CS342: Lab Report

Team M6

Spotify Analysis

The protocols used at different layers are:

Network Layer: IPv4 (Internet Protocol Version 4), IPv6 (Internet Protocol Version 6), IGMP (Internet Group Management Protocol)

Transport Layer: TCP (Transmission Control Protocol), UDP (User Datagram Protocol), QUIC (QUIC UDP Internet Connections)

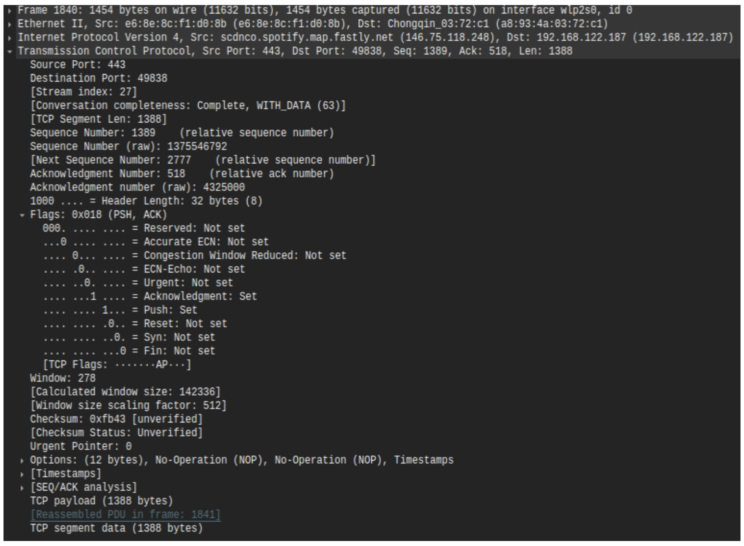
Application Layer: DNS (Domain Name System), mDNS (Multicast Domain Name System), SSDP (Simple Service Discovery Protocol), TLSv1.2 (Transport Layer Security version 1.2), TLSv1.3 (Transport Layer Security version 1.3)

Note that IGMP, SSDP are not directly used by Spotify

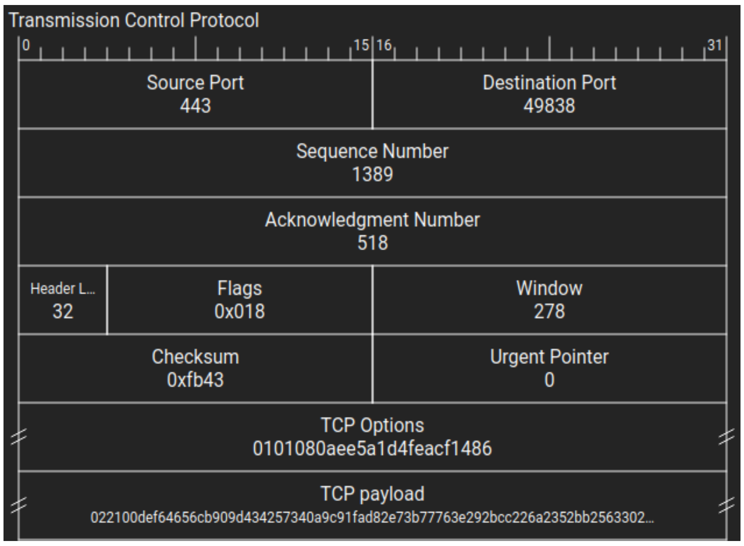
Protocols Analysis:

Transport Layer:

I) TCP (Transmission Control Protocol):



```
▸ Frame 1840: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: e6:8e:8c:f1:d0:8b (e6:8e:8c:f1:d0:8b), Dst: Chongqin_03:72:c1 (a8:93:4a:03:72:c1)
▸ Internet Protocol Version 4, Src: scdnco.spotify.map.fastly.net (146.75.118.248), Dst: 192.168.122.187 (192.168.122.187)
▾ Transmission Control Protocol, Src Port: 443, Dst Port: 49838, Seq: 1389, Ack: 518, Len: 1388
    Source Port: 443
    Destination Port: 49838
    [Stream index: 27]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 1388]
    Sequence Number: 1389    (relative sequence number)
    Sequence Number (raw): 1375546792
    [Next Sequence Number: 2777    (relative sequence number)]
    Acknowledgment Number: 518    (relative ack number)
    Acknowledgment number (raw): 4325000
    1000 .... = Header Length: 32 bytes (8)
  ▾ Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······AP···]
    Window: 278
    [Calculated window size: 142336]
    [Window size scaling factor: 512]
    Checksum: 0xfb43 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (1388 bytes)
    [Reassembled PDU in frame: 1841]
    TCP segment data (1388 bytes)
```



Transmission Control Protocol

Field	Value
Source Port	443
Destination Port	49838
Sequence Number	1389
Acknowledgment Number	518
Header L...	32
Flags	0x018
Window	278
Checksum	0xfb43
Urgent Pointer	0
TCP Options	0101080aee5a1d4feacf1486
TCP payload	022100def64656cb909d434257340a9c91fad82e73b77763e292bcc226a2352bb2563302...

The TCP packet contains the following data:

a) **Source Port:** Sender's port number (2 Bytes)

b) **Destination Port:** Receiver's port number (2 Bytes)

c) **Sequence Number**: Orders data (4 Bytes)

d) **Acknowledgment Number**: Confirms received data (4 Bytes)

e) **Header Length**: Data Offset (0.5 Bytes)

f) **Flags**: Indicates Packet type (1.5 Bytes: 0.75 Reserved+0.75 for flags)

Types of Flags:

1) ECN (Explicit Congestion Notification)
2) URG (Urgent)
3) ACK (Acknowledgment): Set in this case
4) PSH (Push): Set in this case
5) RST (Reset)
6) SYN (Synchronize)
7) FIN (Finish)

g) **Window Size**: Data acceptance limit (2 Bytes)

h) **Checksum:** For error detection (2 Bytes)

i) **Urgent Pointer**: To make urgent data (2 Bytes)

j) **TCP Options**: Extra TCP Options (Variable Length)

