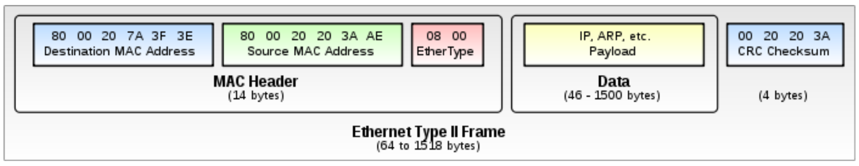
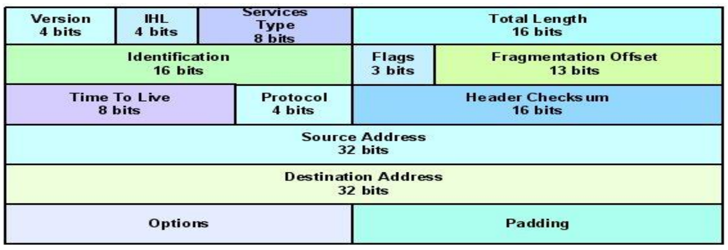
**Solution** 1.
The following are the protocols used by the application Google Hangouts as found using Wireshark :

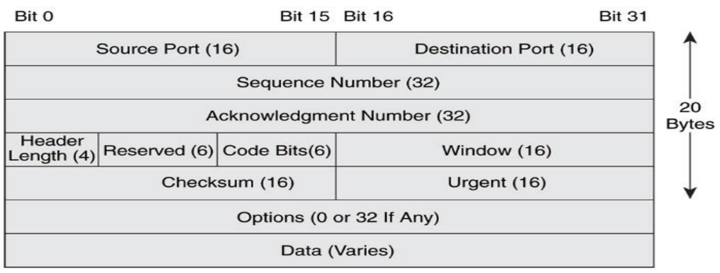
**Ethernet II**: This framing defines the two-octet EtherType field in an Ethernet frame, preceded by destination and source MAC addresses, that identifies an upper layer protocol encapsulated by the frame data.



**IPv4**: The IPv4 packet header consists of 14 fields, of which 13 are required. The 14th field is optional and aptly named: options. It is a connectionless protocol for use on packet switched networks. The encapsulated data is referred to as IP Payload. IP header contains all the necessary information to deliver the packet at the other end.

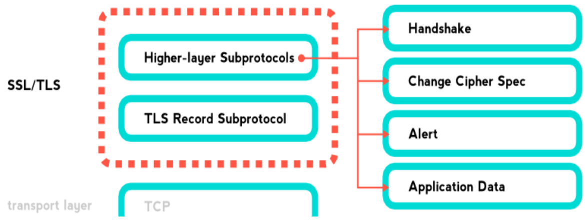


**TCP**: The TCP header contains 10 mandatory fields, and an optional extension field. It identifies source and destination ports and control data. The data section follows the header. Its contents are the payload data carried for the application. The length of the data section is not specified in the TCP segment header.

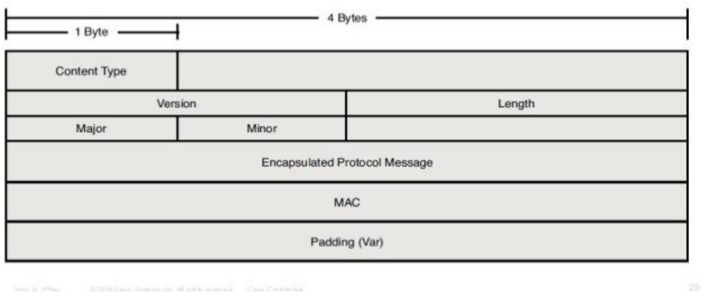


**HTTP**: HTTP headers allow the client and the server to pass additional information with the request or the response. There are various types of HTTP requests. A request header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks). Leading white space before the value is ignored.

