CS 342: Computer Networks Laboratory
Assignment 02

Name	Roll No.
Aasneh Prasad	210101001
Pranav Jain	210101078
Sanjana Siri Kolisetty	210101093

As instructed, we have installed Wireshark and learnt the basics of capturing packets and filtering content from them. The task given to us was to perform the specified network activities, capture relevant packets using Wireshark, and then write a comprehensive report documenting our observations, findings, and insights.

We used ***Netflix Application*** for our ***Wireshark*** analysis.

The available functionalities that we observed are:

- Starting the application
- Playing a video
- Pausing a video
- Jumping forward in the video
- Increasing speed of play
- Going to next video
- Closing the video
- Closing the application

❖ **Note:** We were connected to IITG_CONNECT during the analysis. So, in order to capture our packets, we have highlighted all the packets whose source or destination address is the client's (PC) green. We have also closed all the other windows/tabs to minimize the packets from our PC.

Task 1:

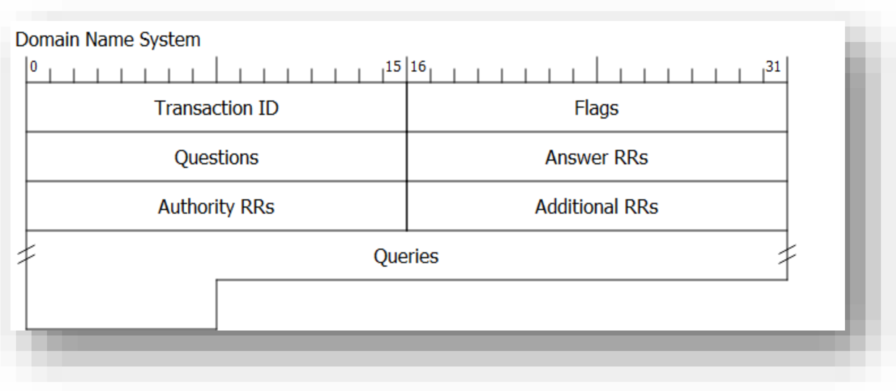
Protocols used by ***Netflix*** at different layers were:

1) Application Layer:

1. **DNS (Domain Name System):**

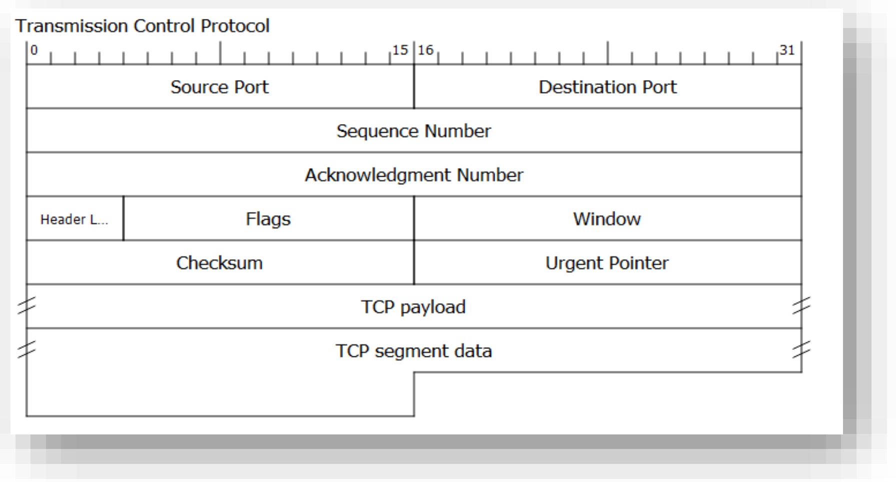
   DNS, or the Domain Name System, translates human readable domain names. DNS used the User Datagram Protocol (UDP) on port 53 to serve DNS queries. UDP is preferred because it is fast and has low overhead. A DNS query is a single UDP request from the DNS client followed by single UDP reply from the server. DNS has two types of messages: query and response. Both types have the same format. Identification field is made up of 16 bits which are used to match response with request from client side.
   Flags contain sub-fields such as: **QR(Query/Response)** which contains 0/1 depending on whether it is a query or a response. **Opcode** specifies the type of query the message is carrying. **Truncation** is set to 1 when message is truncated due to its length longer than the limit of transport medium used (512 bytes). **Number of Questions** is a 16-bit field which

specifies the count of number of questions in Question Section. **Number of answers RR** is a 16-bit field which specifies the count of number of answer records in Answer Section of message. **Queries** contains the domain name and type of record (A, AAAA, MX, TXT, etc.) being resolved.



2. TLSv1.3 (Transport Layer Security):

Transport Layer Security, or TLS, is a widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet. Content Type specifies whether the content is Handshake, Application Data, Alert, Change Cipher Spec etc. **Version** field specifies the version of TLS being used. Here it is version 1.2. **Length** gives the length of packet including header. **Payload** contains the data of the packet and **MAC** is the Message authentication code.



2) Transport Layer:

1. Transmission Control Protocol (TCP):

How does TCP work? TCP is used for organizing data in a way that ensures the secure transmission between the server and client.

Source Port Address: A 16-bit field that holds the port address of the application that is sending the data segment.

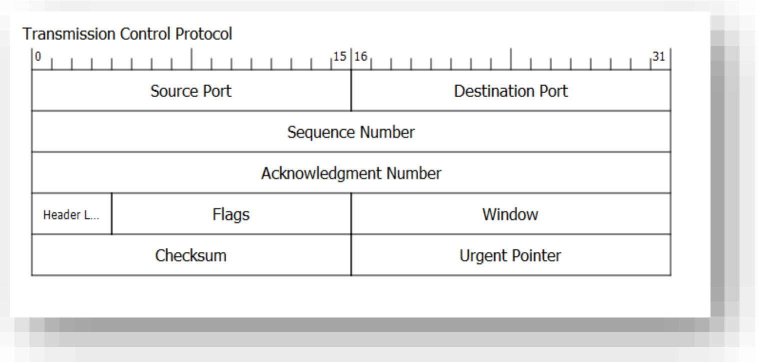
Destination Port Address: A 16-bit field that holds the port address of the application in the host that is receiving the data segment.

Sequence Number: A 32-bit field that holds the sequence number, the byte number of the first byte that is sent in that particular segment.

Acknowledgement Number: A 32-bit field that holds the acknowledgement number, the byte number that the receiver expects to receive next.

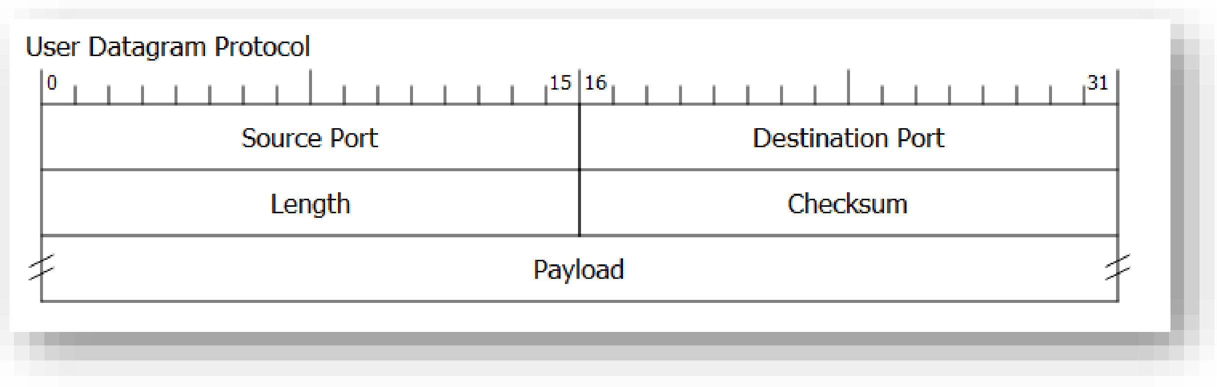
Control flags: These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc.

Urgent pointer: This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest.



2. <u>**User Datagram Protocol (UDP):**</u>

User Datagram Protocol (UDP) is a communications protocol for time-sensitive applications like gaming, playing videos, or Domain Name System (DNS) lookups.
Source Port: Source Port is a 2 Byte long field used to identify the port number of the source.
Destination Port: It is a 2 Byte long field, used to identify the port of the destined packet.
Length: Length is the length of UDP including the header and the data. It is a 16-bits field.
Checksum: Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, the pseudo-header of information from the IP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.



<u>3) Network Layer:</u>

1. <u>**Internet Protocol Version 4:**</u>

Internet Protocol version 4 (IPv4) is the fourth version of the standard that routes Internet traffic and other packet-switched networks.

VERSION: Version of the IP protocol (4 bits), which is 4 for IPv4

Type of service: Low Delay, High Throughput, Reliability (8 bits)

Total Length: Length of header + Data (16 bits), which has a minimum value 20 bytes and the maximum is 65,535 bytes.

Identification: Unique Packet Id for identifying the group of fragments of a single IP datagram (16 bits).

Flags: 3 flags of 1 bit each: reserved bit (must be zero), do not fragment flag, more fragments flag (same order).

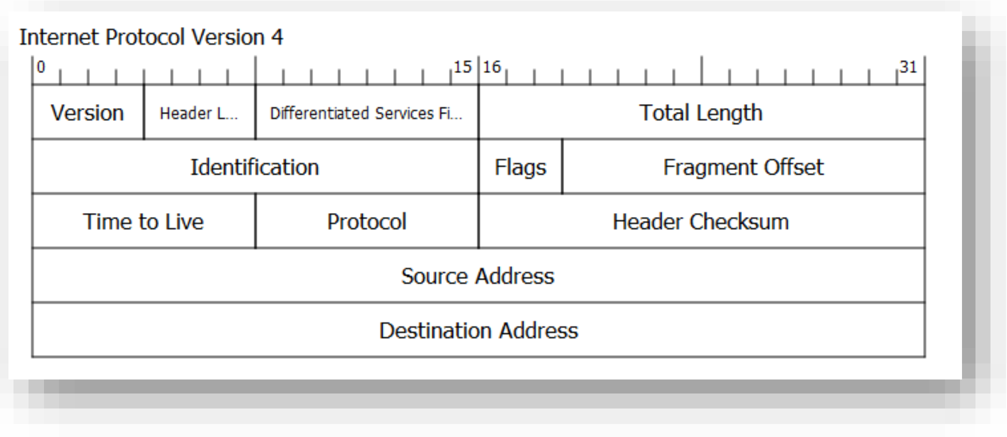
Fragment Offset: Represents the number of Data Bytes ahead of the particular fragment in the particular Datagram. Specified in terms of number of 8 bytes, which has the maximum value of 65,528 bytes.

Time to live: Datagram's lifetime (8 bits), It prevents the datagram to loop through the network by restricting the number of Hops taken by a Packet before delivering to the Destination.
Protocol: Name of the protocol to which the data is to be passed (8 bits).

Header Checksum: 16 bits header checksum for checking errors in the datagram header.

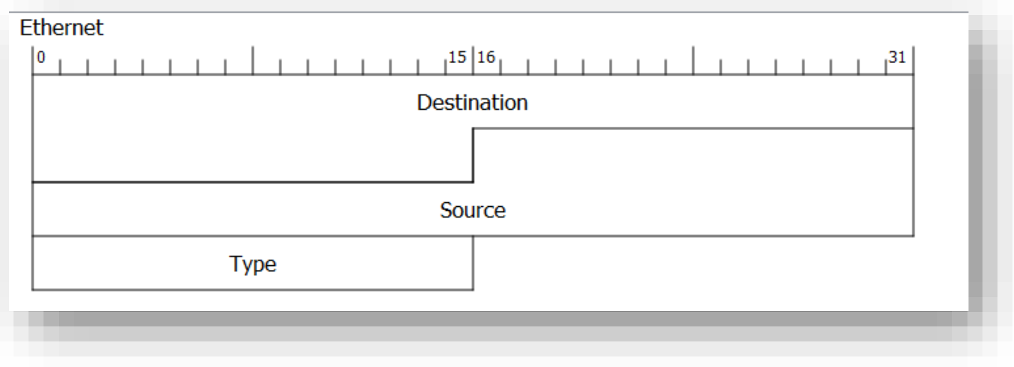
Source IP address: 32 bits IP address of the sender.

Destination IP address: 32 bits IP address of the receiver.



4) Link Layer:

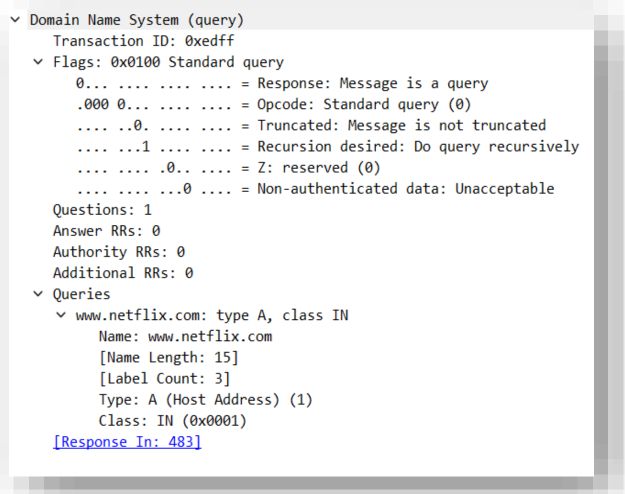
1. **Ethernet II:**
   Destination Address: This is a 6-Byte field that contains the MAC address (Media Access Control address) of the machine for which data is destined.
   Source address: This is a 6-Byte field that contains the MAC address (Media Access Control address) of the source machine.
   Type: Used to denote upper layer protocol that encapsulates the frame.



