sequence number indicating the last urgent data byte.

2. Network Layer

Network layer works for the transmission of data from one host to the other located in different networks.



Bit #	0	7	8	15	16	23	24	31
0	Version	IHL	DSCP	ECN	Total Length			
32	Identification				Flags	Fragment Offset		
64	Time to Live		Protocol		Header Checksum			
96	Source IP Address							
128	Destination IP Address							
160	Options (if IHL > 5)							

Figure: IPv4 Packet Format

```
v Internet Protocol Version 4, Src: 192.168.1.8, Dst: 13.107.6.163
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 394
    Identification: 0x734e (29518)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0xb061 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.8
    Destination Address: 13.107.6.163
```

Figure: Trace of IPv4 Packet

Internet Protocol Version 4 (IPv4) : IPv4 packet header contains 20 bytes of data and are notmally 32 bits long.

- **Identification** : This field is primarily used for uniquely identifying the group of fragments of a single IP datagram.
- **Fragment** : This sections specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram.
- **Internet Header Length** :This field specifies the size of of the header in multiples of 32 bits The minimum is 5 which means length of 5 x 32 bits=160 bits.
- **TTL (Time to live)** : It defines the hop limit and thus limits the lifespan or lifetime of data in a network.
- **Protocol** : Tells the Network layer at the destination host, to which Protocol this packet belongs.
- **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.

3. 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.

7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	46 – 1500 Bytes	4 Bytes
Preamble	SFD	Destination Address	Source Address	Type	Data Payload	Frame Check Sequence (FCS)

Figure: Ethernet II Packet Format

```
Ethernet II, Src: ASUSTekC_e3:08:f3 (04:d4:c4:e3:08:f3), Dst: TaicangT_63:0e:40 (5c:f9:fd:63:0e:40)
  v Destination: TaicangT_63:0e:40 (5c:f9:fd:63:0e:40)
      Address: TaicangT_63:0e:40 (5c:f9:fd:63:0e:40)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: ASUSTekC_e3:08:f3 (04:d4:c4:e3:08:f3)
      Address: ASUSTekC_e3:08:f3 (04:d4:c4:e3:08:f3)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

Figure: Trace of Ethernet II

Ethernet II : IPv4 packet header contains 20 bytes of data and are notmally 32 bits long.

- **Source and Destination MAC addresses** : This section specifies MAC address, which are unique identifiers assigned to the NICs present in the machines and these are globally unique.
- **Destination** : It indicates the address of the destination adapter.
- **EtherType** : This existence of this field differentiates between 802.3 and Ethernet II.
- **Cyclic Redundancy Check** : CRC, is a CRC-32 polynomial code for error detection
- **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
- **SFD** : It is the 8 bit value that marks the end of the preamble.

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.

Continued on next page

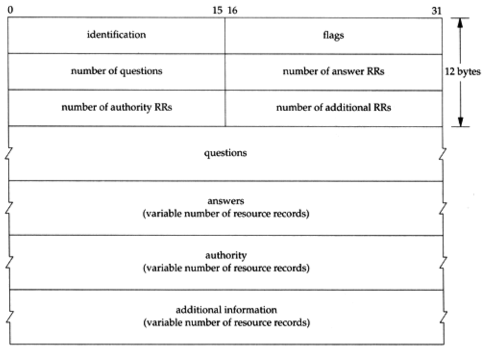


Figure: DNS Packet Format