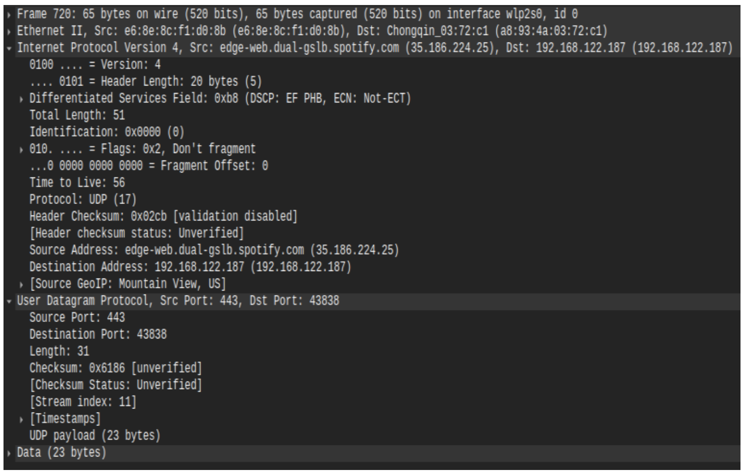
k) **Data**: Carries actual data (Variable Length)

TCP (Transmission Control Protocol) is a reliable, connection-oriented communication protocol used in computer networks to ensure data is delivered accurately and in the correct order.

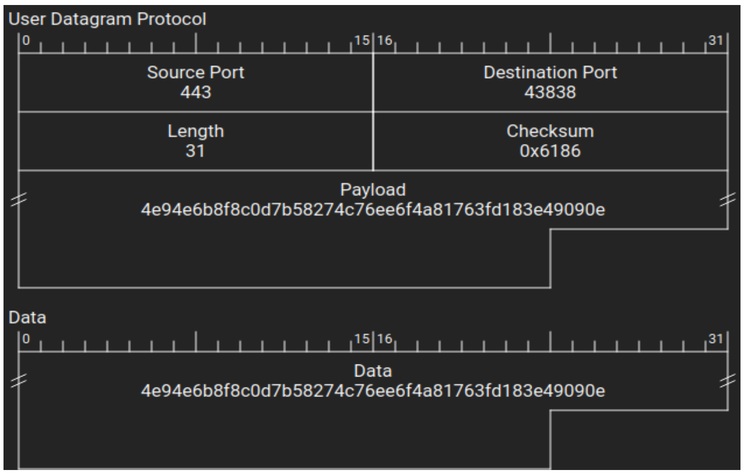
PSH flag is generally used to signal the client to take specific actions or initiate particular operations.

In this case the Spotify server sends a TCP packet with PSH and ACK flags set, indicating that it has data to send to the client immediately. This could be application data or a message to signal client to proceed with a TLS handshake.

II) UDP (User Datagram Protocol):



```
▸ Frame 720: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: e6:8e:8c:f1:d0:8b (e6:8e:8c:f1:d0:8b), Dst: Chongqin_03:72:c1 (a8:93:4a:03:72:c1)
▾ Internet Protocol Version 4, Src: edge-web.dual-gslb.spotify.com (35.186.224.25), Dst: 192.168.122.187 (192.168.122.187)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0xb8 (DSCP: EF PHB, ECN: Not-ECT)
    Total Length: 51
    Identification: 0x0000 (0)
  ▸ 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 56
    Protocol: UDP (17)
    Header Checksum: 0x02cb [validation disabled]
    [Header checksum status: Unverified]
    Source Address: edge-web.dual-gslb.spotify.com (35.186.224.25)
    Destination Address: 192.168.122.187 (192.168.122.187)
  ▸ [Source GeoIP: Mountain View, US]
▾ User Datagram Protocol, Src Port: 443, Dst Port: 43838
    Source Port: 443
    Destination Port: 43838
    Length: 31
    Checksum: 0x6186 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 11]
  ▸ [Timestamps]
    UDP payload (23 bytes)
▸ Data (23 bytes)
```



A UDP Packet contains the following components:

a) **Source Port**: Sender's port number (2 Bytes)

b) **Destination Port**: Receiver's port number (2 Bytes)

c) **Length**: Total Packet Size in bytes (2 Bytes)

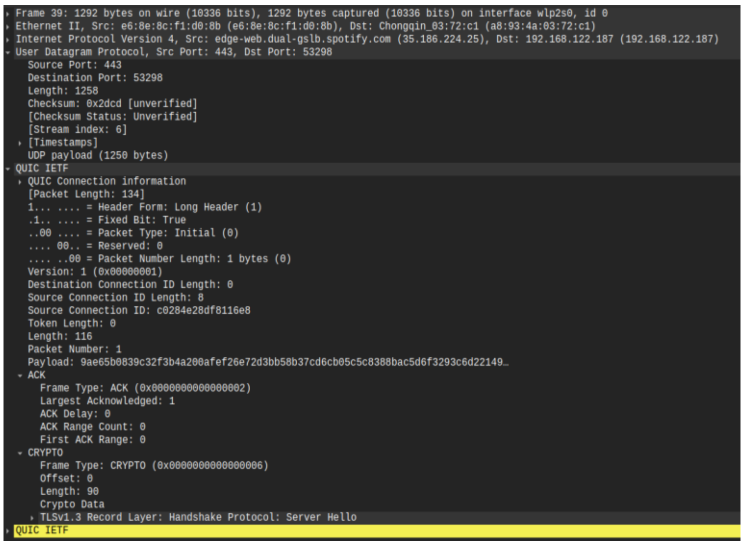
d) **Checksum**: Used for Error detection (2 Bytes)

e) **Data**: Actual Data (Variable Length)

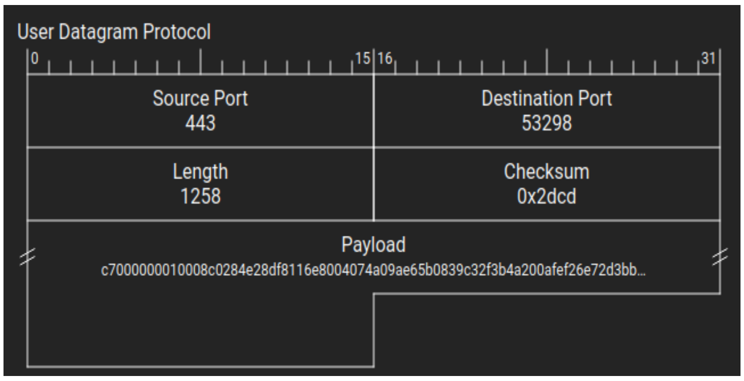
UDP (User Datagram Protocol) is a connectionless, minimalistic communication protocol employed in computer networks. Unlike TCP, UDP does not establish a connection before sending data, which makes it faster but less reliable, as it doesn't guarantee the delivery of packets or their order. It's commonly used for real-time applications like video streaming and online gaming, where speed is more crucial than perfect data integrity.