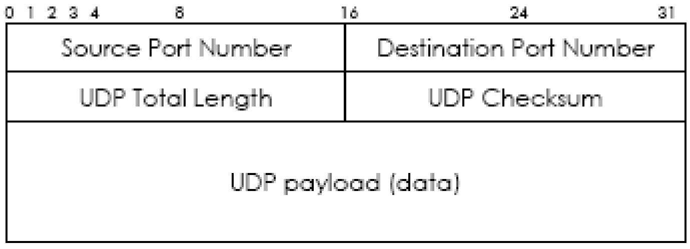
**SSL (TLSv1.2):** SSL/TLS is used in every browser worldwide to provide https (http secure) functionality. TLS is a transport layer cryptographic protocol that adds to the security of the layer. A general SSL record format is given on the right with the understanding of stacking and fuctions of the SSL layer is given below.



This is the general format of all TLS records.



**UDP**: UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.



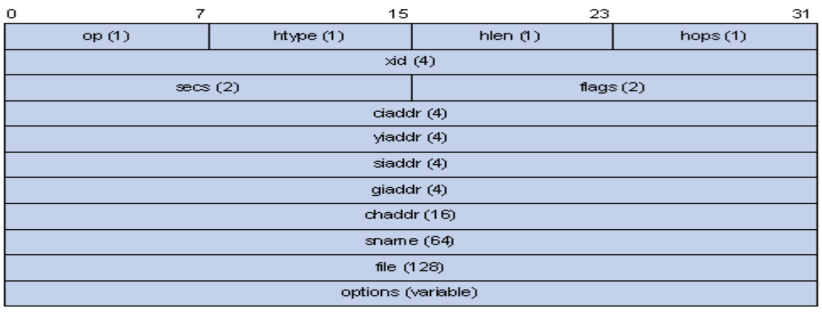
**DNS**: The domain name system (DNS) is the way that internet domain names are located and translated into internet protocol (IP) addresses. The domain name system maps the name people use to locate a website to the IP address that a computer uses to locate a website.

**STUN**: All STUN messages start with a STUN header, followed by a STUN payload. The payload is a series of STUN attributes. The header contains the following:

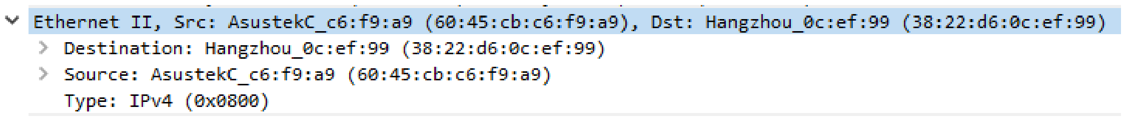
- STUN message type
- Message Length

- STUN Transaction ID

**LLMNR** :
The Link-Local Multicast Name Resolution (LLMNR) is a protocol based on the Domain Name System (DNS) packet format that allows both IPv4 and IPv6 hosts to perform name resolution for hosts on the same local link.



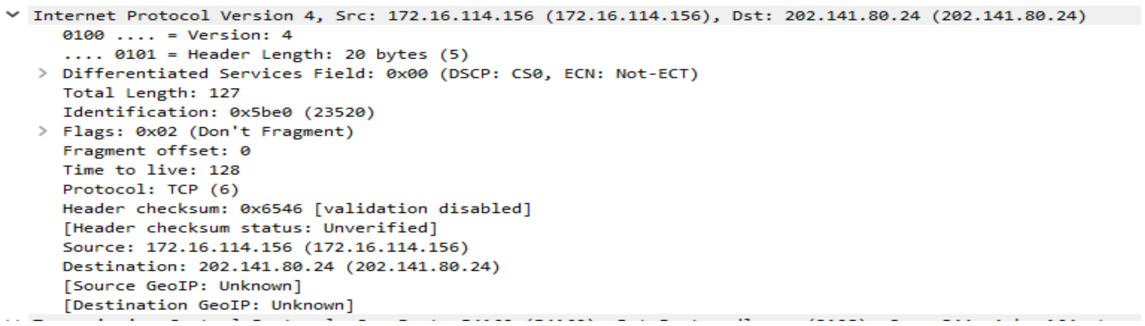
**Solution** 2.
Ethernet II:



```
v Ethernet II, Src: AsustekC_c6:f9:a9 (60:45:cb:c6:f9:a9), Dst: Hangzhou_0c:ef:99 (38:22:d6:0c:ef:99)
  > Destination: Hangzhou_0c:ef:99 (38:22:d6:0c:ef:99)
  > Source: AsustekC_c6:f9:a9 (60:45:cb:c6:f9:a9)
    Type: IPv4 (0x0800)
```

- The first 6 bytes in this header are Destination Address. It specifies the MAC address of the Destination/target system.
- The next 6 bytes specifies the MAC address of the source/transmitting system.
- EtherType is used to specify the higher level protocol frame encapsulated in the packet.