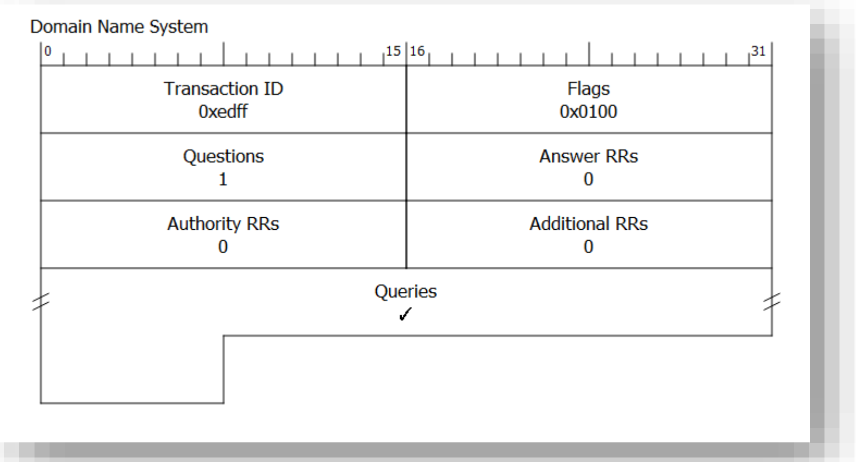
Task 2:

1) Application Layer:

1. DNS (Domain Name System):



```
v Domain Name System (query)
    Transaction ID: 0xedff
  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
    v www.netflix.com: type A, class IN
        Name: www.netflix.com
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 483]
```

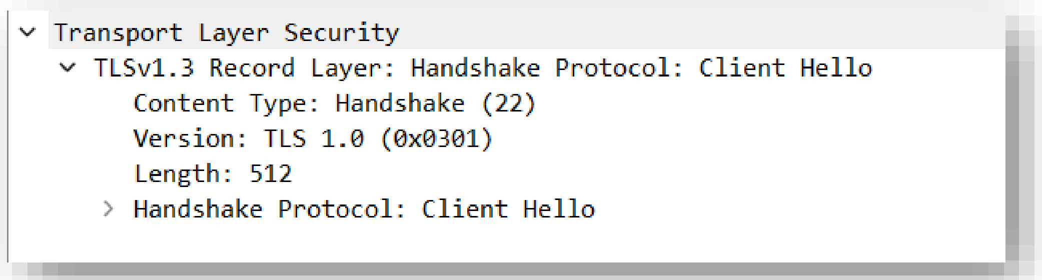


Domain Name System

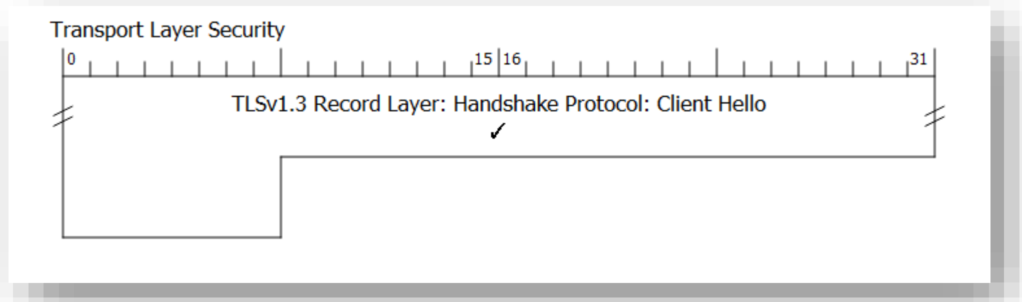
Transaction ID 0xedff	Flags 0x0100
Questions 1	Answer RRs 0
Authority RRs 0	Additional RRs 0
Queries ✓	

Transaction ID of the DNS request is 0xedff (16-bit). **Response Flag** tells us that this packet is a query and Opcode Flag specifies that the query type is standard. **Questions** in the pictures gives the number of queries which is 1. **Answer RR** is 0, **Authority RR** is 0 and **Additional RR** is also 0(since it is a query but not a response).

2. TLSv1.3 (Transport Layer Security):



```
v Transport Layer Security
  v TLSv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    > Handshake Protocol: Client Hello
```



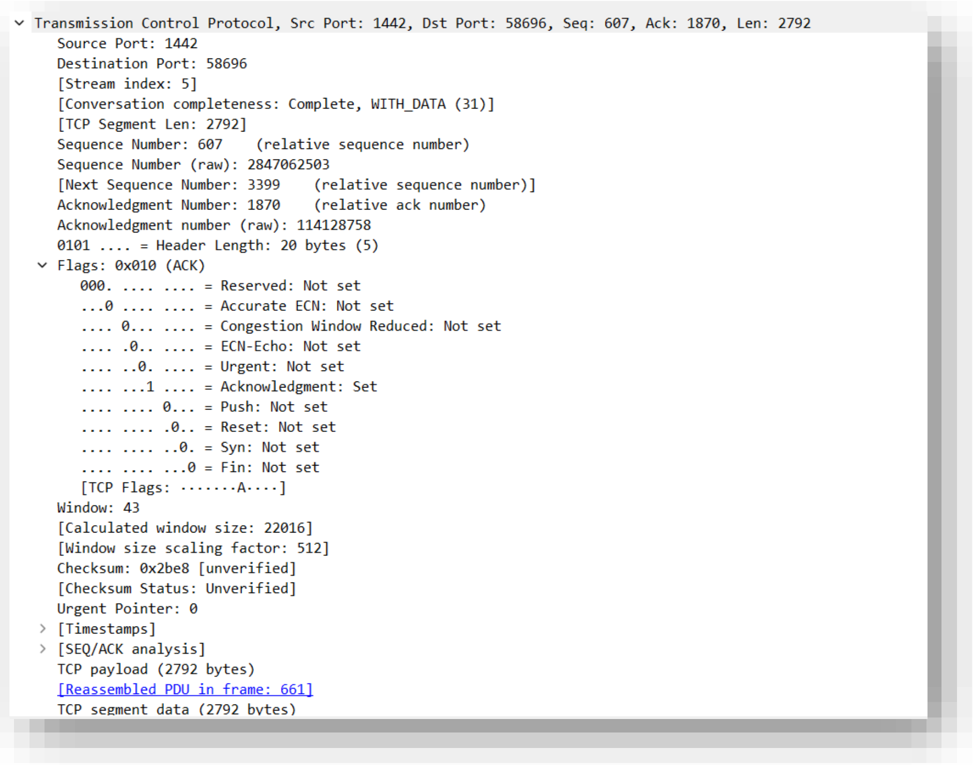
Transport Layer Security

TLSv1.3 Record Layer: Handshake Protocol: Client Hello
✓

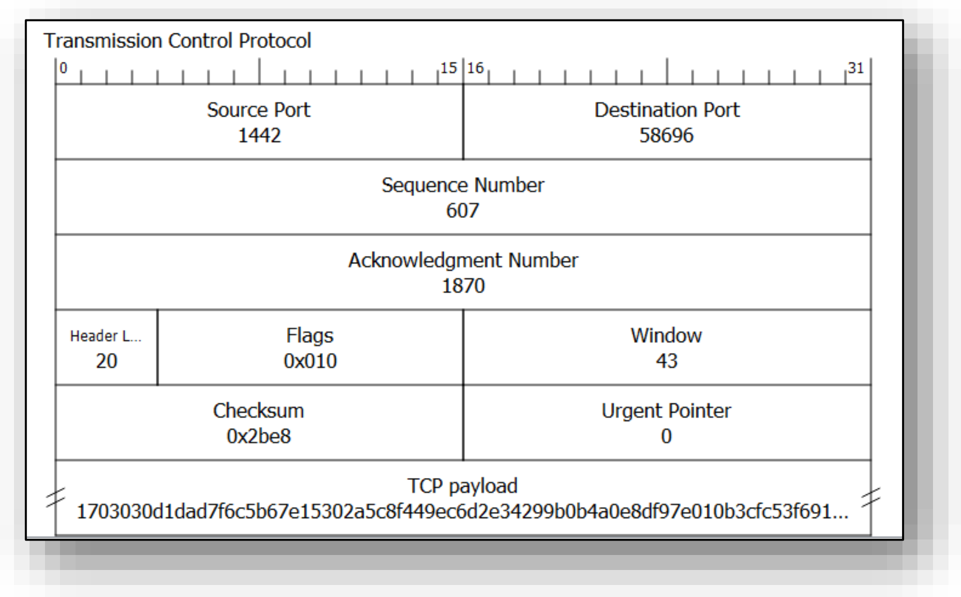
We can observe that the TLS Version being used is TLS 1.0 and content type is handshake. The TLS handshake protocol is being used to authenticate the participants of the communication and negotiate an encryption algorithm. The length of the packet including header is 512.

2) Transport Layer:

1. Transmission Control Protocol (TCP):



```
⌄ Transmission Control Protocol, Src Port: 1442, Dst Port: 58696, Seq: 607, Ack: 1870, Len: 2792
    Source Port: 1442
    Destination Port: 58696
    [Stream index: 5]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 2792]
    Sequence Number: 607    (relative sequence number)
    Sequence Number (raw): 2847062503
    [Next Sequence Number: 3399    (relative sequence number)]
    Acknowledgment Number: 1870    (relative ack number)
    Acknowledgment number (raw): 114128758
    0101 .... = Header Length: 20 bytes (5)
  ⌄ Flags: 0x010 (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
       .... .... 0... = Push: Not set
       .... .... .0.. = Reset: Not set
       .... .... ..0. = Syn: Not set
       .... .... ...0 = Fin: Not set
       [TCP Flags: ·······A····]
    Window: 43
    [Calculated window size: 22016]
    [Window size scaling factor: 512]
    Checksum: 0x2be8 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  › [Timestamps]
  › [SEQ/ACK analysis]
    TCP payload (2792 bytes)
    [Reassembled PDU in frame: 661]
    TCP segment data (2792 bytes)
```



Transmission Control Protocol

Source Port
1442

Destination Port
58696

Sequence Number
607

Acknowledgment Number
1870

Header L...
20

Flags
0x010

Window
43

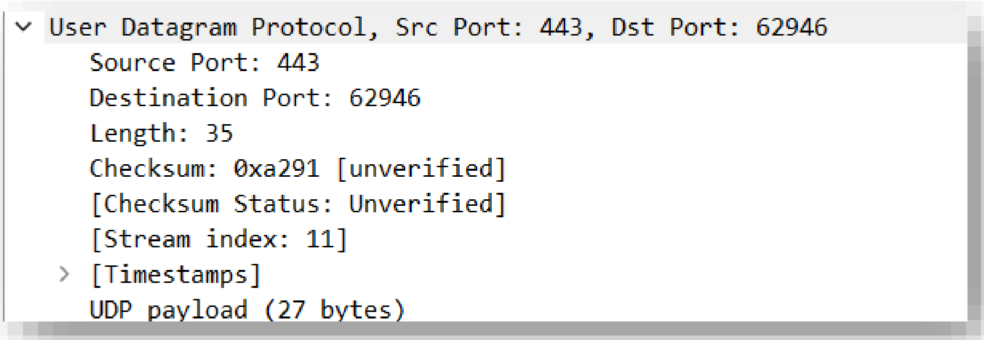
Checksum
0x2be8

Urgent Pointer
0

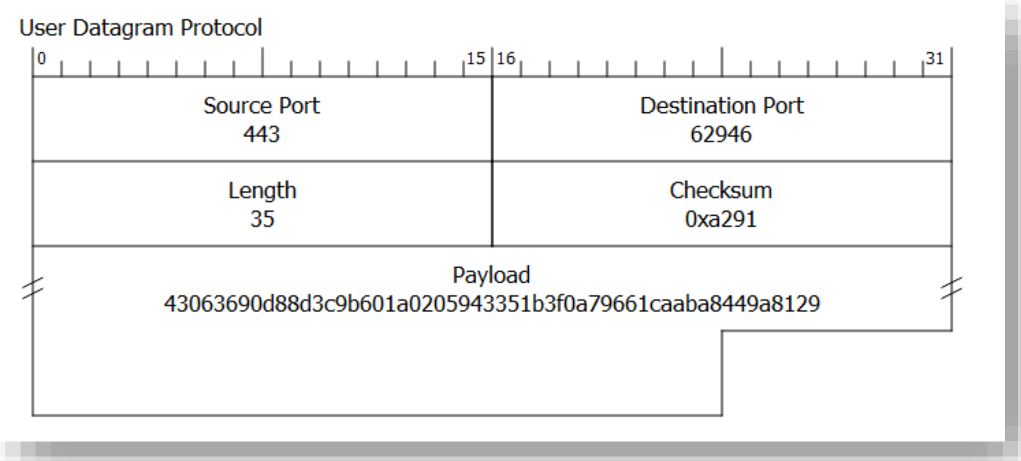
TCP payload
1703030d1dad7f6c5b67e15302a5c8f449ec6d2e34299b0b4a0e8df97e010b3cfc53f691...

The **Source port** and **Destination port** in this case is 1442 and 58696, respectively (of devices between which the communication is taking place). **Sequence Number** is 607(The sequence number is a counter used to keep track of every byte sent outward by a host) and **Acknowledgement Number** is 1870. Only **Acknowledgement Flag** is set which means that the machine sending the packet is acknowledging data that is received from another end. **Header length** is 20 bytes. **Window** is 43 and **Urgent Pointer** is 0 which means that no further bytes are urgent.

2. User Datagram Protocol (UDP):



```
v User Datagram Protocol, Src Port: 443, Dst Port: 62946
    Source Port: 443
    Destination Port: 62946
    Length: 35
    Checksum: 0xa291 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 11]
  > [Timestamps]
    UDP payload (27 bytes)
```



User Datagram Protocol

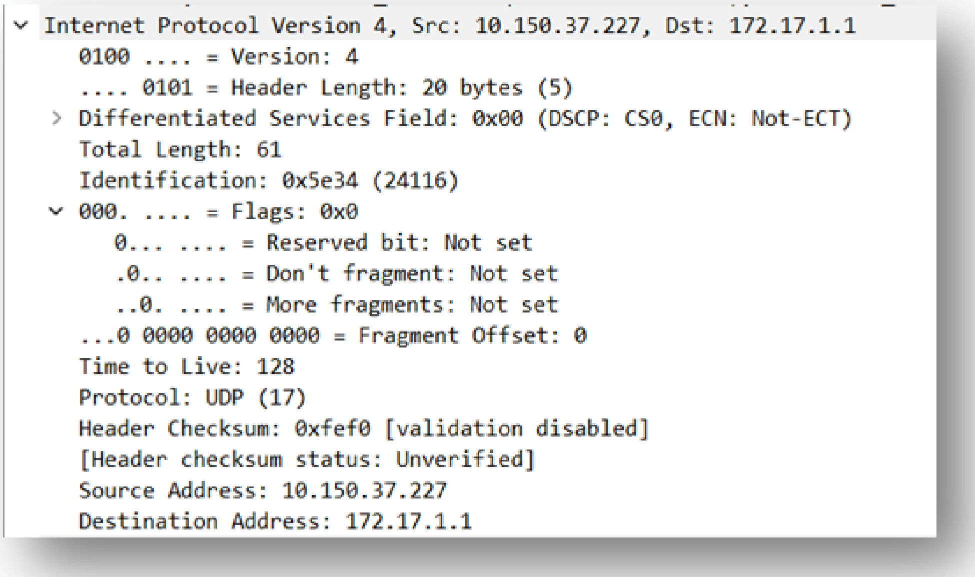
Source Port 443	Destination Port 62946
Length 35	Checksum 0xa291

Payload
43063690d88d3c9b601a0205943351b3f0a79661caaba8449a8129

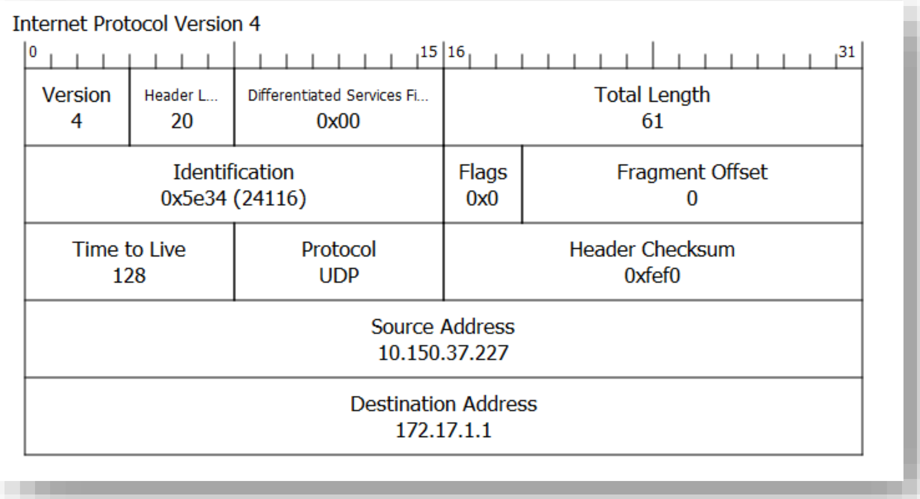
The **Source Port** and the **Destination Port** are 443 and 62946, respectively. The **Length** of the packet is 35 and the **Checksum Status** is unverified. **Stream index** displays unique number for each stream, in this case it is 11, and is used to uniquely identify a TCP stream.

