Andre Hei Wang Law

40175600

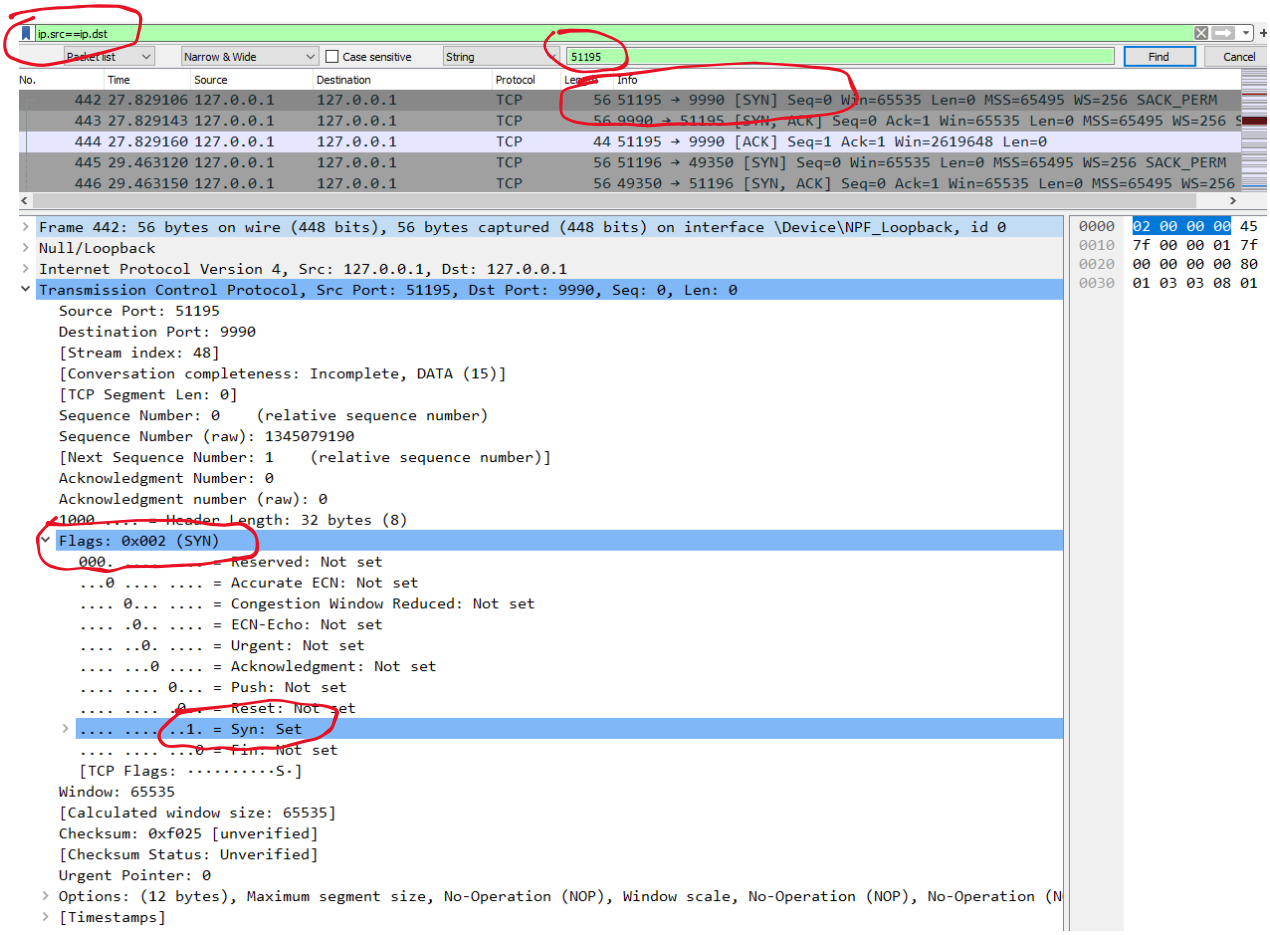
Design Document - Wireshark

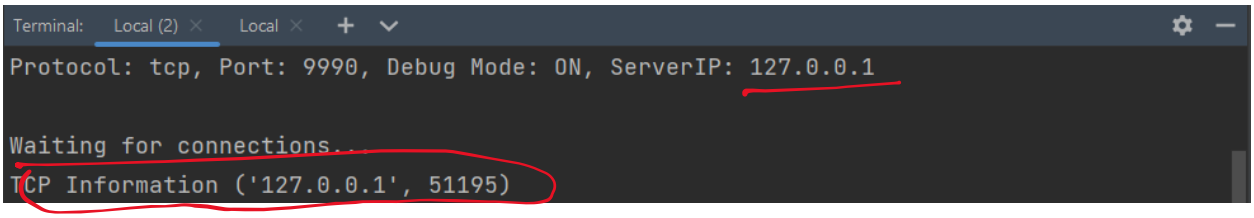
Task:

Open the Wireshark while running the client-server program, and capture the packet that is sent from server to the client. To recognize the packets that are being transferred from the server program, check the source and destination addresses. The source and destination address will be 127.0.0.1 as you are using localhost

For TCP:

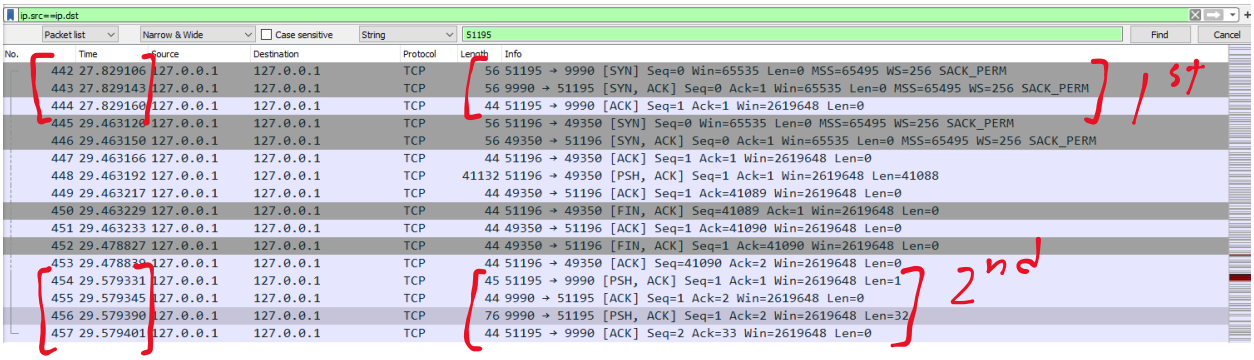
- What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client and server? What is in the segment that identifies as a SYN segment?





Sequence number of SYN is 0. “Syn” flag is “Set” means that it is a SYN segment.

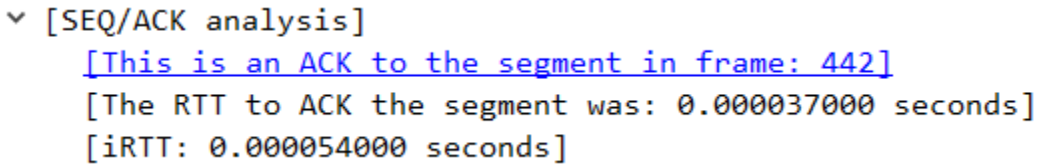
- What are the sequence numbers of the first two segments in the TCP connection? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgment was received, what is the RTT value for each of the two segments? Build the round-trip time graph.



Sequence number of the first two segments in the TCP connection is 0.

First segment sent at time 27.829106. ACK received at time 27.829143. 3rd handshake at time 27.829160.

Second segment sent at time 29.579331. ACK received at time 29.579345. 3rd handshake at 29.579401.



RTT for first segment: 0.000054

RTT for second segment: 0.00007

- What is the length of each of the first six TCP segments?

```
    TCP payload (32 bytes)
> Data (32 bytes)

    TCP payload (1 byte)
> Data (1 byte)

    TCP payload (32 bytes)
> Data (32 bytes)

    TCP payload (1 byte)
> Data (1 byte)

    TCP payload (32 bytes)
> Data (32 bytes)
```

• What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

```
127.0.0.1    127.0.0.1    TCP    44 9990 → 51195 [ACK] Seq=1 Ack=2 Win=2619648 Len=0
127.0.0.1    127.0.0.1    TCP    76 9990 → 51195 [PSH, ACK] Seq=1 Ack=2 Win=2619648 Len=32
127.0.0.1    127.0.0.1    TCP    44 51195 → 9990 [ACK] Seq=2 Ack=33 Win=2619648 Len=0
```

Since "Win" are all 2619648, the minimum amount of buffer space is 2619648. Thus, it does not throttle.

• Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

For this connection, there are no retransmitted segments in the trace file. This is due to the fact that no same sequence number appears at two different times, thus no re-requests of previous segments.

• How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing in every other received segment.

```
Transmission Control Protocol, Src Port: 51195, Dst Port: 9990, Seq: 1, Ack: 1, Len: 1
    Source Port: 51195
    Destination Port: 9990
    [Stream index: 48]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 1]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1345079191
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 503166755
    0101 .... = Header Length: 20 bytes (5)
```

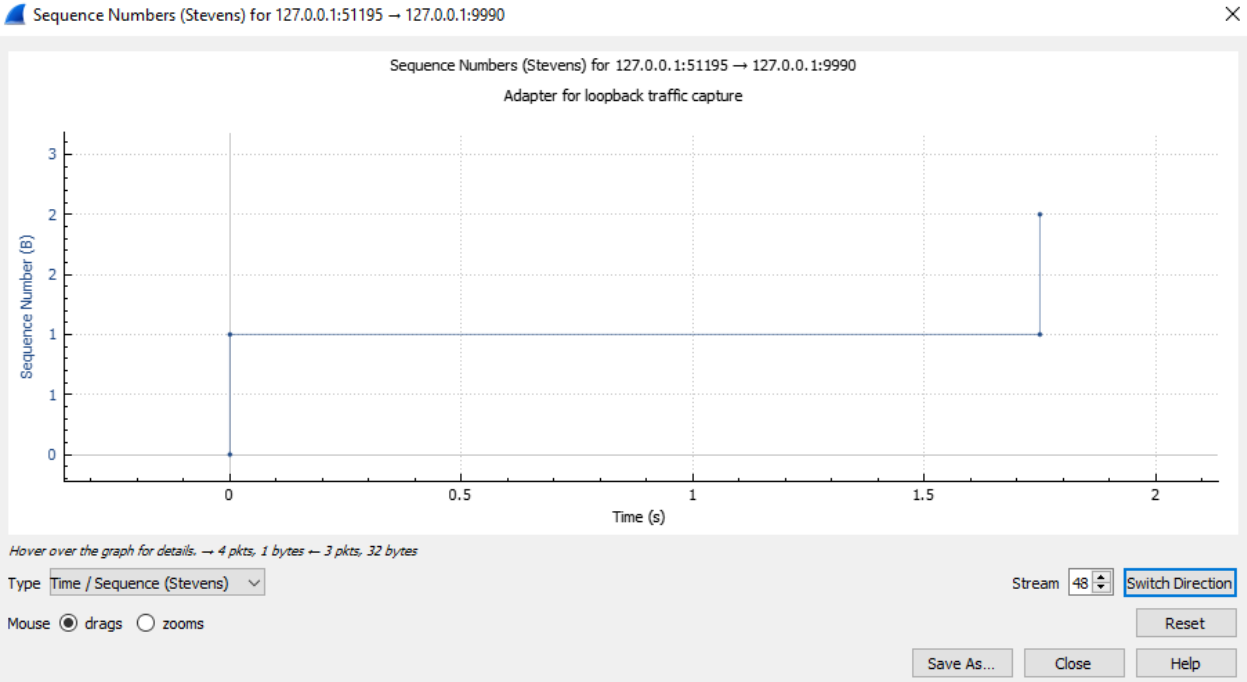
The ACK number is 1. I couldn't find in my cases of a receiver ACKing every other received segment.