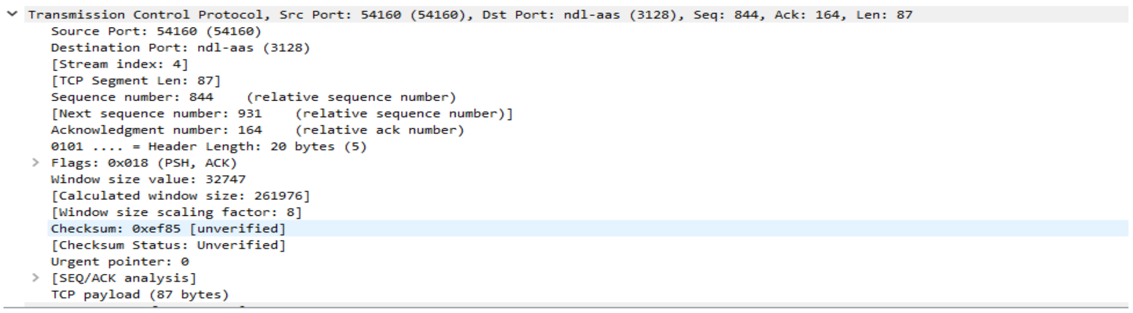
IPv4:



```
v Internet Protocol Version 4, Src: 172.16.114.156 (172.16.114.156), Dst: 202.141.80.24 (202.141.80.24)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 127
    Identification: 0x5be0 (23520)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x6546 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.114.156 (172.16.114.156)
    Destination: 202.141.80.24 (202.141.80.24)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
```

- Version: The version of the IP protocol used, which is 4 for this frame.
- IHL: The length of entire IP header, 20 bytes in this case.
- DSCP=CS0 implies best effort service.
- ECN is Explicit Congestion Notification; ECN requires specific support at both the Internet layer and the transport layer. So, ECN = not ECT implies Non ECN-Capable Transport.
- Total Length=127 is the length of entire IP packet.
- Identification: If IP packet is fragmented during the transmission, all the fragments contain same identification number. to identify original IP packet they belong to.
- Flags: As required by the network resources, if IP Packet is too large to handle, these 'flags' tells if they can be fragmented. Here the Don't Fragment bit is set, hence this frame will not be fragmented even if the IP packet is large.
- Fragment Offset: This offset tells the exact position of the fragment in the original IP Packet. In this packet it is set to 0.
- Time to Live: To avoid looping in the network, every packet is sent with some TTL value set, which tells the network how many routers (hops) this packet can cross. At each hop, its value is decremented by one and when the value reaches zero, the packet is discarded.
- Protocol: Tells the Network layer at the destination host, to which Protocol this packet belongs to. For example protocol number of ICMP is 1, TCP is 6 and UDP is 17.
- Header Checksum: This field is used to keep checksum value of entire header which is then used to check if the packet is received error-free. This also contains a pseudo header which again contains the source and destination IP addresses.

TCP:

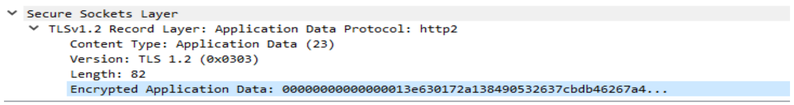


```
v Transmission Control Protocol, Src Port: 54160 (54160), Dst Port: ndl-aas (3128), Seq: 844, Ack: 164, Len: 87
    Source Port: 54160 (54160)
    Destination Port: ndl-aas (3128)
    [Stream index: 4]
    [TCP Segment Len: 87]
    Sequence number: 844    (relative sequence number)
    [Next sequence number: 931    (relative sequence number)]
    Acknowledgment number: 164    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 32747
    [Calculated window size: 261976]
    [Window size scaling factor: 8]
    Checksum: 0xef85 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
    TCP payload (87 bytes)
```