3) Network Layer:

1. Internet Protocol Version 4:



```
v Internet Protocol Version 4, Src: 10.150.37.227, Dst: 172.17.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 61
    Identification: 0x5e34 (24116)
  v 000. .... = Flags: 0x0
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0xfef0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.150.37.227
    Destination Address: 172.17.1.1
```



Internet Protocol Version 4

Version 4	Header L... 20	Differentiated Services Fi... 0x00	Total Length 61

Identification 0x5e34 (24116)	Flags 0x0	Fragment Offset 0

Time to Live 128	Protocol UDP	Header Checksum 0xfef0

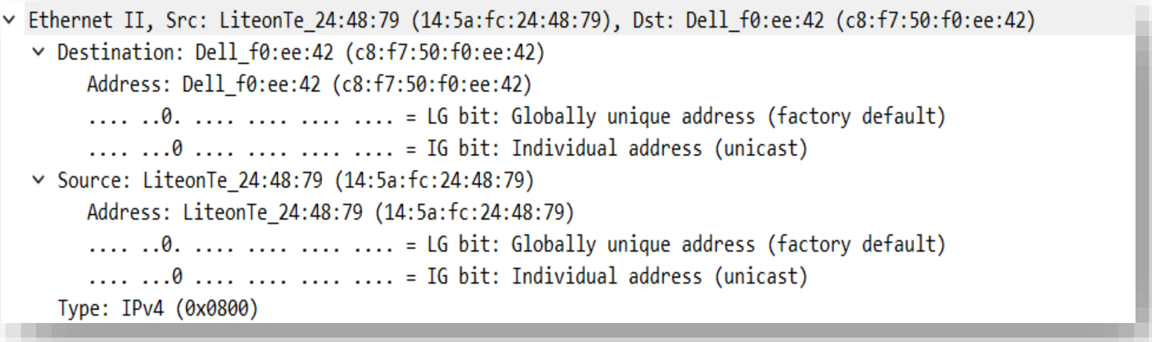
Source Address
10.150.37.227

Destination Address
172.17.1.1

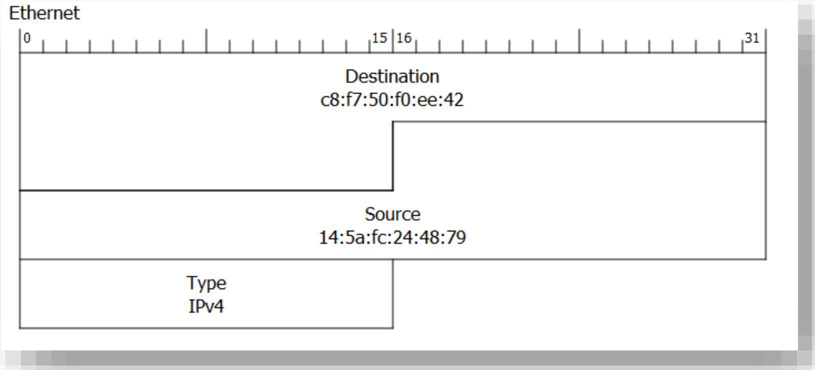
Version of IP protocol used is 4 i.e., IPv4. The **Header Length** is 20 bytes and the **Total Length** of packet is 61. The **Do not Fragment** bit is not set which indicates that packet can be fragmented. **More Fragments** bit is not set, so no fragments of same packet will arrive later. **Fragment Offset** is 0, so this packet is not fragmented (because of the previous 2 flags). **Time to Live** is 128 which means that packet can make at most 128 hops. The next level protocol to be used is UDP and source and destination addresses are 10.150.37.227 and 172.17.1.1 respectively.

4) Link Layer:

1. Ethernet II:



```
v Ethernet II, Src: LiteonTe_24:48:79 (14:5a:fc:24:48:79), Dst: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
  v Destination: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
      Address: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: LiteonTe_24:48:79 (14:5a:fc:24:48:79)
      Address: LiteonTe_24:48:79 (14:5a:fc:24:48:79)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```



Ethernet

Destination
c8:f7:50:f0:ee:42

Source
14:5a:fc:24:48:79

Type
IPv4

The **Source** and **Destination Values** contain the source and destination device's MAC addresses which are 14:5a:fc: 24:48:79 and c8: f7:50: f0:ee:42 respectively. **Type** indicates the upper layer protocol used which is IPv4 in this case.

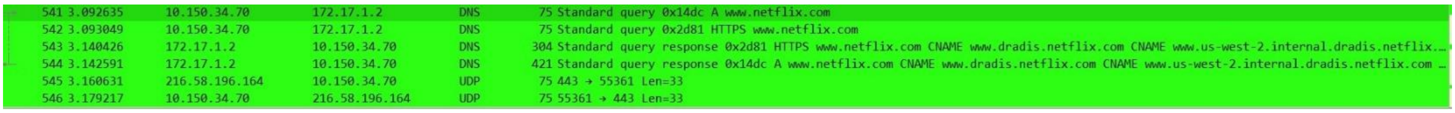
Task 3:

The functionalities of the application that we observed are the following:

1. **Starting the application**

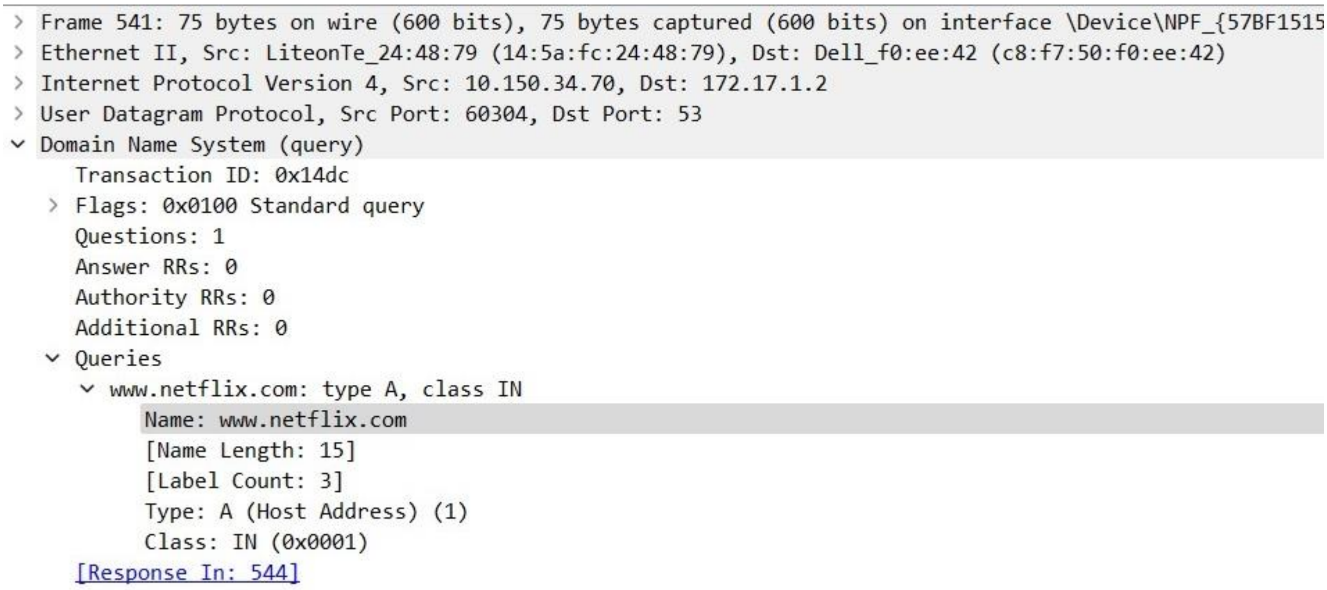
    ○ **DNS Query**

    The client (Our PC) first sends a request to the DNS server for Netflix's IP address and then it receives a response containing the IP address.



```
541 3.092635  10.150.34.70   172.17.1.2     DNS   75 Standard query 0x14dc A www.netflix.com
542 3.093049  10.150.34.70   172.17.1.2     DNS   75 Standard query 0x2d81 HTTPS www.netflix.com
543 3.140426  172.17.1.2     10.150.34.70   DNS  304 Standard query response 0x2d81 HTTPS www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.…
544 3.142591  172.17.1.2     10.150.34.70   DNS  421 Standard query response 0x14dc A www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.com …
545 3.160631  216.58.196.164 10.150.34.70   UDP   75 443 → 55361 Len=33
546 3.179217  10.150.34.70   216.58.196.164 UDP   75 55361 → 443 Len=33
```

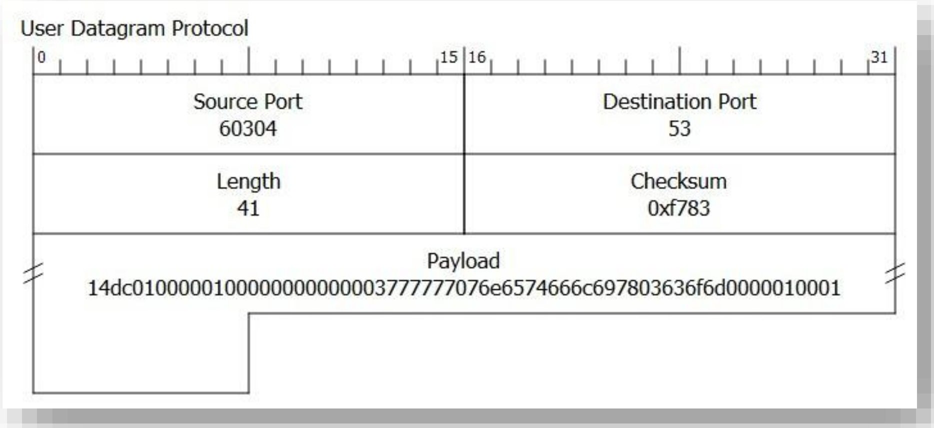
The protocol followed is UDP. The packet diagram is given below.