```
Domain Name System (query)
  Transaction ID: 0xc97d
v 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
v Queries
  > github.com: type A, class IN
  [Response In: 888]
```

Figure: Trace of DNS packet

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

- **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

Some of the important functionalities of GitHub are as follows:-

- Push to a Repository
- Pull a Repository
- Creating a Repository
- Creating Branch of the Repository
- Cloning a Repository

DNS protocol: It is **used by the every functionality** of GitHub. This is used to **map domain names to IP addresses**, indicating the server's address (either MAC or IP) the client is trying to connect in order to use the application. DNS helps to translate numbered IP or MAC addresses to user-readable domains like google.com instead of some IP address.

TCP Protocol: It is **used by all functionalities** of GitHub. It ensures **reliable transport of data** from source to destination without any interceptions in between. It **uses handshaking protocol** on connection establishment and termination. It has **flow control**, **network congestion control** mechanism.

TLS Protocol: It is also **used by all the functionalities** of this application. It is a security layer protocol which **encrypts the application data** and thus **prevents hackers** to gain access to important information that might lead to breaches in security. **Login credentials etc. are safely transmitted from sender to receiver** without granting access to any third party because of the presence of this protocol.

Ethernet II: Being the most widely used data link layer protocol, it is **used by all the functionalities** of the application, since it has a **reliability**, **rate of data transfer** coupled with **flow control**. It also allows proper **error handling**.

UDP: 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**.

OCSP : The Online Certificate Status Protocol is used by GitHub to **check the status of server's certificate** issued by the server for performing various actions such as push or pull.

Question 3

Cloning a Repository

No.	Time	Source	Destination	Protocol	Length	Info
60	1.379427	13.234.210.38	192.168.1.8	TCP	60	443 → 2957 [ACK] Seq=2947 Ack=874 Win=69632 Len=0
61	1.623278	13.234.210.38	192.168.1.8	TLSv1.2	481	Application Data
62	1.623564	13.234.210.38	192.168.1.8	TLSv1.2	123	Application Data
63	1.623624	192.168.1.8	13.234.210.38	TCP	54	2957 → 443 [ACK] Seq=874 Ack=3443 Win=261632 Len=0
64	1.623633	192.168.1.8	13.234.210.38	TCP	54	[TCP Dup ACK 63#1] 2957 → 443 [ACK] Seq=874 Ack=3443 Win=261632 Len=0
65	1.626887	13.234.210.38	192.168.1.8	TLSv1.2	225	Application Data
66	1.627249	192.168.1.8	13.234.210.38	TLSv1.2	645	Application Data
67	1.627259	192.168.1.8	13.234.210.38	TCP	645	[TCP Retransmission] 2957 → 443 [PSH, ACK] Seq=874 Ack=3614 Win=261376 Len=591
68	1.648599	192.168.1.8	192.168.1.255	NBNS	92	Name query NB WPAD<00>
69	1.648617	192.168.1.8	192.168.1.255	NBNS	92	Name query NB WPAD<00>
70	1.662909	13.234.210.38	192.168.1.8	TCP	60	443 → 2957 [ACK] Seq=3614 Ack=1465 Win=70656 Len=0
71	1.906776	13.234.210.38	192.168.1.8	TLSv1.2	474	Application Data
72	1.914215	13.234.210.38	192.168.1.8	TLSv1.2	239	Application Data
73	1.914317	192.168.1.8	13.234.210.38	TCP	54	2957 → 443 [ACK] Seq=1465 Ack=4219 Win=262400 Len=0
74	1.914327	192.168.1.8	13.234.210.38	TCP	54	[TCP Dup ACK 73#1] 2957 → 443 [ACK] Seq=1465 Ack=4219 Win=262400 Len=0
75	1.925774	192.168.1.8	13.234.210.38	TLSv1.2	683	Application Data
76	1.925789	192.168.1.8	13.234.210.38	TCP	683	[TCP Retransmission] 2957 → 443 [PSH, ACK] Seq=1465 Ack=4219 Win=262400 Len=629
77	1.961524	13.234.210.38	192.168.1.8	TCP	60	443 → 2957 [ACK] Seq=4219 Ack=2094 Win=71680 Len=0

Figure: Trace when Cloning a Repository in GitHub

Data exchange happens using TCP protocol with the use of various ACKs and [PSH,ACK]s.
TCP's push capability accomplishes two things:

1. 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.

2. 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.

TLS ensures that the exchanged data is encrypted. Sometimes, packets may arrive out of order hence the data needs to be reassembled.

Submit a File

No.	Time	Source	Destination	Protocol	Length	Info
1862	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=629678 Win=1256448 Len=0
1863	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=642386 Win=1247232 Len=0
1864	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=646622 Win=1244160 Len=0
1865	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=650858 Win=1241088 Len=0
1866	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=659330 Win=1234944 Len=0
1867	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=663566 Win=1231872 Len=0
1868	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=670626 Win=1226752 Len=0
1869	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=674862 Win=1223680 Len=0