- Source Port 54160 tells us that the message originated from port 54160 on the source machine.
- Destination Port 3128 signifies the port to which the message is to be delivered on the target machine.
- The sequence number 844 is the sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

- Acknowledgment Number: If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.
- Data Offset: The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.
- Control Bits: 6 bits (from left to right): The flags that are set in the above frame are shown in **BOLD**
  **ACK**: Acknowledgment field significant
  **PSH**: Push Function When you send data, your TCP buffers it. So if you send a character it won't send it immediately but wait to see if you've got more. If push is set, your TCP will immediately create a segment (or a few segments) and push them.
- Checksum = 0xef85 is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.
- Urgent Pointer=0 communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. This field is only be interpreted in segments with the URG control bit set.

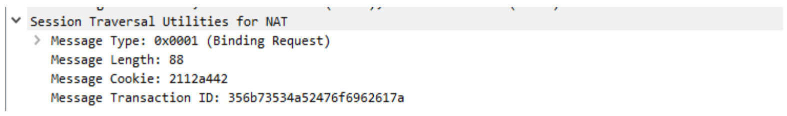
SSL:



```
˅ Secure Sockets Layer
  ˅ TLSv1.2 Record Layer: Application Data Protocol: http2
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 82
      Encrypted Application Data: 0000000000000013e630172a138490532637cbdb46267a4...
```

- Content Type = Application data, there are 4 types of SSL records.
- Version = TLSv1.2, The version of the TLS protocol used.
- Encrypted Application Data holds the application data encrypted based on the cipher suite decided between the client and server.

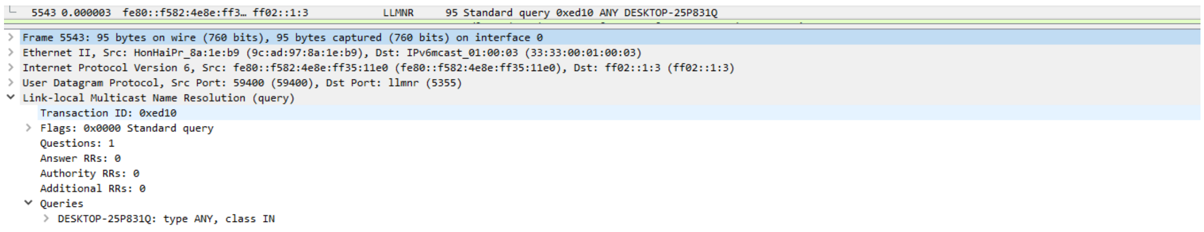
STUN:



```
˅ Session Traversal Utilities for NAT
   > Message Type: 0x0001 (Binding Request)
     Message Length: 88
     Message Cookie: 2112a442
     Message Transaction ID: 356b73534a52476f6962617a
```

Stun Message Type: Describes type of STUN request. Here it is binding request. There are six types of requests.
Message length – Indicates the total length of the STUN payload in bytes but does not include the 20 bytes header.
Transaction id –Is used to correlate requests and responses.

LLMNR:



```
└  5543 0.000003  fe80::f582:4e8e:ff3…  ff02::1:3          LLMNR     95 Standard query 0xed10 ANY DESKTOP-25P831Q

> Frame 5543: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface 0
> Ethernet II, Src: HonHaiPr_8a:1e:b9 (9c:ad:97:8a:1e:b9), Dst: IPv6mcast_01:00:03 (33:33:00:01:00:03)
> Internet Protocol Version 6, Src: fe80::f582:4e8e:ff35:11e0 (fe80::f582:4e8e:ff35:11e0), Dst: ff02::1:3 (ff02::1:3)
> User Datagram Protocol, Src Port: 59400 (59400), Dst Port: llmnr (5355)
˅ Link-local Multicast Name Resolution (query)
    Transaction ID: 0xed10
  > Flags: 0x0000 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ˅ Queries
    > DESKTOP-25P831Q: type ANY, class IN
```