```
> Frame 541: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface \Device\NPF_{57BF1515
> Ethernet II, Src: LiteonTe_24:48:79 (14:5a:fc:24:48:79), Dst: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
> Internet Protocol Version 4, Src: 10.150.34.70, Dst: 172.17.1.2
> User Datagram Protocol, Src Port: 60304, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0x14dc
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    v www.netflix.com: type A, class IN
        Name: www.netflix.com
        [Name Length: 15]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 544]
```

The source port is 60304 and destination port is 53. A DNS Type A query is sent to the DNS server for getting www.netflix.com's IP address.

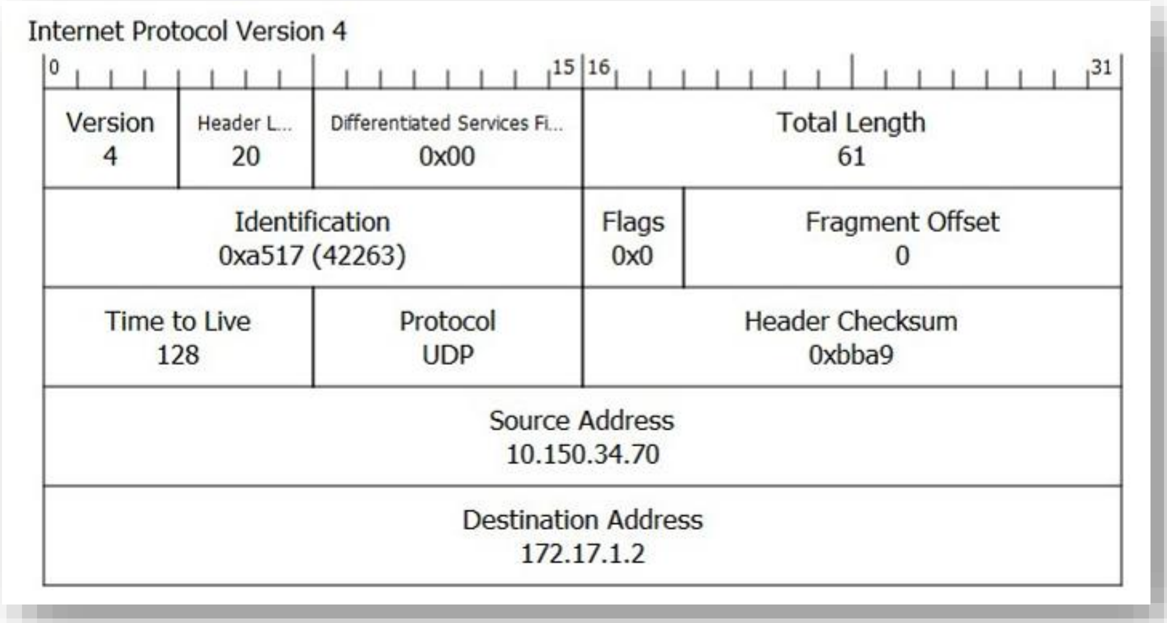
The source address (PC) is 10.150.34.70 and destination address is 172.17.1.2, which is the address of IITG DNS server.



User Datagram Protocol

Source Port 60304	Destination Port 53
Length 41	Checksum 0xf783

Payload
14dc0100000100000000000003777777076e6574666c697803636f6d0000010001



Internet Protocol Version 4

Version 4	Header L... 20	Differentiated Services Fi... 0x00	Total Length 61	
Identification 0xa517 (42263)			Flags 0x0	Fragment Offset 0
Time to Live 128		Protocol UDP	Header Checksum 0xbba9	
Source Address 10.150.34.70				
Destination Address 172.17.1.2				

- **Handshaking**

The handshaking process usually takes place in order to establish rules for communication when a computer attempts to communicate with another device. Signals are usually exchanged between two devices to establish a communication link. For example, when a computer communicates with another device such as a modem, the two devices will signal each other that they are switched on and ready to work, as well as to agree to which protocols are being used. Here, two handshaking processes have occurred. One, the **TCP three-way handshake**, and the other being **TLS handshake.**

- **TCP three-way handshake**

Establishing a normal TCP connection requires three separate steps:

❖ The first host (Alice) sends the second host (Bob) a "synchronize" (SYN) message with its own sequence number , which Bob receives.

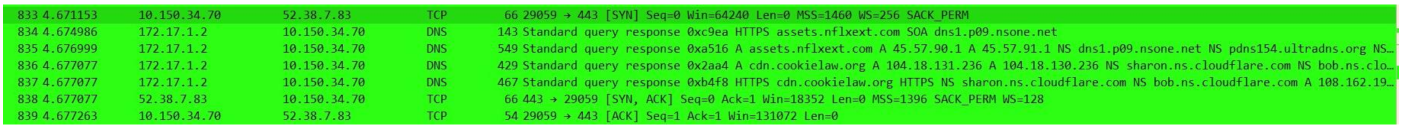
❖ Bob replies with a synchronize-acknowledgment (SYN-ACK) message with its own sequence number and acknowledgement number , which Alice receives.

❖ Alice replies with an acknowledgment (ACK) message with acknowledgement number , which Bob receives and to which he does not need to reply.

In this setup, the synchronize messages act as service requests from one server to the other, while the acknowledgement messages return to the requesting server to let it know the message was received.

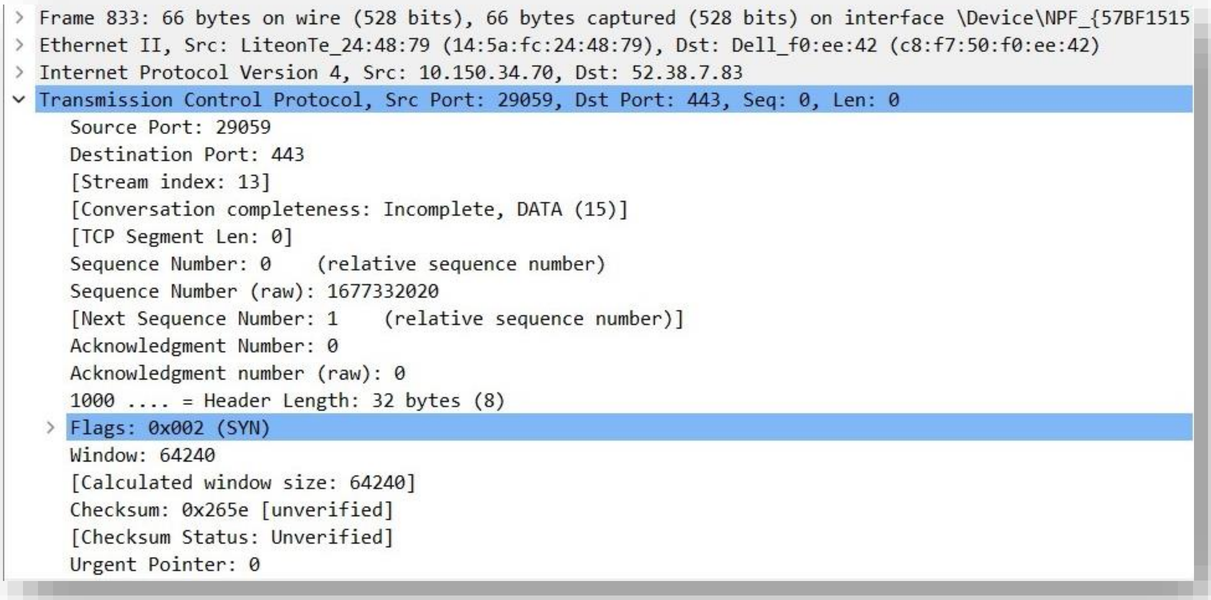
The reason for the client and server not using a default sequence number such as 0 for establishing the connection is to protect against two incarnations of the same connection reusing the same sequence number too soon, which means a segment from an earlier incarnation of a connection might interfere with a later incarnation of the connection.

The TCP three-way handshake happens when we start the application.



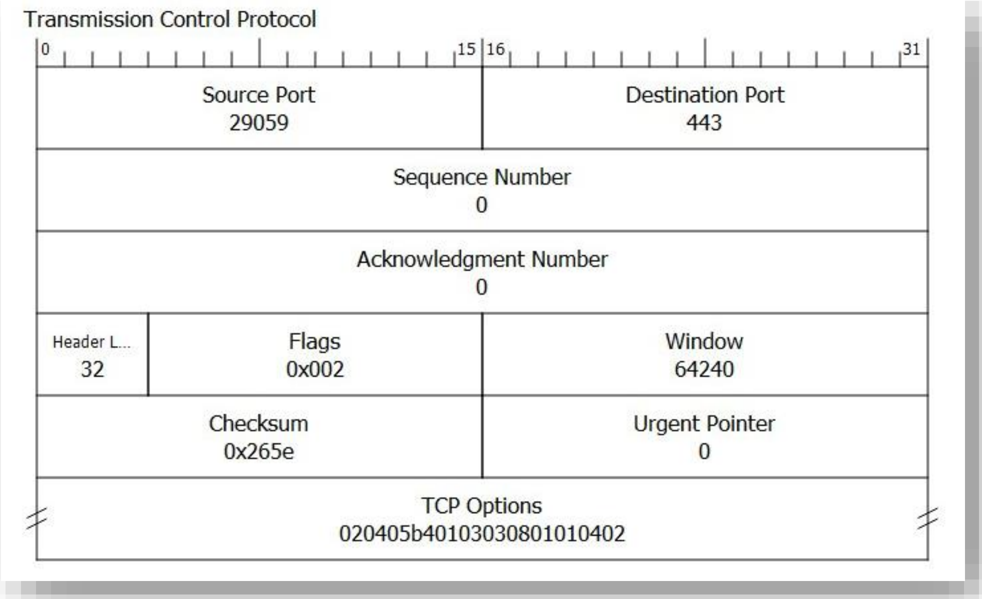
```
833 4.671153   10.150.34.70   52.38.7.83     TCP    66 29059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
834 4.674986   172.17.1.2     10.150.34.70   DNS   143 Standard query response 0xc9ea HTTPS assets.nflxext.com SOA dns1.p09.nsone.net
835 4.676999   172.17.1.2     10.150.34.70   DNS   549 Standard query response 0xa516 A assets.nflxext.com A 45.57.90.1 A 45.57.91.1 NS dns1.p09.nsone.net NS pdns154.ultradns.org NS…
836 4.677077   172.17.1.2     10.150.34.70   DNS   429 Standard query response 0x2aa4 A cdn.cookielaw.org A 104.18.131.236 A 104.18.130.236 NS sharon.ns.cloudflare.com NS bob.ns.clo…
837 4.677077   172.17.1.2     10.150.34.70   DNS   467 Standard query response 0xb4f8 HTTPS cdn.cookielaw.org HTTPS NS sharon.ns.cloudflare.com NS bob.ns.cloudflare.com A 108.162.19…
838 4.677077   52.38.7.83     10.150.34.70   TCP    66 443 → 29059 [SYN, ACK] Seq=0 Ack=1 Win=18352 Len=0 MSS=1396 SACK_PERM WS=128
839 4.677263   10.150.34.70   52.38.7.83     TCP    54 29059 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
```

The client (PC) establishes a connection with the Netflix server (IP Address - 52.38.7.83), so it sends a segment with SYN (Synchronize Sequence Number) which informs the server that the client is likely to start communication and the sequence number. Server responds to the client request with SYN-ACK signal bits set. The ACK signifies the response of the segment it received and SYN signifies with what sequence number it is likely to start the segments with. The client acknowledges the response of the server and thus establish a reliable connection with which they start the actual data transfer.



```
> Frame 833: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{57BF1515
> Ethernet II, Src: LiteonTe_24:48:79 (14:5a:fc:24:48:79), Dst: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
> Internet Protocol Version 4, Src: 10.150.34.70, Dst: 52.38.7.83
v Transmission Control Protocol, Src Port: 29059, Dst Port: 443, Seq: 0, Len: 0
     Source Port: 29059
     Destination Port: 443
     [Stream index: 13]
     [Conversation completeness: Incomplete, DATA (15)]
     [TCP Segment Len: 0]
     Sequence Number: 0    (relative sequence number)
     Sequence Number (raw): 1677332020
     [Next Sequence Number: 1    (relative sequence number)]
     Acknowledgment Number: 0
     Acknowledgment number (raw): 0
     1000 .... = Header Length: 32 bytes (8)
   > Flags: 0x002 (SYN)
     Window: 64240
     [Calculated window size: 64240]
     Checksum: 0x265e [unverified]
     [Checksum Status: Unverified]
     Urgent Pointer: 0
```

The source port (PC) is 29059 and destination port (Netflix server) is 443.



Transmission Control Protocol

Source Port
29059

Destination Port
443

Sequence Number
0

Acknowledgment Number
0

Header L...
32

Flags
0x002

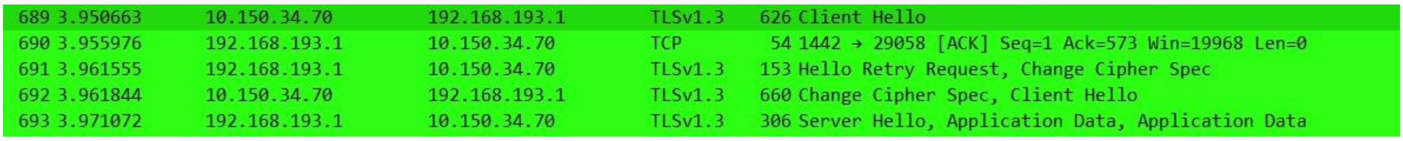
Window
64240

Checksum
0x265e

Urgent Pointer
0

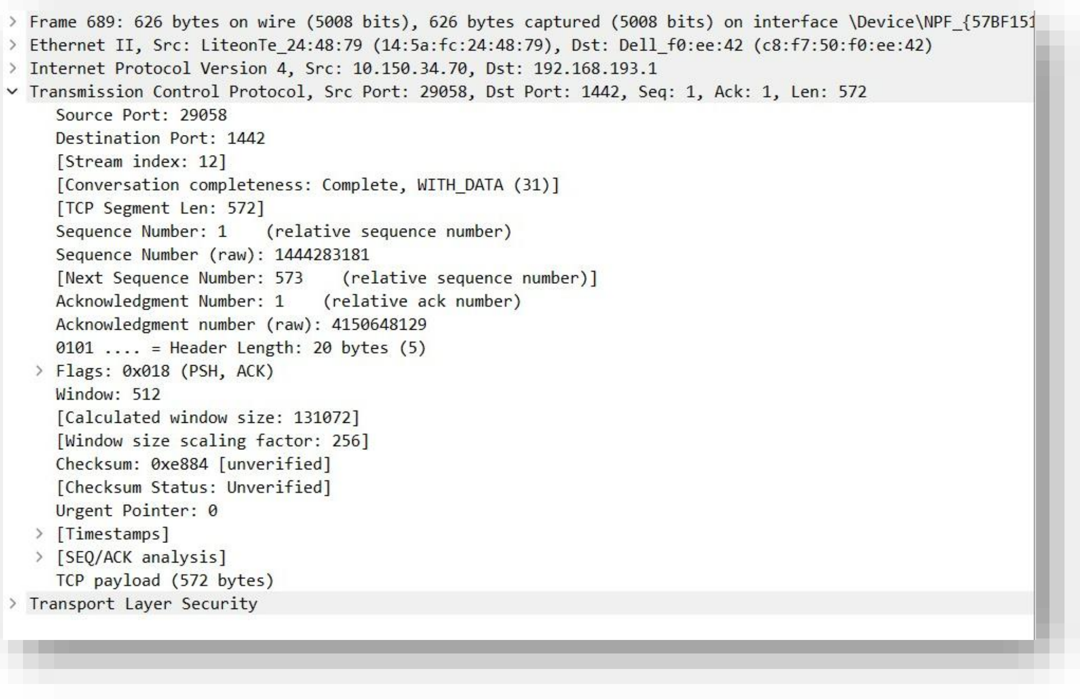
TCP Options
020405b40103030801010402

- ○ **TLS Handshake**
  TLS is an encryption and authentication protocol designed to secure Internet communications. A TLS handshake is the process that kicks off a communication session that uses TLS. During a TLS handshake, the two communicating sides exchange messages to acknowledge each other, verify each other, establish the cryptographic algorithms they will use, and agree on session keys. TLS handshakes are a foundational part of how HTTPS works.

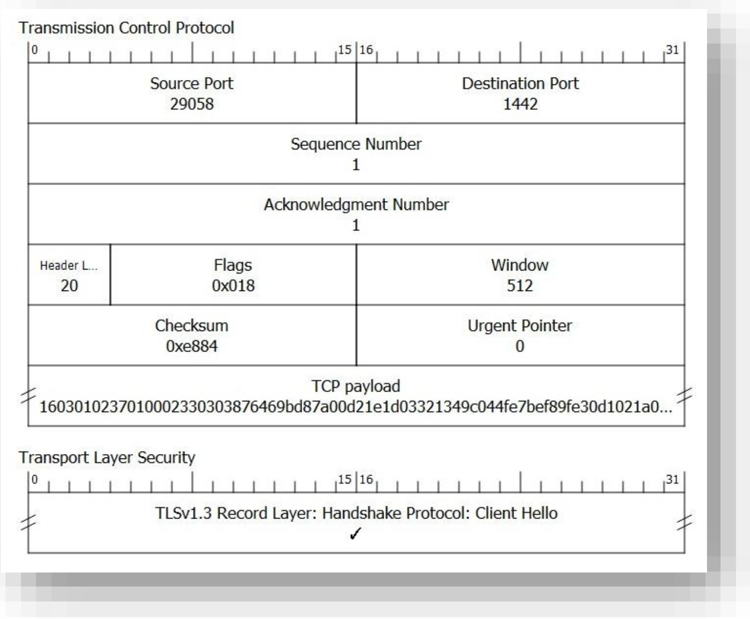


```
689 3.950663    10.150.34.70     192.168.193.1    TLSv1.3   626 Client Hello
690 3.955976    192.168.193.1    10.150.34.70     TCP        54 1442 → 29058 [ACK] Seq=1 Ack=573 Win=19968 Len=0
691 3.961555    192.168.193.1    10.150.34.70     TLSv1.3   153 Hello Retry Request, Change Cipher Spec
692 3.961844    10.150.34.70     192.168.193.1    TLSv1.3   660 Change Cipher Spec, Client Hello
693 3.971072    192.168.193.1    10.150.34.70     TLSv1.3   306 Server Hello, Application Data, Application Data
```

The client sends a “Client Hello” message, to which the server responds with “Server Hello” and data transfer takes place.