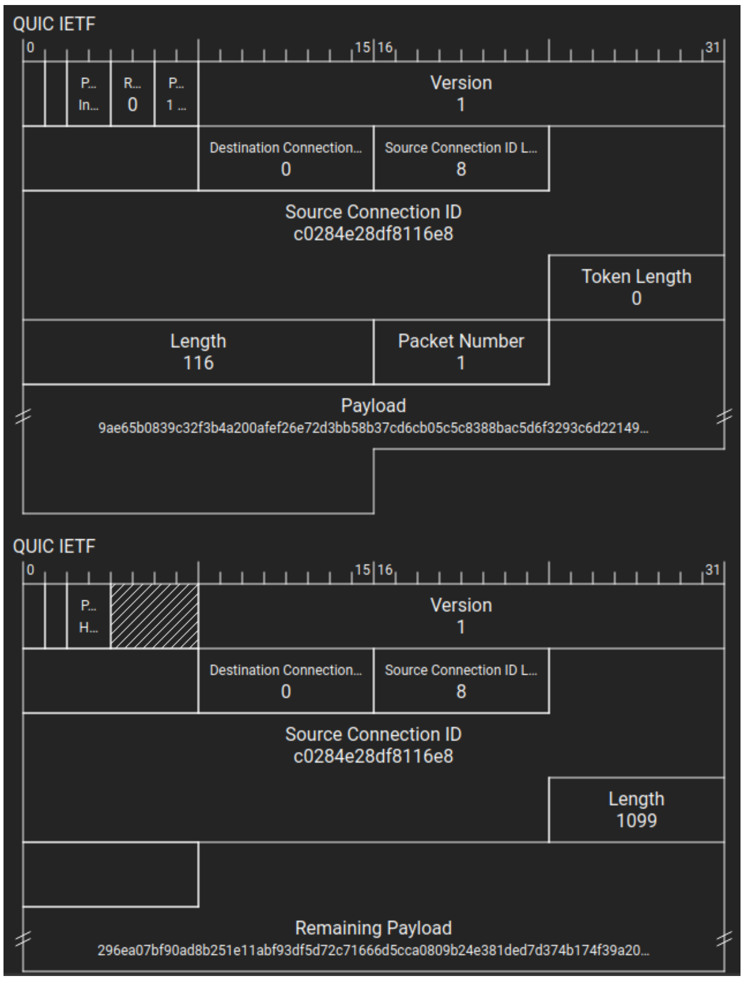
In this case Spotify may be using UDP for real time data like notifications and other features like service discovery and Telemetry.

III) QUIC (Quick UDP Internet Connections):



```
▸ Frame 39: 1292 bytes on wire (10336 bits), 1292 bytes captured (10336 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: e6:8e:8c:f1:d0:8b (e6:8e:8c:f1:d0:8b), Dst: Chongqin_03:72:c1 (a8:93:4a:03:72:c1)
▸ Internet Protocol Version 4, Src: edge-web.dual-gslb.spotify.com (35.186.224.25), Dst: 192.168.122.187 (192.168.122.187)
▾ User Datagram Protocol, Src Port: 443, Dst Port: 53298
    Source Port: 443
    Destination Port: 53298
    Length: 1258
    Checksum: 0x2dcd [unverified]
    [Checksum Status: Unverified]
    [Stream index: 6]
  ▸ [Timestamps]
    UDP payload (1250 bytes)
▾ QUIC IETF
  ▸ QUIC Connection information
    [Packet Length: 134]
    1... .... = Header Form: Long Header (1)
    .1.. .... = Fixed Bit: True
    ..00 .... = Packet Type: Initial (0)
    .... 00.. = Reserved: 0
    .... ..00 = Packet Number Length: 1 bytes (0)
    Version: 1 (0x00000001)
    Destination Connection ID Length: 0
    Source Connection ID Length: 8
    Source Connection ID: c0284e28df8116e8
    Token Length: 0
    Length: 116
    Packet Number: 1
    Payload: 9ae65b0839c32f3b4a200afef26e72d3bb58b37cd6cb05c5c8388bac5d6f3293c6d22149…
  ▾ ACK
      Frame Type: ACK (0x0000000000000002)
      Largest Acknowledged: 1
      ACK Delay: 0
      ACK Range Count: 0
      First ACK Range: 0
  ▾ CRYPTO
      Frame Type: CRYPTO (0x0000000000000006)
      Offset: 0
      Length: 90
      Crypto Data
    ▸ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
▸ QUIC IETF
```





A QUIC packet contains a UDP packet followed by one or more QUIC IETF packets.