- ID - A 16-bit identifier assigned by the program that generates any kind of query. Here it is 0xed10
- Flags – contains: 0x0000 Standard query
  0... .... .... .... = Response: Message is a query
  .000 0... .... .... = Opcode: Standard query (0)
  .... .0.. .... .... = Conflict: None
  .... ..0. .... .... = Truncated: Message is not truncated
  .... ...0 .... .... = Tentative: Not tentative

The following fileds are a 16 bit unsigned number:

- QDCOUNT : number of entries in question section ; Questions: 1
- ANCOUNT : number of resource records in answer section; Answer RRs: 0
- NSCOUNT : number of name server records in authority records section; Authority RRs: 0
- ARCOUNT : numbe of records in additional records section ; Additional RRs: 0

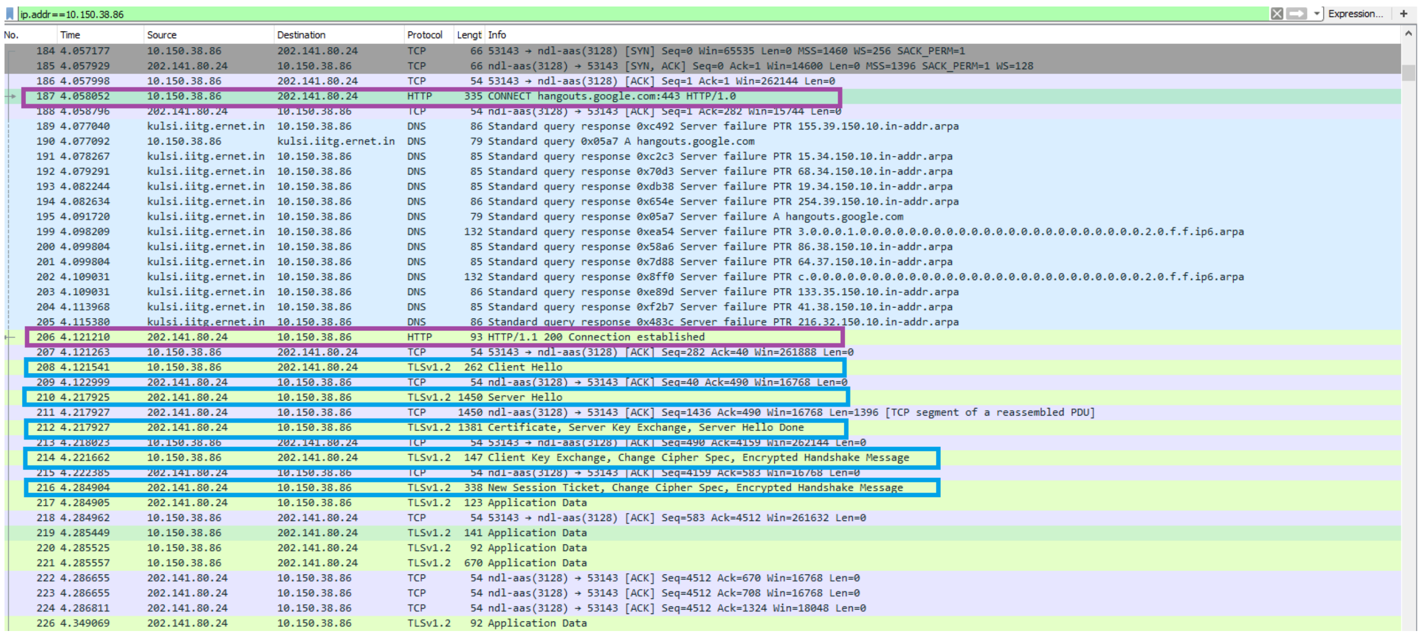
**Solution** 3. On opening or refreshing the hangouts page, the following are first transaction of the packets between client and server. These packets were captured using an IITG Wifi connection and the IP assigned to my computer is 10.150.38.86 .