```
> Frame 689: 626 bytes on wire (5008 bits), 626 bytes captured (5008 bits) on interface \Device\NPF_{57BF151
> Ethernet II, Src: LiteonTe_24:48:79 (14:5a:fc:24:48:79), Dst: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
> Internet Protocol Version 4, Src: 10.150.34.70, Dst: 192.168.193.1
v Transmission Control Protocol, Src Port: 29058, Dst Port: 1442, Seq: 1, Ack: 1, Len: 572
     Source Port: 29058
     Destination Port: 1442
     [Stream index: 12]
     [Conversation completeness: Complete, WITH_DATA (31)]
     [TCP Segment Len: 572]
     Sequence Number: 1    (relative sequence number)
     Sequence Number (raw): 1444283181
     [Next Sequence Number: 573    (relative sequence number)]
     Acknowledgment Number: 1    (relative ack number)
     Acknowledgment number (raw): 4150648129
     0101 .... = Header Length: 20 bytes (5)
   > Flags: 0x018 (PSH, ACK)
     Window: 512
     [Calculated window size: 131072]
     [Window size scaling factor: 256]
     Checksum: 0xe884 [unverified]
     [Checksum Status: Unverified]
     Urgent Pointer: 0
   > [Timestamps]
   > [SEQ/ACK analysis]
     TCP payload (572 bytes)
> Transport Layer Security
```



Transmission Control Protocol

Source Port 29058 | Destination Port 1442

Sequence Number 1

Acknowledgment Number 1

Header L... 20 | Flags 0x018 | Window 512

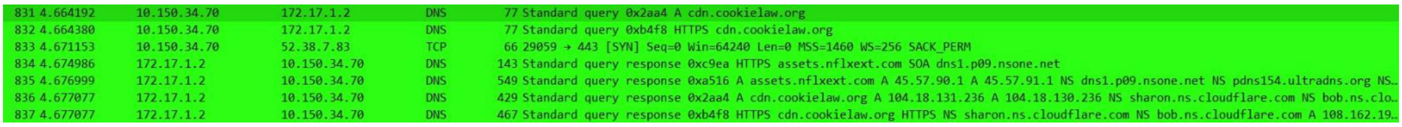
Checksum 0xe884 | Urgent Pointer 0

TCP payload
1603010237010002330303876469bd87a00d21e1d03321349c044fe7bef89fe30d1021a0...

Transport Layer Security

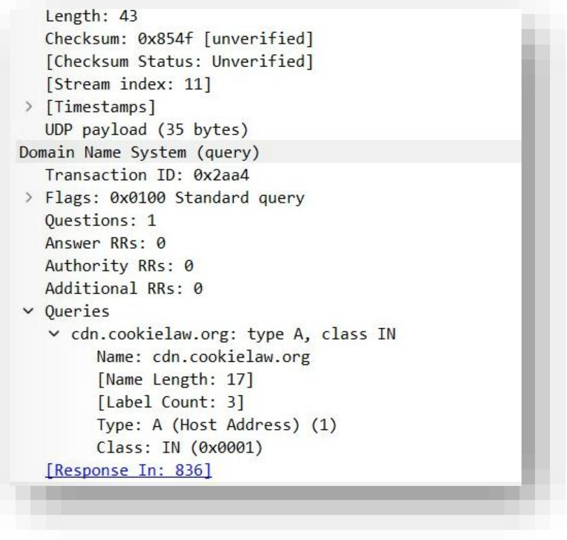
TLSv1.3 Record Layer: Handshake Protocol: Client Hello
✓

- ### Cookie Query



```
831 4.664192   10.150.34.70   172.17.1.2     DNS    77 Standard query 0x2aa4 A cdn.cookielaw.org
832 4.664380   10.150.34.70   172.17.1.2     DNS    77 Standard query 0xb4f8 HTTPS cdn.cookielaw.org
833 4.671153   10.150.34.70   52.38.7.83     TCP    66 29059 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
834 4.674986   172.17.1.2     10.150.34.70   DNS   143 Standard query response 0xc9ea HTTPS assets.nflxext.com SOA dns1.p09.nsone.net
835 4.676999   172.17.1.2     10.150.34.70   DNS   549 Standard query response 0xa516 A assets.nflxext.com A 45.57.90.1 A 45.57.91.1 NS dns1.p09.nsone.net NS pdns154.ultradns.org NS…
836 4.677077   172.17.1.2     10.150.34.70   DNS   429 Standard query response 0x2aa4 A cdn.cookielaw.org A 104.18.131.236 A 104.18.130.236 NS sharon.ns.cloudflare.com NS bob.ns.clo…
837 4.677077   172.17.1.2     10.150.34.70   DNS   467 Standard query response 0xb4f8 HTTPS cdn.cookielaw.org HTTPS NS sharon.ns.cloudflare.com NS bob.ns.cloudflare.com A 108.162.19…
```

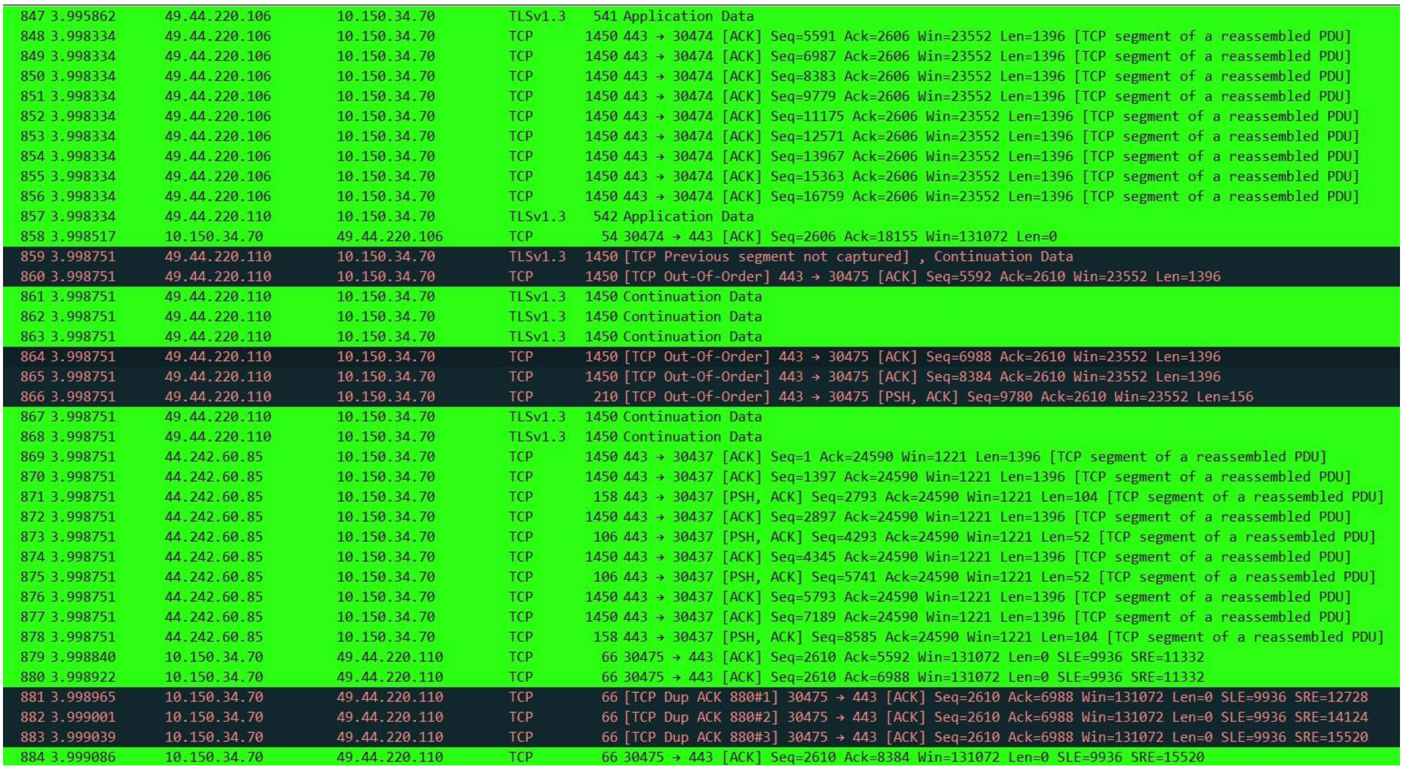
Cookies are small pieces of text sent to the browser by a website we visit. They help that website remember information about our visit, which can both make it easier to visit the site again and make the site more useful to us. Here, the client and the server exchange information about the client’s cookie.



```
   Length: 43
   Checksum: 0x854f [unverified]
   [Checksum Status: Unverified]
   [Stream index: 11]
 > [Timestamps]
   UDP payload (35 bytes)
Domain Name System (query)
   Transaction ID: 0x2aa4
 > Flags: 0x0100 Standard query
   Questions: 1
   Answer RRs: 0
   Authority RRs: 0
   Additional RRs: 0
 ⌄ Queries
    ⌄ cdn.cookielaw.org: type A, class IN
         Name: cdn.cookielaw.org
         [Name Length: 17]
         [Label Count: 3]
         Type: A (Host Address) (1)
         Class: IN (0x0001)
   [Response In: 836]
```

2. Playing a video

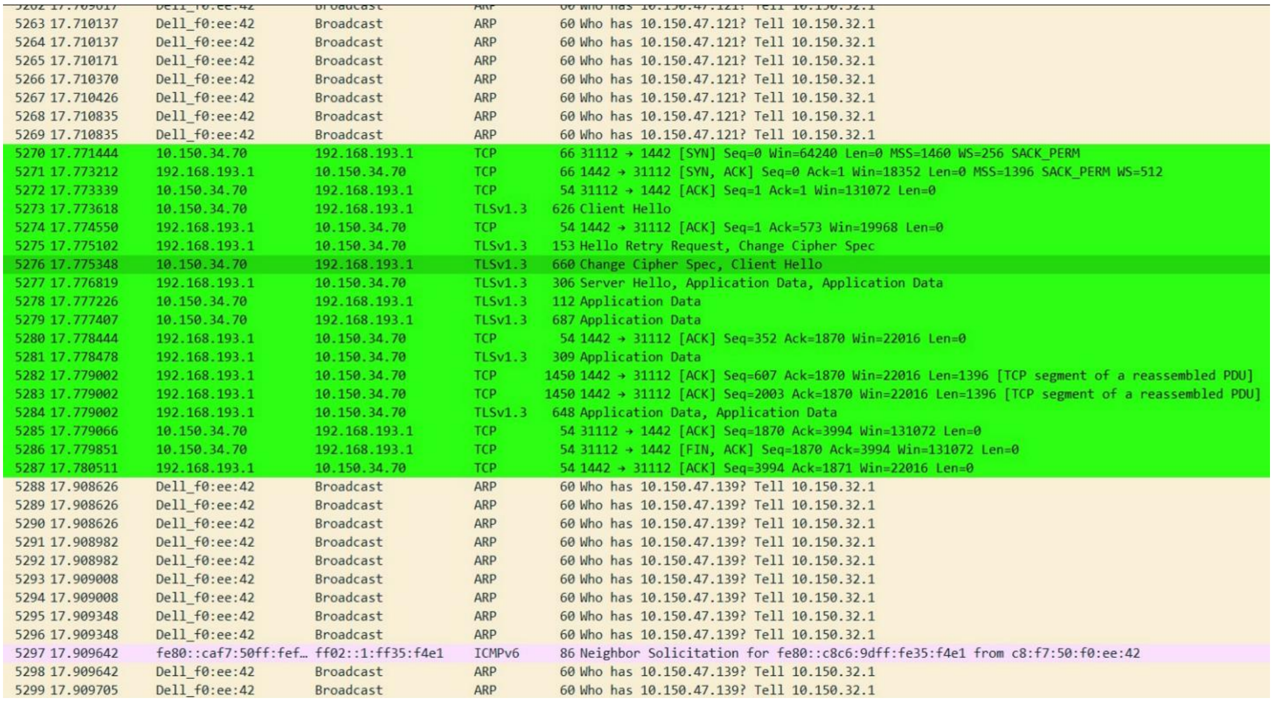
When a video is started, the server sends packets continuously to ensure smooth streaming service and avoid buffering of the video. Given below is a screenshot of the data transfer form the Netflix server to the client (PC).