1870	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=681922 Win=1218560 Len=0
1871	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=687570 Win=1214464 Len=0
1872	11.398360	13.234.210.38	192.168.1.8	TCP	60	443 → 14978 [ACK] Seq=4119 Ack=691806 Win=1211392 Len=0
1873	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1151342 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1874	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1152754 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1875	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1154166 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1876	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1155578 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1877	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1156990 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1878	11.398547	192.168.1.8	13.234.210.38	TLSv1.2	1466	Application Data, Application Data
1879	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1159814 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]
1880	11.398547	192.168.1.8	13.234.210.38	TCP	1466	14978 → 443 [ACK] Seq=1161226 Ack=4119 Win=262400 Len=1412 [TCP segment of a reassembled PDU]

Figure: Trace found while submitting a file to a repository

The client sends application data to the server when we submit a file on GitHub and the **sever responds by sending ACK packets**. No Handshaking is observed in the middle of file submission.
As shown in the above image, the packets **Protocol Data Unit (PDU)** need to be reassembled because they might arrive out of order because of using different routes, to ensure load balancing.

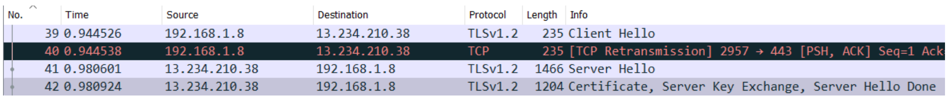
Handshaking

The handshaking process takes place in order to **establish rules for communication** when a computer tries to connect with another device. In addition to exchanging protocol information, handshaking also **verifies the quality or bandwidth of the connection**, as well as any **secure authority** that may be required to complete the connection between devices. Handshaking takes place only when the connection is established with a particular server IP.

Continued on next page

The following handshaking sequences were observed in the messages:-

1. TLS Handshaking

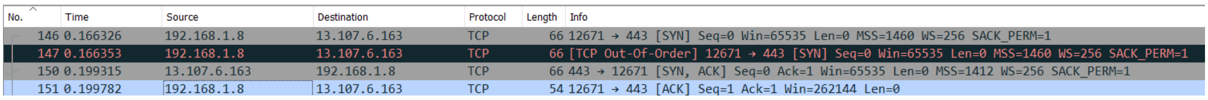


No.	Time	Source	Destination	Protocol	Length	Info
39	0.944526	192.168.1.8	13.234.210.38	TLSv1.2	235	Client Hello
40	0.944538	192.168.1.8	13.234.210.38	TCP	235	[TCP Retransmission] 2957 → 443 [PSH, ACK] Seq=1 Ack
41	0.980601	13.234.210.38	192.168.1.8	TLSv1.2	1466	Server Hello
42	0.980924	13.234.210.38	192.168.1.8	TLSv1.2	1204	Certificate, Server Key Exchange, Server Hello Done

Figure: Trace of TLS handshake

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

2. TCP connection establishment



No.	Time	Source	Destination	Protocol	Length	Info
146	0.166326	192.168.1.8	13.107.6.163	TCP	66	12671 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
147	0.166353	192.168.1.8	13.107.6.163	TCP	66	[TCP Out-Of-Order] 12671 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
150	0.199315	13.107.6.163	192.168.1.8	TCP	66	443 → 12671 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1412 WS=256 SACK_PERM=1
151	0.199782	192.168.1.8	13.107.6.163	TCP	54	12671 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0

Figure: Trace of TLS handshake

Step 1 (SYN) : In the first step, client wants to establish a connection with server, so it sends a segment with SYN (Synchronize Sequence Number) which informs server that client is likely to start communication and with what sequence number it starts its' segments.
Step 2 (SYN + ACK) : Server responds to the client request with SYN-ACK signal bits set.
Acknowledgement(**ACK**) signifies the response of segment it received and SYN signifies with what sequence number it is likely to start its' segments.
Step 3 (ACK) : In the final part client acknowledges the response of server and they both establish a reliable connection with which they will start the actual data transfer.

Question 4

Property	9AM	4PM	9PM
Throughput (Bytes per sec)	3033	2295	3591
RTT (ms)	29.6	36.9	43.2
Packet Size (Bytes)	172	186	166
Number of packets lost	13	18	19
Number of UDP & TCP packets	32 & 99	19 & 141	20 & 132
Responses per request sent	0.55 (31/56)	0.56 (18/32)	0.53 (32/60)

Note: RTT was observed by iRTT value of SYN,ACK packet.

Question 5

Yes, the IP address of destination changes during different times of the day. GitHub is a website having **huge traffic** at almost all times of the day. Thus it has **multiple servers** to **balance the load**, **increase the reliability** and **better network distribution** among its users. A server used in morning might be busy in afternoon so packet must go to other server. 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.

Time	IP
Morning	13.234.176.102
Afternoon	13.234.168.60
Night	13.234.210.38