When the site is opened, a connection request, (No. 187 in the image below), is sent from the client to the server to which the server responds, with an "HTTP/1.1 200 Connection Established" packet (No. 206 in the image below), status 200, implying it has successfully processed the client's request and agrees to communicate with client with HTTP version 1.1. The packets used for this are highlighted in purple in the screenshot below.

Following this, another handshake is done to set up the secure connection with the TLSv1.2 protocol. The packets associated with this are highlighted blue in the image

- The client sends a "Client hello" message to the server, along with the client's random value and supported cipher suites.
- The server responds by sending a "Server hello" message to the client, along with the server's random value.
- The server sends its certificate to the client for authentication and server sends the "Server hello done" message.

- The client creates a random Pre-Master Secret and encrypts it with the public key from the server's certificate, sending the encrypted Pre-Master Secret to the server.
- The server receives the Pre-Master Secret. The server and client each generate the Master Secret and session keys based on the Pre-Master Secret. The client sends "Change cipher spec" notification to server to indicate that the client will start using the new session keys for hashing and encrypting messages.
- Server receives "Change cipher spec" and switches its record layer security state to symmetric encryption using the session keys. Acknowledges using “New session Ticket”
- Client and server can now exchange application data over the secured channel they have established. All messages sent from client to server and from server to client are encrypted using session key.

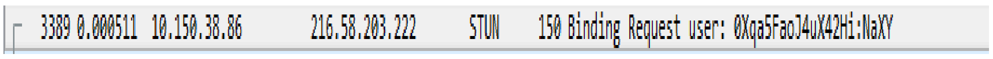


```
ip.addr==10.150.38.86                                                                     Expression...  +
No.   Time      Source              Destination         Protocol  Lengt Info
  184 4.057177  10.150.38.86        202.141.80.24       TCP        66 53143 → ndl-aas(3128) [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
  185 4.057929  202.141.80.24       10.150.38.86        TCP        66 ndl-aas(3128) → 53143 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1396 SACK_PERM=1 WS=128
  186 4.057998  10.150.38.86        202.141.80.24       TCP        54 53143 → ndl-aas(3128) [ACK] Seq=1 Ack=1 Win=262144 Len=0
  187 4.058052  10.150.38.86        202.141.80.24       HTTP      335 CONNECT hangouts.google.com:443 HTTP/1.0
  188 4.058796  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=1 Ack=282 Win=15744 Len=0
  189 4.077040  kulsi.iitg.ernet.in 10.150.38.86        DNS        86 Standard query response 0xc492 Server failure PTR 155.39.150.10.in-addr.arpa
  190 4.077092  10.150.38.86        kulsi.iitg.ernet.in DNS        79 Standard query 0x05a7 A hangouts.google.com
  191 4.078267  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0xc2c3 Server failure PTR 15.34.150.10.in-addr.arpa
  192 4.079291  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0x70d3 Server failure PTR 68.34.150.10.in-addr.arpa
  193 4.082244  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0xdb38 Server failure PTR 19.34.150.10.in-addr.arpa
  194 4.082634  kulsi.iitg.ernet.in 10.150.38.86        DNS        86 Standard query response 0x654e Server failure PTR 254.39.150.10.in-addr.arpa
  195 4.091720  kulsi.iitg.ernet.in 10.150.38.86        DNS        79 Standard query response 0x05a7 Server failure A hangouts.google.com
  199 4.098209  kulsi.iitg.ernet.in 10.150.38.86        DNS       132 Standard query response 0xea54 Server failure PTR 3.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa
  200 4.099804  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0x58a6 Server failure PTR 86.38.150.10.in-addr.arpa
  201 4.099804  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0x7d88 Server failure PTR 64.37.150.10.in-addr.arpa
  202 4.109031  kulsi.iitg.ernet.in 10.150.38.86        DNS       132 Standard query response 0x8ff0 Server failure PTR c.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa
  203 4.109031  kulsi.iitg.ernet.in 10.150.38.86        DNS        86 Standard query response 0xe89d Server failure PTR 133.35.150.10.in-addr.arpa
  204 4.113968  kulsi.iitg.ernet.in 10.150.38.86        DNS        85 Standard query response 0xf2b7 Server failure PTR 41.38.150.10.in-addr.arpa
  205 4.115380  kulsi.iitg.ernet.in 10.150.38.86        DNS        86 Standard query response 0x483c Server failure PTR 216.32.150.10.in-addr.arpa
  206 4.121210  202.141.80.24       10.150.38.86        HTTP       93 HTTP/1.1 200 Connection established
  207 4.121263  10.150.38.86        202.141.80.24       TCP        54 53143 → ndl-aas(3128) [ACK] Seq=282 Ack=40 Win=261888 Len=0
  208 4.121541  10.150.38.86        202.141.80.24       TLSv1.2   262 Client Hello
  209 4.122999  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=40 Ack=490 Win=16768 Len=0
  210 4.217925  202.141.80.24       10.150.38.86        TLSv1.2  1450 Server Hello
  211 4.217927  202.141.80.24       10.150.38.86        TCP      1450 ndl-aas(3128) → 53143 [ACK] Seq=1436 Ack=490 Win=16768 Len=1396 [TCP segment of a reassembled PDU]
  212 4.217927  202.141.80.24       10.150.38.86        TLSv1.2  1381 Certificate, Server Key Exchange, Server Hello Done
  213 4.218023  10.150.38.86        202.141.80.24       TCP        54 53143 → ndl-aas(3128) [ACK] Seq=490 Ack=4159 Win=262144 Len=0
  214 4.221662  10.150.38.86        202.141.80.24       TLSv1.2   147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
  215 4.222385  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=4159 Ack=583 Win=16768 Len=0
  216 4.284904  202.141.80.24       10.150.38.86        TLSv1.2   338 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
  217 4.284905  202.141.80.24       10.150.38.86        TLSv1.2   123 Application Data
  218 4.284962  10.150.38.86        202.141.80.24       TCP        54 53143 → ndl-aas(3128) [ACK] Seq=583 Ack=4512 Win=261632 Len=0
  219 4.285449  10.150.38.86        202.141.80.24       TLSv1.2   141 Application Data
  220 4.285525  10.150.38.86        202.141.80.24       TLSv1.2    92 Application Data
  221 4.285557  10.150.38.86        202.141.80.24       TLSv1.2   670 Application Data
  222 4.286655  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=4512 Ack=670 Win=16768 Len=0
  223 4.286655  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=4512 Ack=708 Win=16768 Len=0
  224 4.286811  202.141.80.24       10.150.38.86        TCP        54 ndl-aas(3128) → 53143 [ACK] Seq=4512 Ack=1324 Win=18048 Len=0
  226 4.349069  202.141.80.24       10.150.38.86        TLSv1.2    92 Application Data
```