The UDP Packet contains the following components:

a) **Source Por**t: 2 Bytes

b) **Destination Port**: 2 Bytes

c) **Length:** 2 Bytes

d) **Checksum**: 2 Bytes

e) **Data:** Variable Length

The QUIC IETF packet has the following components:

a) **Packet Number:** Represents the packet number, which is used for sequencing and acknowledgments (1 - 6 Bytes)

b) **Frames:** Frames carry data and control information

Types of Frames:

1) Stream Frame: Used for sending/receiving application data

2) ACK Frame: Provides acknowledgment for received packet

3) Connection Close Frame: Indicates abrupt connection close

4) PING Frame: Tests connection liveness and RTT

5) CRYPTO Frame: Carries Cryptographic handshake data

c) **Header Protection:** Contains data for header protection (Variable)

d) **Connection ID:** Helps identify QUIC connection endpoints (Variable)

e) **Packet Type:** Specifies the type of QUIC packet (typically 1 Byte)

f) **Packet Header form:** Indicates whether the packet is header is in long or short format (1 Bit)

g) **Reserved Bits:** Reserved for future use (2 Bits)

h) **Version:** Indicates QUIC protocol Version (4 Bytes)

i) **Payload:** Contains actual data being sent (Variable)

QUIC (Quick UDP Internet Connections) is a modern transport protocol developed by Google that is designed to improve the efficiency and speed of internet communication. It is built on top of UDP (User Datagram Protocol) but incorporates features from both UDP and TCP (Transmission Control Protocol). QUIC is designed to offer low-latency connections, improved security, and reliable data transfer, making it ideal for services that require fast and secure communication, such as web applications and media streaming.

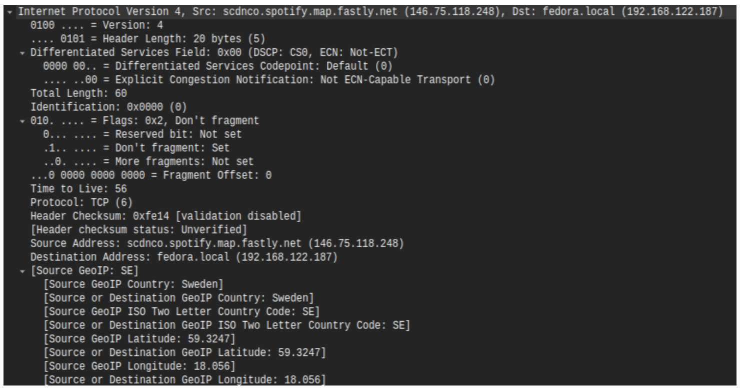
Spotify can use the QUIC (Quick UDP Internet Connections) protocol for various purposes to enhance its music streaming service and improve the user experience. Here are some potential use cases for which Spotify may utilize QUIC:

- Optimized Music Streaming: QUIC improves music streaming efficiency, reducing buffering and ensuring smoother playback.

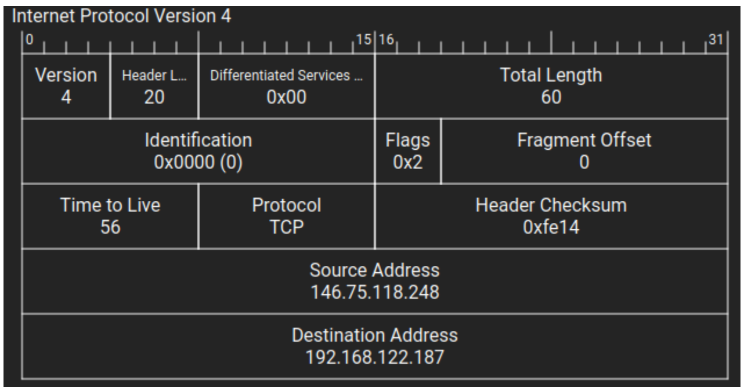
- Low Latency: QUIC minimizes latency for quick music playback and responsiveness.
- Connection Resilience: QUIC handles network disruptions well, ensuring uninterrupted music even during challenging conditions.
- Enhanced Security: QUIC provides encryption for user data and music streams, ensuring data privacy and security.
- Real-time Updates: Spotify uses QUIC for real-time synchronization of user libraries and playlists across devices.

Network Layer:

IV) IPv4:



```
▾ Internet Protocol Version 4, Src: scdnco.spotify.map.fastly.net (146.75.118.248), Dst: fedora.local (192.168.122.187)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▾ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      0000 00.. = Differentiated Services Codepoint: Default (0)
      .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 60
    Identification: 0x0000 (0)
  ▾ 010. .... = Flags: 0x2, Don't fragment
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 56
    Protocol: TCP (6)
    Header Checksum: 0xfe14 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: scdnco.spotify.map.fastly.net (146.75.118.248)
    Destination Address: fedora.local (192.168.122.187)
  ▾ [Source GeoIP: SE]
      [Source GeoIP Country: Sweden]
      [Source or Destination GeoIP Country: Sweden]
      [Source GeoIP ISO Two Letter Country Code: SE]
      [Source or Destination GeoIP ISO Two Letter Country Code: SE]
      [Source GeoIP Latitude: 59.3247]
      [Source or Destination GeoIP Latitude: 59.3247]
      [Source GeoIP Longitude: 18.056]
      [Source or Destination GeoIP Longitude: 18.056]
```



Internet Protocol Version 4

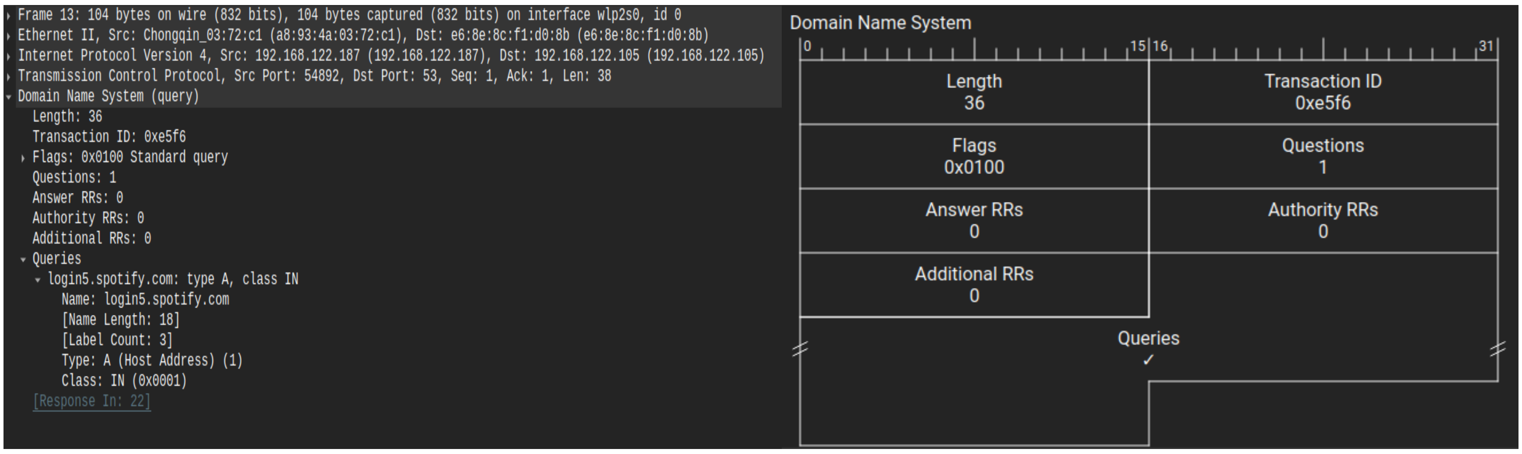
Version 4	Header L... 20	Differentiated Services ... 0x00	Total Length 60	
Identification 0x0000 (0)			Flags 0x2	Fragment Offset 0
Time to Live 56		Protocol TCP	Header Checksum 0xfe14	
Source Address 146.75.118.248				
Destination Address 192.168.122.187				