```
847 3.995862   49.44.220.106   10.150.34.70    TLSv1.3   541 Application Data
848 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=5591 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
849 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=6987 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
850 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=8383 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
851 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=9779 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
852 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=11175 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
853 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=12571 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
854 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=13967 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
855 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=15363 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
856 3.998334   49.44.220.106   10.150.34.70    TCP      1450 443 → 30474 [ACK] Seq=16759 Ack=2606 Win=23552 Len=1396 [TCP segment of a reassembled PDU]
857 3.998334   49.44.220.110   10.150.34.70    TLSv1.3   542 Application Data
858 3.998517   10.150.34.70    49.44.220.106   TCP        54 30474 → 443 [ACK] Seq=2606 Ack=18155 Win=131072 Len=0
859 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 [TCP Previous segment not captured] , Continuation Data
860 3.998751   49.44.220.110   10.150.34.70    TCP      1450 [TCP Out-Of-Order] 443 → 30475 [ACK] Seq=5592 Ack=2610 Win=23552 Len=1396
861 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 Continuation Data
862 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 Continuation Data
863 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 Continuation Data
864 3.998751   49.44.220.110   10.150.34.70    TCP      1450 [TCP Out-Of-Order] 443 → 30475 [ACK] Seq=6988 Ack=2610 Win=23552 Len=1396
865 3.998751   49.44.220.110   10.150.34.70    TCP      1450 [TCP Out-Of-Order] 443 → 30475 [ACK] Seq=8384 Ack=2610 Win=23552 Len=1396
866 3.998751   49.44.220.110   10.150.34.70    TCP       210 [TCP Out-Of-Order] 443 → 30475 [PSH, ACK] Seq=9780 Ack=2610 Win=23552 Len=156
867 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 Continuation Data
868 3.998751   49.44.220.110   10.150.34.70    TLSv1.3  1450 Continuation Data
869 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=1 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
870 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=1397 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
871 3.998751   44.242.60.85    10.150.34.70    TCP       158 443 → 30437 [PSH, ACK] Seq=2793 Ack=24590 Win=1221 Len=104 [TCP segment of a reassembled PDU]
872 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=2897 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
873 3.998751   44.242.60.85    10.150.34.70    TCP       106 443 → 30437 [PSH, ACK] Seq=4293 Ack=24590 Win=1221 Len=52 [TCP segment of a reassembled PDU]
874 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=4345 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
875 3.998751   44.242.60.85    10.150.34.70    TCP       106 443 → 30437 [PSH, ACK] Seq=5741 Ack=24590 Win=1221 Len=52 [TCP segment of a reassembled PDU]
876 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=5793 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
877 3.998751   44.242.60.85    10.150.34.70    TCP      1450 443 → 30437 [ACK] Seq=7189 Ack=24590 Win=1221 Len=1396 [TCP segment of a reassembled PDU]
878 3.998751   44.242.60.85    10.150.34.70    TCP       158 443 → 30437 [PSH, ACK] Seq=8585 Ack=24590 Win=1221 Len=104 [TCP segment of a reassembled PDU]
879 3.998840   10.150.34.70    49.44.220.110   TCP        66 30475 → 443 [ACK] Seq=2610 Ack=5592 Win=131072 Len=0 SLE=9936 SRE=11332
880 3.998922   10.150.34.70    49.44.220.110   TCP        66 30475 → 443 [ACK] Seq=2610 Ack=6988 Win=131072 Len=0 SLE=9936 SRE=11332
881 3.998965   10.150.34.70    49.44.220.110   TCP        66 [TCP Dup ACK 880#1] 30475 → 443 [ACK] Seq=2610 Ack=6988 Win=131072 Len=0 SLE=9936 SRE=12728
882 3.999001   10.150.34.70    49.44.220.110   TCP        66 [TCP Dup ACK 880#2] 30475 → 443 [ACK] Seq=2610 Ack=6988 Win=131072 Len=0 SLE=9936 SRE=14124
883 3.999039   10.150.34.70    49.44.220.110   TCP        66 [TCP Dup ACK 880#3] 30475 → 443 [ACK] Seq=2610 Ack=6988 Win=131072 Len=0 SLE=9936 SRE=15520
884 3.999086   10.150.34.70    49.44.220.110   TCP        66 30475 → 443 [ACK] Seq=2610 Ack=8384 Win=131072 Len=0 SLE=9936 SRE=15520
```

3. **Pausing a video**

When the video is paused, the frequency of the packets decreases. The server sends packets at a lower frequency compared to the frequency at which packets were sent when the video was playing. Given below is the screenshot of the packets when the video was paused.

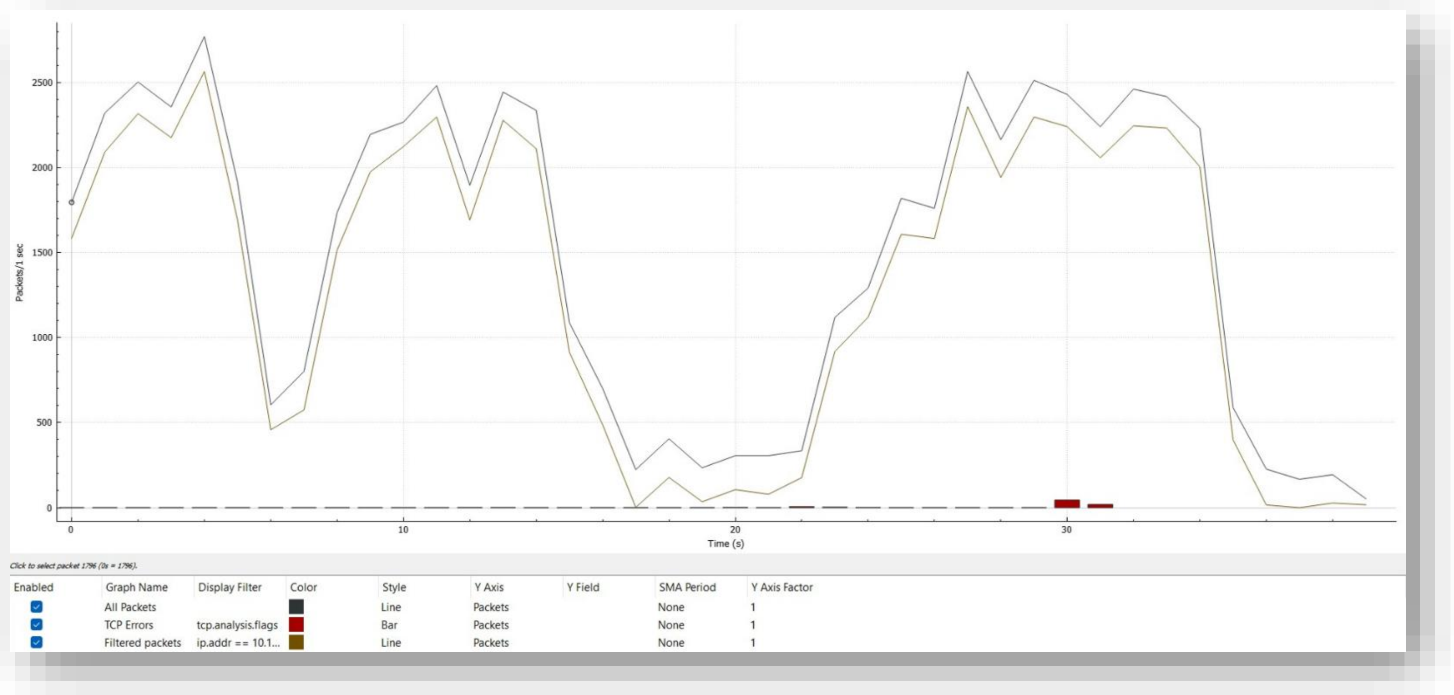


```
5263 17.710137   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5264 17.710137   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5265 17.710171   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5266 17.710370   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5267 17.710426   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5268 17.710835   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5269 17.710835   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.121? Tell 10.150.32.1
5270 17.771444   10.150.34.70       192.168.193.1      TCP        66 31112 → 1442 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5271 17.773212   192.168.193.1      10.150.34.70       TCP        66 1442 → 31112 [SYN, ACK] Seq=0 Ack=1 Win=18352 Len=0 MSS=1396 SACK_PERM WS=512
5272 17.773339   10.150.34.70       192.168.193.1      TCP        54 31112 → 1442 [ACK] Seq=1 Ack=1 Win=131072 Len=0
5273 17.773618   10.150.34.70       192.168.193.1      TLSv1.3   626 Client Hello
5274 17.774550   192.168.193.1      10.150.34.70       TCP        54 1442 → 31112 [ACK] Seq=1 Ack=573 Win=19968 Len=0
5275 17.775102   192.168.193.1      10.150.34.70       TLSv1.3   153 Hello Retry Request, Change Cipher Spec
5276 17.775348   10.150.34.70       192.168.193.1      TLSv1.3   660 Change Cipher Spec, Client Hello
5277 17.776819   192.168.193.1      10.150.34.70       TLSv1.3   306 Server Hello, Application Data, Application Data
5278 17.777226   10.150.34.70       192.168.193.1      TLSv1.3   112 Application Data
5279 17.777407   10.150.34.70       192.168.193.1      TLSv1.3   687 Application Data
5280 17.778444   192.168.193.1      10.150.34.70       TCP        54 1442 → 31112 [ACK] Seq=352 Ack=1870 Win=22016 Len=0
5281 17.778478   192.168.193.1      10.150.34.70       TLSv1.3   309 Application Data
5282 17.779002   192.168.193.1      10.150.34.70       TCP      1450 1442 → 31112 [ACK] Seq=607 Ack=1870 Win=22016 Len=1396 [TCP segment of a reassembled PDU]
5283 17.779002   192.168.193.1      10.150.34.70       TCP      1450 1442 → 31112 [ACK] Seq=2003 Ack=1870 Win=22016 Len=1396 [TCP segment of a reassembled PDU]
5284 17.779002   192.168.193.1      10.150.34.70       TLSv1.3   648 Application Data, Application Data
5285 17.779066   10.150.34.70       192.168.193.1      TCP        54 31112 → 1442 [ACK] Seq=1870 Ack=3994 Win=131072 Len=0
5286 17.779851   10.150.34.70       192.168.193.1      TCP        54 31112 → 1442 [FIN, ACK] Seq=1870 Ack=3994 Win=131072 Len=0
5287 17.780511   192.168.193.1      10.150.34.70       TCP        54 1442 → 31112 [ACK] Seq=3994 Ack=1871 Win=22016 Len=0
5288 17.908626   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5289 17.908626   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5290 17.908626   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5291 17.908982   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5292 17.908982   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5293 17.909008   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5294 17.909008   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5295 17.909348   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5296 17.909348   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5297 17.909642   fe80::caf7:50ff:fef… ff02::1:ff35:f4e1 ICMPv6   86 Neighbor Solicitation for fe80::c8c6:9dff:fe35:f4e1 from c8:f7:50:f0:ee:42
5298 17.909642   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
5299 17.909705   Dell_f0:ee:42      Broadcast          ARP        60 Who has 10.150.47.139? Tell 10.150.32.1
```

The packets highlighted/coloured green are the client’s packets. We can compare this to the previous screenshot and observe that the packets received are less in number.

4. **Jumping ahead in the video**

The video was started from the beginning. So, initially the number of packets was very high. After some time, at around 20 second, the buffer had sufficient data packets and hence, the number of packets fell. But at around that time, we jumped ahead in the video. As a result,

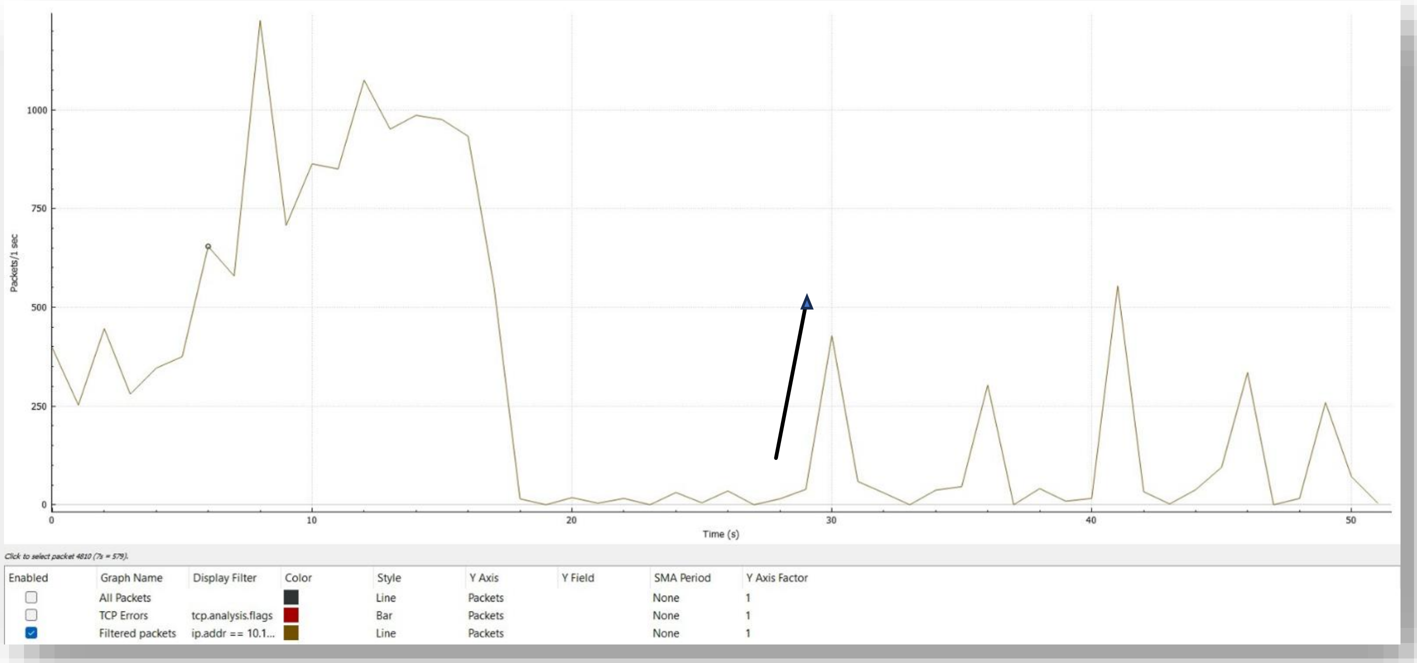


Click to select packet 1796 (0s = 1796).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period	Y Axis Factor
☑	All Packets		■	Line	Packets		None	1
☑	TCP Errors	tcp.analysis.flags	🟥	Bar	Packets		None	1
☑	Filtered packets	ip.addr == 10.1…	■	Line	Packets		None	1

the number of packets increased rapidly to keep up with the demand and ensure a smooth streaming of the video and avoid any buffering of the video.

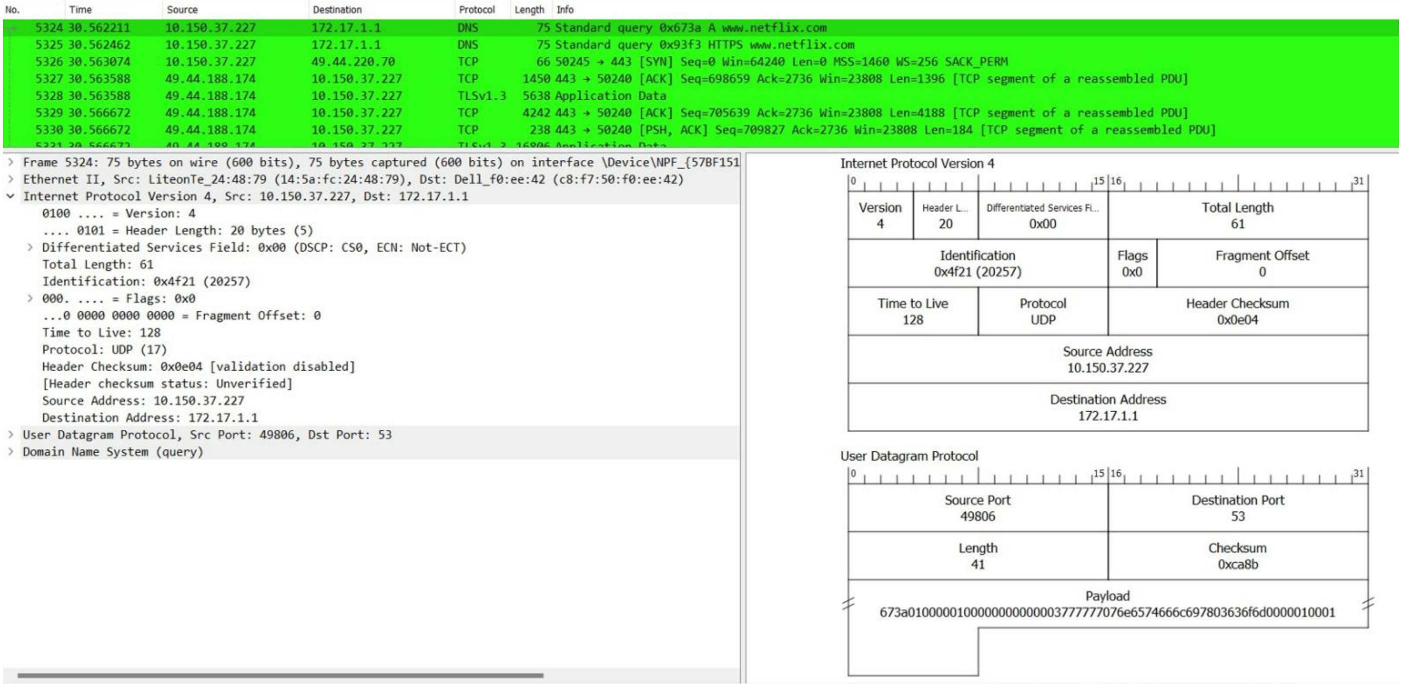
5. **Increasing speed of play**
   The video was started from the beginning. So, initially the number of packets was very high. After some time, at around 20 seconds, the buffer had sufficient data packets and hence, the number of packets fell. At around 27 or 28 seconds, we increased the speed of play to 1.5x. As a result, the number of packets increased rapidly.



Click to select packet 4810 (7s = 579).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period	Y Axis Factor
☐	All Packets		■	Line	Packets		None	1
☐	TCP Errors	tcp.analysis.flags	🟥	Bar	Packets		None	1
☑	Filtered packets	ip.addr == 10.1...	🟫	Line	Packets		None	1

6. **Going to next video**
   When we click the option to go to the next video, the client makes a request to the DNS server for the next video. Also, since we are going to play a different video, the inflow of packets from the server increases at once. Here, we clicked next video at around 30 seconds.