A similar handshaking, but less complex, also occurs periodically between client and server for secure transmission of data. It also occurs for resuming the session. If the hangouts page remains idle and then I use it again, the following handshake occurs:

- The client sends a "Client hello" message using the Session ID of the session to be resumed.
- The server checks its session cache for a matching Session ID. If a match is found, and the server is able to resume the session, it sends a "Server hello" message with the Session ID.
- Client and server exchange "Change cipher spec" messages
- Client and server can now resume application data exchange over the secure channel.

When a video call is used in the hangouts, there is another exchange that happens over the STUN protocol. As shown in the image below:



```
3389 0.000511  10.150.38.86    216.58.203.222    STUN    150 Binding Request user: 0Xqa5FaoJ4uX42Hi:NaXY
```

- A STUN client (typically embedded in VoIP software, such as an IP PBX or IP Phone) sends a request to a STUN server to discover its public IP and port(s), and the STUN server returns a response. There are two types of requests; Binding Requests which are typically sent over UDP, and Shared Secret Requests, which are sent over TLS (secure communication) over TCP. Shared Secret Requests ask the server to return a temporary set of credentials which are then used in a Binding Request and Binding Response exchange, for the purposes of authentication and message integrity.
- Binding requests sent from the STUN client to the STUN server are used to determine the IP and port(s) bindings allocated by NAT’s. The STUN client sends a Binding Request to the STUN server, over UDP; the server examines the source IP address and port of the binding request, and copies them into a binding response that is sent back to the client.