IPv4 (Internet Protocol version 4) is the fourth version of the Internet Protocol and is widely used for routing data packets across computer networks. An IPv4 packet consists of several components, each serving a specific purpose in the packet's header. Here are the key components of an IPv4 packet:

1) **Version** (4 bits): Identifies the IP version; in IPv4, this field is set to "4."
2) **Header Length** (4 bits): Indicates the length of the IP header in 32-bit words. It defines the starting point of the data within the packet.
3) **Type of Service (TOS) or Differentiated Services** (8 bits): Originally used for specifying the Quality of Service (QoS) level for the packet. In modern networks, it is often replaced with the Differentiated Services Code Point (DSCP) field, which is part of the IPv4 header and is used for traffic classification and marking.
4) **Total Length** (16 bits): Specifies the total length of the IP packet, including both the header and the payload (data). This field allows for packet lengths of up to 65,535 bytes.
5) **Identification** (16 bits): Used to uniquely identify a fragment of an IP datagram. It helps reassemble fragmented packets at the destination.
6) **Flags** (3 bits): Control flags related to fragmentation:
   a) Bit 0: Reserved (always set to 0).
   b) Bit 1: Don't Fragment (DF) flag.
   c) Bit 2: More Fragments (MF) flag.
7) **Fragment Offset** (13 bits): Indicates the position of the fragment in the original IP datagram when fragmentation is used.
8) **Time-to-Live (TTL)** (8 bits): Represents the maximum number of hops (routers) the packet can traverse before being discarded. It helps prevent packets from circulating endlessly in the network.
9) **Protocol** (8 bits): Specifies the higher-layer protocol to which the packet should be delivered (e.g., TCP, UDP, ICMP).
10) **Header Checksum** (16 bits): A checksum computed over the header to detect errors in the header's transmission.
11) **Source IP Address** (32 bits): The IP address of the sender (source) of the packet.
12) **Destination IP Address** (32 bits): The IP address of the intended recipient (destination) of the packet.

Application Layer:

V) DNS:



```
▸ Frame 13: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: Chongqin_03:72:c1 (a8:93:4a:03:72:c1), Dst: e6:8e:8c:f1:d0:8b (e6:8e:8c:f1:d0:8b)
▸ Internet Protocol Version 4, Src: 192.168.122.187 (192.168.122.187), Dst: 192.168.122.105 (192.168.122.105)
▸ Transmission Control Protocol, Src Port: 54892, Dst Port: 53, Seq: 1, Ack: 1, Len: 38
▾ Domain Name System (query)
    Length: 36
    Transaction ID: 0xe5f6
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ login5.spotify.com: type A, class IN
        Name: login5.spotify.com
        [Name Length: 18]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 22]
```

A DNS packet contains the following components:

A UDP packet followed by the DNS packet:

a) **Query Length:** Length of DNS query/response (2 Bytes)

b) **Transaction ID:** A unique identifier for the DNS query/response pair, used to match responses to their corresponding queries (2 Bytes)

c) **Flags:** This field contains various control flags (2 Bytes):

1. QR (1 bit): Query/Response flag (0 for query, 1 for response).
2. Opcode (4 bits): Specifies the query's purpose (e.g., standard query, inverse query, server status).
3. AA (1 bit): Authoritative Answer flag (1 if the responding server is authoritative for the domain).
4. TC (1 bit): Truncation flag (1 if the message was truncated).
5. RD (1 bit): Recursion Desired flag (1 if the client requests recursive resolution).
6. RA (1 bit): Recursion Available flag (1 if the server supports recursive resolution).
7. Z (3 bits): Reserved for future use (should be 0).
8. RCODE (4 bits): Response code (indicates the result of the query).

d) **Question Count:** Indicates the number of DNS questions (queries) in the packet (2 Bytes)

e) **Answers RRs (Resource Records):** Specifies the number of DNS resource records (RRs) in the answer section (2 Bytes)

f) **Authority RRs:** Indicates the number of DNS RRs in the authority section (2 Bytes)

g) **Additional RRs:** Specifies the number of DNS RRs in the additional section (2 Bytes)