```
No.   Time          Source          Destination     Protocol  Length  Info
 5324 30.562211     10.150.37.227   172.17.1.1      DNS          75 Standard query 0x673a A www.netflix.com
 5325 30.562462     10.150.37.227   172.17.1.1      DNS          75 Standard query 0x93f3 HTTPS www.netflix.com
 5326 30.563074     10.150.37.227   49.44.220.70    TCP          66 50245 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 5327 30.563588     49.44.188.174   10.150.37.227   TCP        1450 443 → 50240 [ACK] Seq=698659 Ack=2736 Win=23808 Len=1396 [TCP segment of a reassembled PDU]
 5328 30.563588     49.44.188.174   10.150.37.227   TLSv1.3    5638 Application Data
 5329 30.566672     49.44.188.174   10.150.37.227   TCP        4242 443 → 50240 [ACK] Seq=705639 Ack=2736 Win=23808 Len=4188 [TCP segment of a reassembled PDU]
 5330 30.566672     49.44.188.174   10.150.37.227   TCP         238 443 → 50240 [PSH, ACK] Seq=709827 Ack=2736 Win=23808 Len=184 [TCP segment of a reassembled PDU]
```

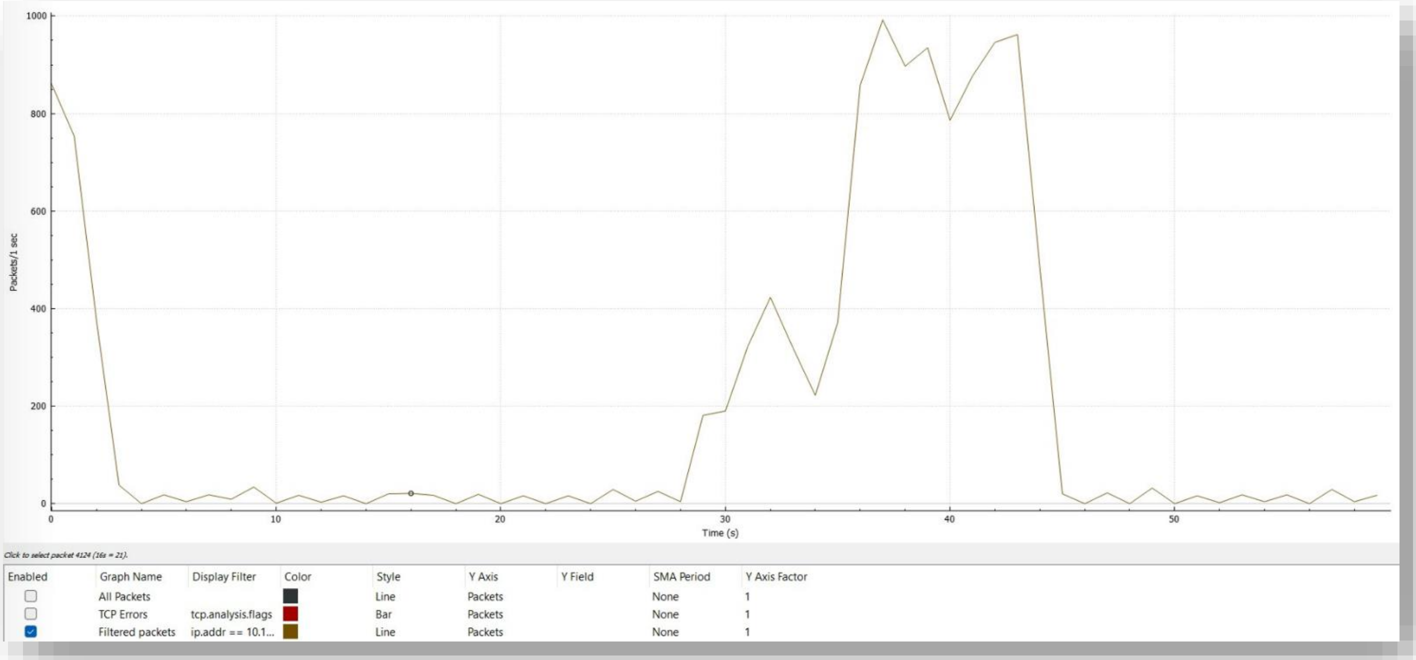
```
> Frame 5324: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface \Device\NPF_{57BF151
> Ethernet II, Src: LiteonTe_24:48:79 (14:5a:fc:24:48:79), Dst: Dell_f0:ee:42 (c8:f7:50:f0:ee:42)
v Internet Protocol Version 4, Src: 10.150.37.227, Dst: 172.17.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 61
    Identification: 0x4f21 (20257)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x0e04 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.150.37.227
    Destination Address: 172.17.1.1
> User Datagram Protocol, Src Port: 49806, Dst Port: 53
> Domain Name System (query)
```

Internet Protocol Version 4

Version 4	Header L... 20	Differentiated Services Fi... 0x00	Total Length 61	
Identification 0x4f21 (20257)			Flags 0x0	Fragment Offset 0
Time to Live 128	Protocol UDP	Header Checksum 0x0e04		
Source Address 10.150.37.227				
Destination Address 172.17.1.1				

User Datagram Protocol

Source Port 49806	Destination Port 53
Length 41	Checksum 0xca8b
Payload 673a01000001000000000000037777770 76e6574666c697803636f6d0000010001	



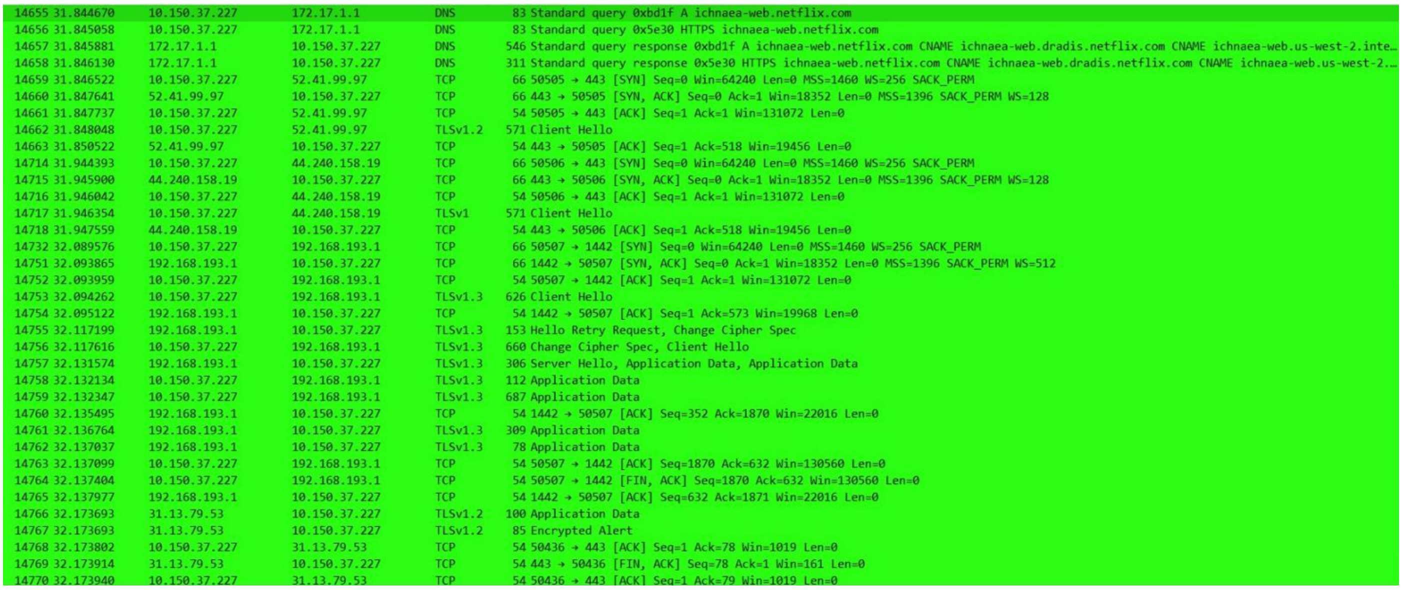
Click to select packet 4124 (16s = 21).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period	Y Axis Factor
☐	All Packets		■	Line	Packets		None	1
☐	TCP Errors	tcp.analysis.flags	🟥	Bar	Packets		None	1
☑	Filtered packets	ip.addr == 10.1...	🟫	Line	Packets		None	1

The spike at around 30 seconds shows the increase in the number of packets.

7. **Closing the video**

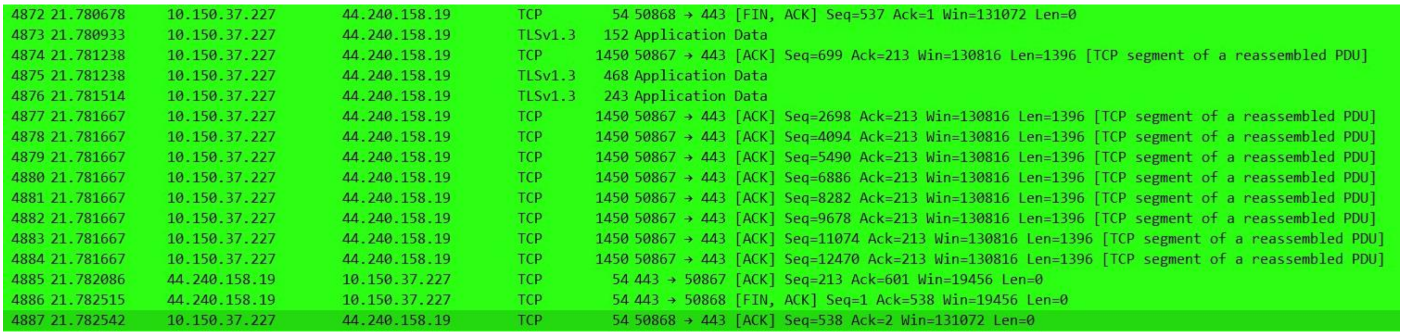
   Once we close the video, we go back to the home screen. In Netflix, short trailers of the content are played automatically in the home screen. This is done by the **ichnaea-web.netflix.com.** This domain is responsible for the short previews that auto play on any movie or series page within Netflix. The previews that normally have the volume muted by default. So, as soon as we close the video, the client connects to this domain.



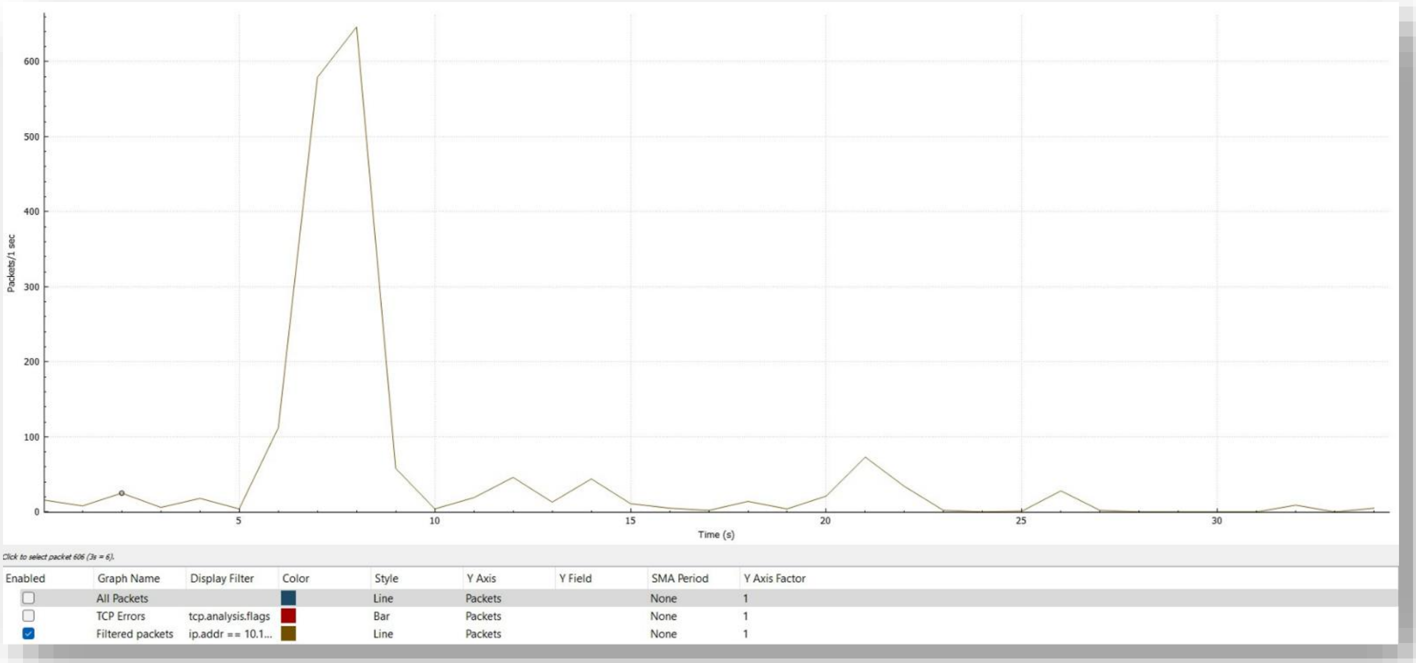
```
14655 31.844670   10.150.37.227   172.17.1.1      DNS       83 Standard query 0xbd1f A ichnaea-web.netflix.com
14656 31.845058   10.150.37.227   172.17.1.1      DNS       83 Standard query 0x5e30 HTTPS ichnaea-web.netflix.com
14657 31.845881   172.17.1.1      10.150.37.227   DNS      546 Standard query response 0xbd1f A ichnaea-web.netflix.com CNAME ichnaea-web.dradis.netflix.com CNAME ichnaea-web.us-west-2.inte…
14658 31.846130   172.17.1.1      10.150.37.227   DNS      311 Standard query response 0x5e30 HTTPS ichnaea-web.netflix.com CNAME ichnaea-web.dradis.netflix.com CNAME ichnaea-web.us-west-2.…
14659 31.846522   10.150.37.227   52.41.99.97     TCP       66 50505 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
14660 31.847641   52.41.99.97     10.150.37.227   TCP       66 443 → 50505 [SYN, ACK] Seq=0 Ack=1 Win=18352 Len=0 MSS=1396 SACK_PERM WS=128
14661 31.847737   10.150.37.227   52.41.99.97     TCP       54 50505 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
14662 31.848048   10.150.37.227   52.41.99.97     TLSv1.2  571 Client Hello
14663 31.850522   52.41.99.97     10.150.37.227   TCP       54 443 → 50505 [ACK] Seq=1 Ack=518 Win=19456 Len=0
14714 31.944393   10.150.37.227   44.240.158.19   TCP       66 50506 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
14715 31.945900   44.240.158.19   10.150.37.227   TCP       66 443 → 50506 [SYN, ACK] Seq=0 Ack=1 Win=18352 Len=0 MSS=1396 SACK_PERM WS=128
14716 31.946042   10.150.37.227   44.240.158.19   TCP       54 50506 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
14717 31.946354   10.150.37.227   44.240.158.19   TLSv1    571 Client Hello
14718 31.947559   44.240.158.19   10.150.37.227   TCP       54 443 → 50506 [ACK] Seq=1 Ack=518 Win=19456 Len=0
14732 32.089576   10.150.37.227   192.168.193.1   TCP       66 50507 → 1442 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
14751 32.093865   192.168.193.1   10.150.37.227   TCP       66 1442 → 50507 [SYN, ACK] Seq=0 Ack=1 Win=18352 Len=0 MSS=1396 SACK_PERM WS=512
14752 32.093959   10.150.37.227   192.168.193.1   TCP       54 50507 → 1442 [ACK] Seq=1 Ack=1 Win=131072 Len=0
14753 32.094262   10.150.37.227   192.168.193.1   TLSv1.3  626 Client Hello
14754 32.095122   192.168.193.1   10.150.37.227   TCP       54 1442 → 50507 [ACK] Seq=1 Ack=573 Win=19968 Len=0
14755 32.117199   192.168.193.1   10.150.37.227   TLSv1.3  153 Hello Retry Request, Change Cipher Spec
14756 32.117616   10.150.37.227   192.168.193.1   TLSv1.3  660 Change Cipher Spec, Client Hello
14757 32.131574   192.168.193.1   10.150.37.227   TLSv1.3  306 Server Hello, Application Data, Application Data
14758 32.132134   10.150.37.227   192.168.193.1   TLSv1.3  112 Application Data
14759 32.132347   10.150.37.227   192.168.193.1   TLSv1.3  687 Application Data
14760 32.135495   192.168.193.1   10.150.37.227   TCP       54 1442 → 50507 [ACK] Seq=352 Ack=1870 Win=22016 Len=0
14761 32.136764   192.168.193.1   10.150.37.227   TLSv1.3  309 Application Data
14762 32.137037   192.168.193.1   10.150.37.227   TLSv1.3   78 Application Data
14763 32.137099   10.150.37.227   192.168.193.1   TCP       54 50507 → 1442 [ACK] Seq=1870 Ack=632 Win=130560 Len=0
14764 32.137404   10.150.37.227   192.168.193.1   TCP       54 50507 → 1442 [FIN, ACK] Seq=1870 Ack=632 Win=130560 Len=0
14765 32.137977   192.168.193.1   10.150.37.227   TCP       54 1442 → 50507 [ACK] Seq=632 Ack=1871 Win=22016 Len=0
14766 32.173693   31.13.79.53     10.150.37.227   TLSv1.2  100 Application Data
14767 32.173693   31.13.79.53     10.150.37.227   TLSv1.2   85 Encrypted Alert
14768 32.173802   10.150.37.227   31.13.79.53     TCP       54 50436 → 443 [ACK] Seq=1 Ack=78 Win=1019 Len=0
14769 32.173914   31.13.79.53     10.150.37.227   TCP       54 443 → 50436 [FIN, ACK] Seq=78 Ack=1 Win=161 Len=0
14770 32.173940   10.150.37.227   31.13.79.53     TCP       54 50436 → 443 [ACK] Seq=1 Ack=79 Win=1019 Len=0
```

We can observe that, first, the client (PC) sends a query to the DNS server to get the domain's IP address. Then, TCP handshaking takes place between the Netflix server and the client. Also, a TLS handshake occurs. Then, the Netflix server starts sending application data.

8. **Closing the application**
   The client sends a FIN acknowledgement to halt the data transmission (piggybacked by an ACK) which is then ACKed by the server which in turn sends a FIN acknowledgement to the client and client acknowledges it to complete the **three-way TCP Termination handshake.**