**Solution** 4.

Ethernet: Ethernet is a link layer protocol in the TCP/IP stack, describing how networked devices can format data for transmission to other network devices on the same network segment, and how to put that data out on the network connection. This is essential for all types of applications.

TCP: It provides host-to-host connectivity at the Transport Layer. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application. TCP is optimized for accurate delivery rather than timely delivery and hence used by web browsers to connect to the internet for peer to peer file sharing, email transfer etc. This is important for all kinds of internet applications but also that for a chat application that the sent message is free of errors.

IP: The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet. Essential for all types of applications

SSL/TLSv1.2: SSL/TLS is used in every browser worldwide to provide https (http secure) functionality. Default usage in HTTPS is to verify server authenticity with trusted Certificate Authorities known by the browser, which is an important issue with chat applications so that private information about its client is secured.

STUN: STUN protocol plays an important role in VoIP implementations: to discover the presence of NAT and to learn and use the bindings allocate to the client by the NAT. The STUN client sends a Binding Request to the STUN server, over UDP; the server examines the source IP address and port of the binding request, and copies them into a binding response that is sent back to the client.

UDP: UDP is faster than TCP since it doesn't have the overhead of creating and maintaining a stream and hence preferred over TCP for live video streaming applications.

(LLMNR: This is not used by application but used by the device to query the name resolution. LLMNR is designed for both IPv4 and IPv6 clients when other name-resolution systems are not available, such as on a small-office or ad hoc network. LLMNR can also be used on corporate networks where DNS services are not available. Hence it is possible that at the time the DNS service was not available and the device used LLMNR protocol instead. )

HTTP: The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. As soon as a Web user opens their Web browser, the user is indirectly making use of HTTP. It is a protocol that runs on top of the TCP/IP suite of protocols. Connecting from the browser to the server for the char application is important.

**Solution 5.** The following is the data required:

| IP:port | Avg.RTT | Throughput (in Bytes) | Packet lost | Avg. Packet Size | No. of TCP packets | No of UDP packets | No of responses wrt to one request sent | Time | Network |
|---|---|---|---|---|---|---|---|---|---|
| 172.16.114.156:55151 | 0.00355 | 29k | 0 | 1109.5 | 2453 | 30584 | 0.129 | 17:30 | Ethernet(LAB) |
| 10.150.38.86 | 0.001310 | 10k | 0 | 521 | 235 | 1359 | 0.026 | 12:00 | Wifi(LAB) |
| 10.150.38.86 | 0.001004 | 44k | 0 | 521 | 20885 | 2579 | 0.1245 | 14:00 | Wifi(LAB) |
| 10.150.33.56 | 0.000947 | 3k | 0 | 712 | 493 | 3043 | 1.0 | 21:00 | Wifi (LAB) |
| 10.150.39.89 | 0.012 | 55k | 0 | 455 | 9434 | 1078 | 0.08666 | 19:00 | WIFI(CCD) |

**Solution 6.** An HTTP client normally uses the host name part of the URL to do a DNS lookup to connect to the requested host. Then is sends an HTTP GET of the rest of the URL. The host name is also included in that request. Now, if there is a proxy configured for a client it opens a connection to there and then sends the HTTP GET, with the complete URL, including the host name. The proxy then uses that information to setup its own connection with the intended host and forwards the HTTP GET. Any responses received are send back the originating client. Now, since, the campus is proxy authenticated network, all traffic is through this proxy server and thus we do not find the destination address of the original target server in the packets.

But we can extract the DNS name of the target, header of HTTP request. Some of them are:

- play.google.com:443
- people-pa.clients6.google.com:443
- ogs.google.com:443
- notifications.google.com:443
- lh5.googleusercontent.com:443
- www.google.com:443
- gce-beacons.gcp.gvt2.com:443 HTTP/1.1\r\n
- client-channel.google.com
- hangouts.google.com

We can then do the name resolving by ourselves, but there is no 100% guarantee that the Proxy uses the same address. This is because

- it might use a different nameserver that returns a different IP address
- the same name server might respond with a different IP address to different clients
- there might be multiple IP addresses returned and you don't know which one the proxy picked