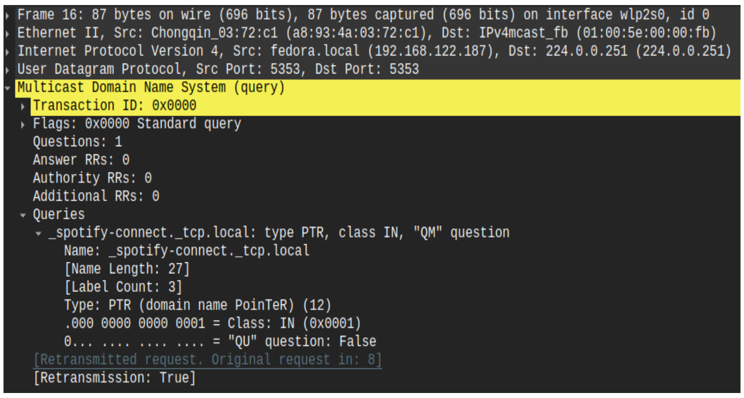
h) **Queries:**

1. Query Domain Name: Variable Length
2. Query Type: 2 Bytes
3. Query Class: 2 Bytes

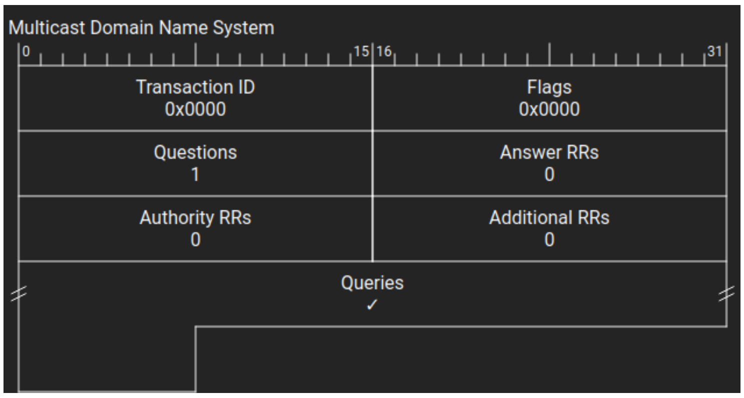
DNS (Domain Name System):

1) Purpose:
   a) DNS is a global and hierarchical system used to translate human-readable
   b) domain names (e.g., www.spotify.com) into IP addresses (35.186.224.47) on the
   c) internet.
2) Scope:
   a) DNS operates on the public internet and is used for resolving domain names
   b) across the entire internet.
3) Use Cases in Spotify:
   a) Spotify uses DNS for resolving its own domain names, such as the domain name
   b) for its servers, which allows clients to connect to Spotify's servers over the
   c) internet.
   d) DNS is used for resolving the domain names of third-party content delivery
   e) networks (CDNs) and service providers that Spotify may rely on for content
   f) delivery

VI) MDNS:



```
Frame 16: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface wlp2s0, id 0
Ethernet II, Src: Chongqin_03:72:c1 (a8:93:4a:03:72:c1), Dst: IPv4mcast_fb (01:00:5e:00:00:fb)
Internet Protocol Version 4, Src: fedora.local (192.168.122.187), Dst: 224.0.0.251 (224.0.0.251)
User Datagram Protocol, Src Port: 5353, Dst Port: 5353
Multicast Domain Name System (query)
  Transaction ID: 0x0000
  Flags: 0x0000 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    _spotify-connect._tcp.local: type PTR, class IN, "QM" question
      Name: _spotify-connect._tcp.local
      [Name Length: 27]
      [Label Count: 3]
      Type: PTR (domain name PoinTeR) (12)
      .000 0000 0000 0001 = Class: IN (0x0001)
      0... .... .... .... = "QU" question: False
  [Retransmitted request. Original request in: 8]
  [Retransmission: True]
```



A MDNS packet has the following components:

a) **Transaction ID**: 2 Bytes

b) **Flags**: 2 Bytes

c) **Number of Questions**: 2 Bytes

d) **Answer RRs**: 2 Bytes

e) **Authority RRs**: 2 Bytes

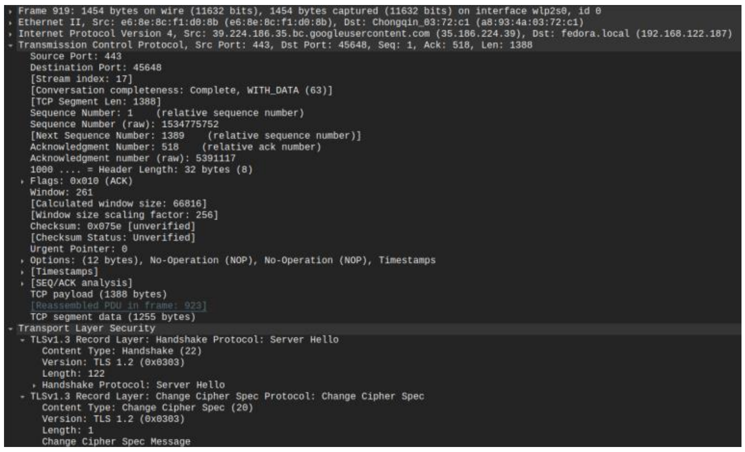
f) **Additional RRs**: 2 Bytes

g) **Query**: Variable Length

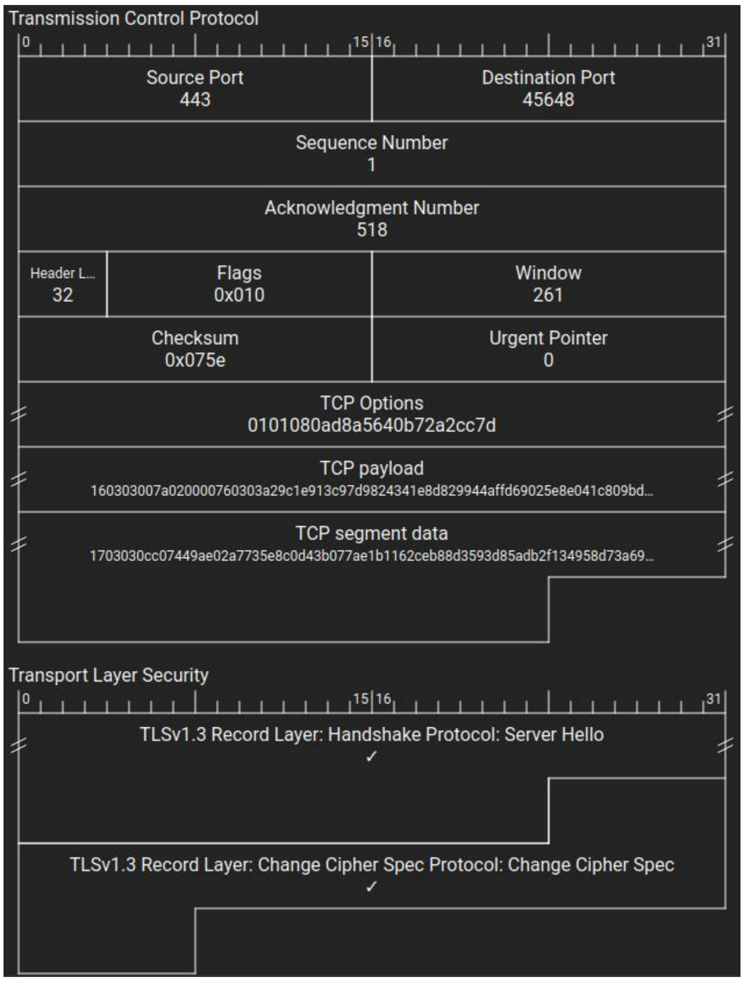
mDNS (Multicast Domain Name System):