```
4872 21.780678     10.150.37.227      44.240.158.19      TCP        54 50868 → 443 [FIN, ACK] Seq=537 Ack=1 Win=131072 Len=0
4873 21.780933     10.150.37.227      44.240.158.19      TLSv1.3   152 Application Data
4874 21.781238     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=699 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4875 21.781238     10.150.37.227      44.240.158.19      TLSv1.3   468 Application Data
4876 21.781514     10.150.37.227      44.240.158.19      TLSv1.3   243 Application Data
4877 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=2698 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4878 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=4094 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4879 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=5490 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4880 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=6886 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4881 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=8282 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4882 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=9678 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4883 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=11074 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4884 21.781667     10.150.37.227      44.240.158.19      TCP      1450 50867 → 443 [ACK] Seq=12470 Ack=213 Win=130816 Len=1396 [TCP segment of a reassembled PDU]
4885 21.782086     44.240.158.19      10.150.37.227      TCP        54 443 → 50867 [ACK] Seq=213 Ack=601 Win=19456 Len=0
4886 21.782515     44.240.158.19      10.150.37.227      TCP        54 443 → 50868 [FIN, ACK] Seq=1 Ack=538 Win=19456 Len=0
4887 21.782542     10.150.37.227      44.240.158.19      TCP        54 50868 → 443 [ACK] Seq=538 Ack=2 Win=131072 Len=0
```



Click to select packet 606 (3s = 6).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period	Y Axis Factor
☐	All Packets		■	Line	Packets		None	1
☐	TCP Errors	tcp.analysis.flags	■	Bar	Packets		None	1
☑	Filtered packets	ip.addr == 10.1...	■	Line	Packets		None	1

Here, we can observe that the application was closed around 21-22 seconds and then, the number of packets decreased to almost zero.

Task 4:

- **HTTP – Hypertext Transfer Protocol:**

When using the HTTP protocol, a browser requests a web page from a server. HTTP operates over TCP, establishing communication with the server through a TCP layer. Any content to be displayed on the browser screen, whether it's video, search results, or an error message, is provided by the server in response to this HTTP request. Video packets are also transmitted using a proprietary version of HTTP called DASH.

- **TLS - Transport Layer Security:**

TLS is a cryptographic protocol designed to secure communication between web applications and servers. It is utilized by web browsers when loading websites. The standard port for encrypted HTTPS traffic is 443, in contrast to port 80, which is used for unencrypted HTTP traffic. Based on packet analysis, we determined that our computer (IP: 10.150.37.227) uses TLSv1.3 for encryption, while the server (IP: 49.44.188.174) uses TLSv1.2. Without TLS, malicious actors could potentially initiate a Man-in-the-Middle attack to intercept packets, compromising user privacy.

- **DNS - Domain Name System:**
The DNS protocol is essential for resolving URL addresses and requesting files hosted on Netflix's servers. DNS assigns an alias domain name to www.netflix.com and then translates this alias into an IP address. Netflix's servers are hosted on Amazon AWS. DNS communication is carried out using the UDP transport mechanism.

- **TCP – Transmission Control Protocol:**
TCP plays a crucial role in transmitting and receiving all HTTP-related queries at both the client and server ends. We observed that the 'Seq' and 'Ack' fields in the TCP header indicate the current packet sequence number being transferred and the expected sequence number for the response packet. Specifically, Ack = Seq + TCP segment Length. These packets are transmitted over the network layer using the Internet Protocol (IP). TCP ensures the ordered delivery of packets. Although packets may come out of order, TCP rearranges them before sending them to application.

- **UDP – User Datagram Protocol**
Netflix exclusively utilizes UDP for resolving DNS queries. UDP is a connectionless protocol that offers faster data transfer but lower reliability. It is particularly valuable when individual packets need to be transmitted across the network. UDP packets feature smaller headers, making them lightweight and thus facilitating quicker data transfer.

- **IPv4 – Internet Protocol version 4:**
In the context of Netflix, IPv4 serves as the backbone for data communication over the internet. However, due to its lack of reliability and ordered data transfer, Netflix pairs IPv4 with TCP (Transmission Control Protocol) to ensure that content is delivered reliably and in the correct sequence, providing users with a smooth streaming experience.

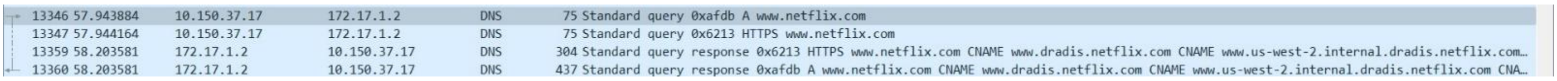
Task 5:

Indeed, we noticed that video packets that have been played are subject to caching. These incoming video packets are initially stored in a buffer and then subsequently moved to a cache once they have been viewed. This caching mechanism comes into play when using Netflix's rewind feature, allowing you to go back 10 seconds in the video.

We also observed that the cache for video packets is noticeably smaller in size compared to the buffer. If we attempt a significant rewind, such as going back 20 minutes, the video playback is paused until the browser requests the subsequent packets starting from that point. If the packets preceding the 20-minute mark had already been cached, the video would have loaded much more quickly, and there would have been no buffering delays.

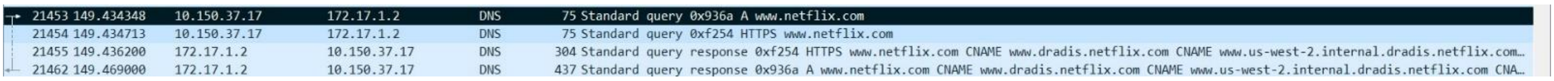
In DNS query, there exists a field called TTL (Time to live) this indicates the presence of web cache for holding DNS requests, as TTL represents the maximum time a packet can stay in cache before getting replaced. We made 3 consecutive requests. First DNS request has RTT of around 0.24s. But subsequent requests take considerably less time, around 0.02s and 0.1s.

- **First DNS quer**y:



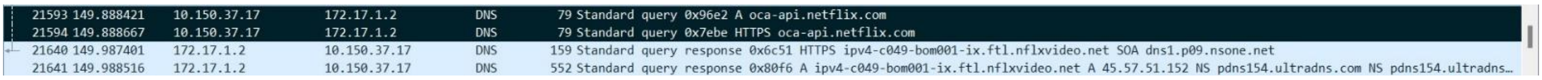
```
13346 57.943884   10.150.37.17   172.17.1.2     DNS   75 Standard query 0xafdb A www.netflix.com
13347 57.944164   10.150.37.17   172.17.1.2     DNS   75 Standard query 0x6213 HTTPS www.netflix.com
13359 58.203581   172.17.1.2     10.150.37.17   DNS  304 Standard query response 0x6213 HTTPS www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.com…
13360 58.203581   172.17.1.2     10.150.37.17   DNS  437 Standard query response 0xafdb A www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.com CNA…
```

- **Second DNS query:**



```
21453 149.434348  10.150.37.17   172.17.1.2     DNS   75 Standard query 0x936a A www.netflix.com
21454 149.434713  10.150.37.17   172.17.1.2     DNS   75 Standard query 0xf254 HTTPS www.netflix.com
21455 149.436200  172.17.1.2     10.150.37.17   DNS  304 Standard query response 0xf254 HTTPS www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.com…
21462 149.469000  172.17.1.2     10.150.37.17   DNS  437 Standard query response 0x936a A www.netflix.com CNAME www.dradis.netflix.com CNAME www.us-west-2.internal.dradis.netflix.com CNA…
```

- **Third DNS query:**



```
21593 149.888421  10.150.37.17   172.17.1.2     DNS   79 Standard query 0x96e2 A oca-api.netflix.com
21594 149.888667  10.150.37.17   172.17.1.2     DNS   79 Standard query 0x7ebe HTTPS oca-api.netflix.com
21640 149.987401  172.17.1.2     10.150.37.17   DNS  159 Standard query response 0x6c51 HTTPS ipv4-c049-bom001-ix.ftl.nflxvideo.net SOA dns1.p09.nsone.net
21641 149.988516  172.17.1.2     10.150.37.17   DNS  552 Standard query response 0x80f6 A ipv4-c049-bom001-ix.ftl.nflxvideo.net A 45.57.51.152 NS pdns154.ultradns.com NS pdns154.ultradns…
```

Task 6:

We have calculated the following statistics and tabulated them while performing experiments at different time of the day:

Throughput, RTT, Packet size, Number of packets lost, Number of UDP & TCP packets, Number of responses received with respect to one request sent.

Time and Place	4pm Hostel	7pm Lab	10am Library
Throughput (Kilobytes/s)	251	382	377
Round Trip Time (ms)	0.711	2.54	1.85
Avg. Packet Size (Bytes)	917	1132	1170
No. of Packets Lost	0	0	0
No. of TCP Packets	40559	47452	45241
No. of UDP Packets	225	222	198
No. of Responses per Request Sent	1.98	2.58	1.81