1) **Purpose:**
   a) mDNS is designed for local network service discovery within a limited broadcast domain, typically a home or small office network.
2) **Scope:**
   a) mDNS operates within a local network, and it does not extend beyond the boundaries of that network segment.
3) **Use Cases in Spotify:**
   a) **Local Device Discovery:** Spotify may use mDNS to discover and connect to devices like speakers, media players, or smart TVs on the same local network. This enables users to stream music to nearby devices.
   b) **Local Media Server Discovery:** If users have local media servers or NAS devices, mDNS can help Spotify discover and access these servers, allowing users to play their locally stored music within the Spotify app.
   c) **Multi-Room Audio:** For scenarios involving multi-room audio setups, mDNS can assist in identifying and coordinating playback across multiple audio zones or speakers in a user's home network.

VII) TLSv1.3:



```
› Frame 919: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits) on interface wlp2s0, id 0
› Ethernet II, Src: e6:8e:8c:f1:d0:8b (e6:8e:8c:f1:d0:8b), Dst: Chongqin_03:72:c1 (a8:93:4a:03:72:c1)
› Internet Protocol Version 4, Src: 39.224.186.35.bc.googleusercontent.com (35.186.224.39), Dst: fedora.local (192.168.122.187)
⌄ Transmission Control Protocol, Src Port: 443, Dst Port: 45648, Seq: 1, Ack: 518, Len: 1388
    Source Port: 443
    Destination Port: 45648
    [Stream index: 17]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 1388]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1534775752
    [Next Sequence Number: 1389    (relative sequence number)]
    Acknowledgment Number: 518    (relative ack number)
    Acknowledgment number (raw): 5391117
    1000 .... = Header Length: 32 bytes (8)
  › Flags: 0x010 (ACK)
    Window: 261
    [Calculated window size: 66816]
    [Window size scaling factor: 256]
    Checksum: 0x075e [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  › Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  › [Timestamps]
  › [SEQ/ACK analysis]
    TCP payload (1388 bytes)
    [Reassembled PDU in frame: 923]
    TCP segment data (1255 bytes)
⌄ Transport Layer Security
  ⌄ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 122
    › Handshake Protocol: Server Hello
  ⌄ TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```



Transmission Control Protocol
0
15 16
31
Source Port
443
Destination Port
45648
Sequence Number
1
Acknowledgment Number
518
Header L...
32
Flags
0x010
Window
261
Checksum
0x075e
Urgent Pointer
0
TCP Options
0101080ad8a5640b72a2cc7d
TCP payload
160303007a020000760303a29c1e913c97d9824341e8d829944affd69025e8e041c809bd...
TCP segment data
1703030cc07449ae02a7735e8c0d43b077ae1b1162ceb88d3593d85adb2f134958d73a69...
Transport Layer Security
0
15 16
31
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
✓
TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
✓

A TLS packet has a TCP packet with various options followed by a TLS data which contains the following:

- **Handshake:** TLS 1.3 begins with a handshake between the client and server. This phase involves messages like Client Hello and Server Hello, where the client and server agree on cryptographic parameters.
- **Record Layer:** After the handshake, all data exchanged between the client and server is encrypted and transmitted within the record layer. This ensures the confidentiality and integrity of the communication.
- **Cipher Suites:** TLS 1.3 supports a set of cipher suites that specify the encryption and authentication algorithms used during the handshake and for encrypting application data. These cipher suites determine the strength and security of the encryption.
- **Key Exchange:** The key exchange process allows the client and server to securely exchange encryption keys.
- **Authentication (Optional):** TLS 1.3 optionally supports client and server authentication. During this phase, certificates may be exchanged to verify the identities of both parties.

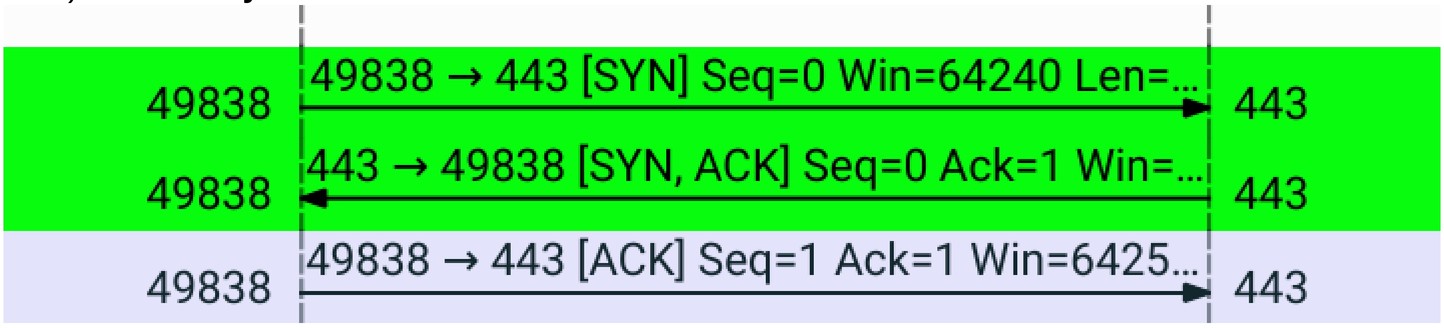
TLS 1.3 is designed to provide robust security while minimizing latency and improving performance compared to previous versions. Its streamlined design and strong cryptographic features make it a crucial protocol for securing data transmitted over the internet.

Sequence of messages being sent for various actions:

1) Login: TCP, TLSv1.3
2) Streaming: QUIC, UDP
3) Pausing: QUIC, TCP, TLSv1.3

Handshaking Observed:

1) **TCP 3-way Handshake observed:**



```
49838   49838 → 443 [SYN] Seq=0 Win=64240 Len=...   443
49838   443 → 49838 [SYN, ACK] Seq=0 Ack=1 Win=...  443
49838   49838 → 443 [ACK] Seq=1 Ack=1 Win=6425...   443
```

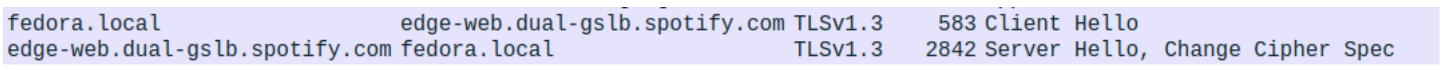
The TCP 3-way handshake, also known as the TCP handshake or SYN-ACK handshake, is a process used to establish a reliable connection between two devices over a TCP/IP network. This handshake is an essential part of the TCP protocol and ensures that both the sender and receiver are ready to exchange data. Here's how the TCP 3-way handshake works:

1) **Step 1: SYN (Synchronize)**:
   a) The initiating device, often referred to as the client, wants to establish a connection with the receiving device, which is the server.
   b) The client sends a TCP packet with the SYN (synchronize) flag set to the server.
   c) The packet also includes an initial sequence number (ISN) that the client chooses. This sequence number is used to keep track of the data exchanged during the connection.
2) **Step 2: SYN-ACK (Synchronize-Acknowledge)**:
   a) When the server receives the SYN packet, it acknowledges the client's request by sending back a TCP packet.
   b) The server sets the SYN flag and the ACK (acknowledge) flag in its packet, indicating that it received the client's request and is ready to establish a connection.
   c) Similar to the client, the server also selects an initial sequence number (ISN).
3) **Step 3: ACK (Acknowledge)**:
   a) Upon receiving the SYN-ACK packet from the server, the client sends an acknowledgment packet back to the server.
   b) The client sets the ACK flag in its packet and acknowledges the server's ISN.
   c) At this point, the connection is established, and both sides are ready to exchange data in a reliable and synchronized manner.

After the TCP 3-way handshake is complete, the client and server can start sending data packets to each other with confidence that the connection is established and reliable. The sequence numbers exchanged during the handshake help ensure that data is correctly ordered, and any lost or duplicated packets can be detected and retransmitted as needed.

This handshake process is fundamental to the reliability and integrity of data transfer over TCP/IP networks and is used in various internet protocols, including HTTP (for web browsing), FTP (for file transfer), and many others.

2) **TLS Handshake:**



```
fedora.local                  edge-web.dual-gslb.spotify.com TLSv1.3    583 Client Hello
edge-web.dual-gslb.spotify.com fedora.local                  TLSv1.3   2842 Server Hello, Change Cipher Spec
```

The TLS (Transport Layer Security) handshake is a process that occurs at the beginning of a TLS-secured communication to establish a secure connection between a client (e.g., a web browser) and a server (e.g., a web server). This handshake process ensures that both parties agree on encryption parameters, exchange cryptographic keys, and verify each other's identities. Let's break down the TLS handshake, including the Client Hello, Server Hello, and Change Cipher Spec phases:

Step 1: Client Hello

1) **Client Initiates**: The TLS handshake begins with the client sending a message known as the "Client Hello" to the server.
2) **Cipher Suites**: In the Client Hello message, the client specifies a list of cipher suites it supports. Each cipher suite defines a combination of encryption and authentication algorithms. The client arranges the list in order of preference.
3) **Random Data**: The Client Hello includes a random value generated by the client. This random value is used as part of the key derivation process.
4) **Session ID (Optional)**: The client can include a session ID if it wishes to resume a previous TLS session. This helps reduce the overhead of a full handshake.
5) **Supported TLS Versions**: The client indicates the TLS versions it supports.
6) **Extensions (Optional)**: The Client Hello message may contain various extensions for additional features or negotiation, such as Server Name Indication (SNI) for specifying the hostname of the server.

Step 2: Server Hello + Change Cipher Spec

1) **Server Responds**: Upon receiving the Client Hello, the server processes the request and selects the most suitable cipher suite based on its own capabilities and the client's preferences.
2) **Server Hello**: The server responds with a "Server Hello" message, which includes:
   a) The chosen cipher suite (selected from the client's list).
   b) Its own random value.
   c) An optional session ID (if session resumption is supported and chosen).
   d) TLS version (typically the highest supported version that both client and server support).
   e) Any applicable extensions (e.g., SNI).
3) **Change Cipher Spec (CCS)**:
   a) Immediately following the Server Hello, the server sends a "Change Cipher Spec" message. This message is crucial as it signals a shift in encryption parameters. It informs the client that subsequent communication will be encrypted using the negotiated parameters.

At this point, both the client and server have agreed on encryption parameters, exchanged random values, and are ready to establish a secure, encrypted connection. The Change Cipher Spec message is a critical part of this process, as it marks the transition from unencrypted communication to encrypted communication, ensuring data confidentiality and integrity. After this step, all data exchanged between the client and server is encrypted using the agreed-upon cipher suite and encryption keys.

Throughput and RTT at different times of the day in different actions:

Idle:

	12 PM	7 PM	11 PM
Throughput	3.306 kbps	7.030 bits/s	82300
RTT	0.08655861492	0.09352664231s	0.07982351682s
Packet Size	68 b	151 b	86 b
No. of packets lost	0	0	0
Number of UDP	159	148	175
Number of TCP	1148	1005	996
Response per Request	1.17434869	1.34944532	1.26875201

Streaming Audio:

	12 PM	7 PM	11 PM
Throughput	45 kbps	51 kbps	49 kbps
RTT	0.09125485942	0.08186472591	0.07991283872
Packet Size	652 b	712 b	667 b
No. of packets lost	0	0	0
Number of UDP	2906	3112	2886
Number of TCP	1213	989	1145
Response per Request	1.22474693	1.33638479	1.26747236