• What is the throughput (bytes transferred per unit of time) for the TCP connection? Explain how you calculated this value

```
Transmission Control Protocol, Src Port: 51196, Dst Port: 49350, Seq: 41090, Ack: 2, Len: 0
    Source Port: 51196
    Destination Port: 49350
    [Stream index: 49]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 41090    (relative sequence number)
```

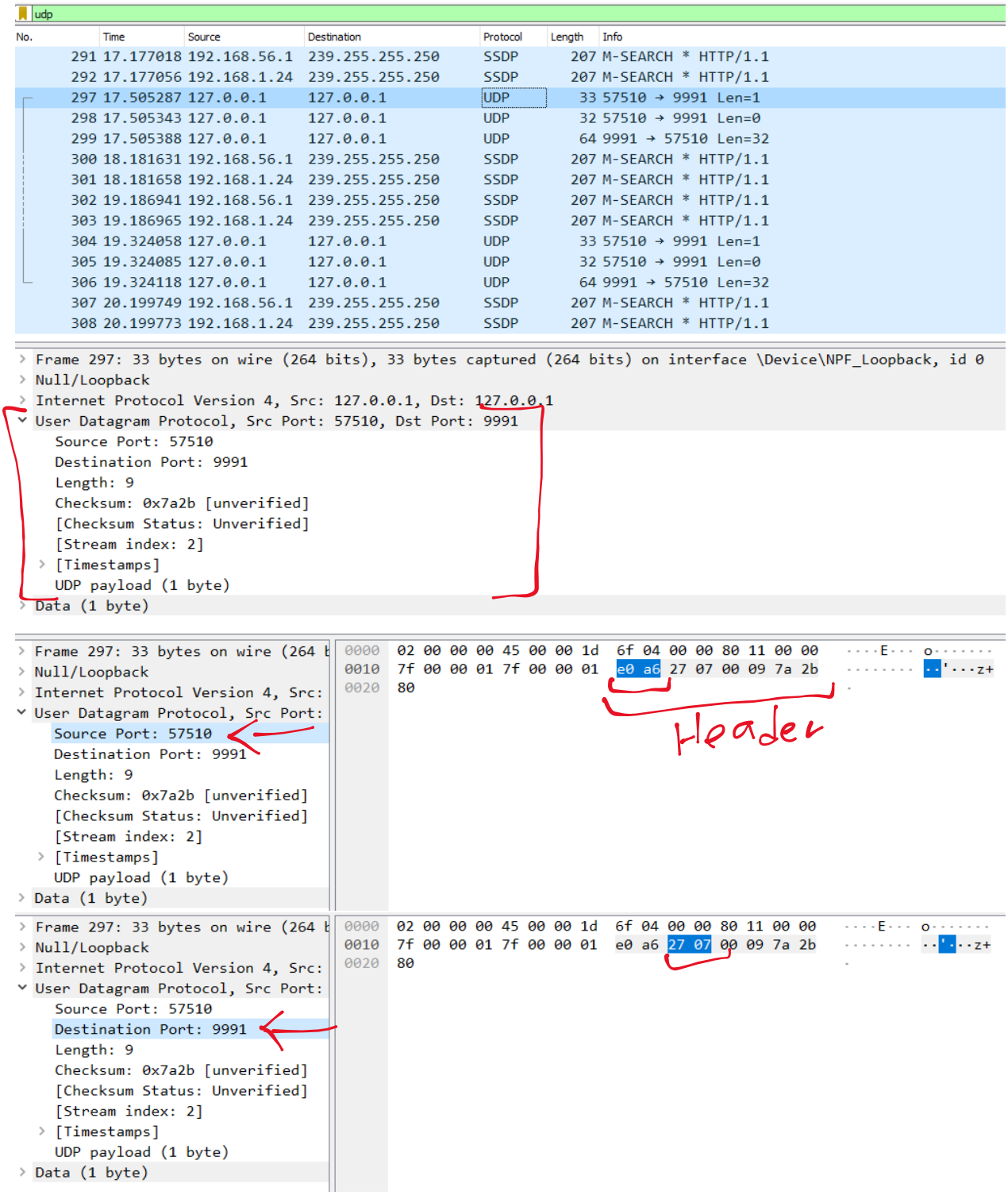
Throughput represents the number of bytes transferred per time. The number of bytes transferred is the last segment number minus the first segment number, thus 41090-1=41089 bytes. The 'per time' can be calculated based on the difference between the first and last segment time which is 1.750295 seconds. The throughput is then 41089/1.750295=23475.4712777 bytes per second.

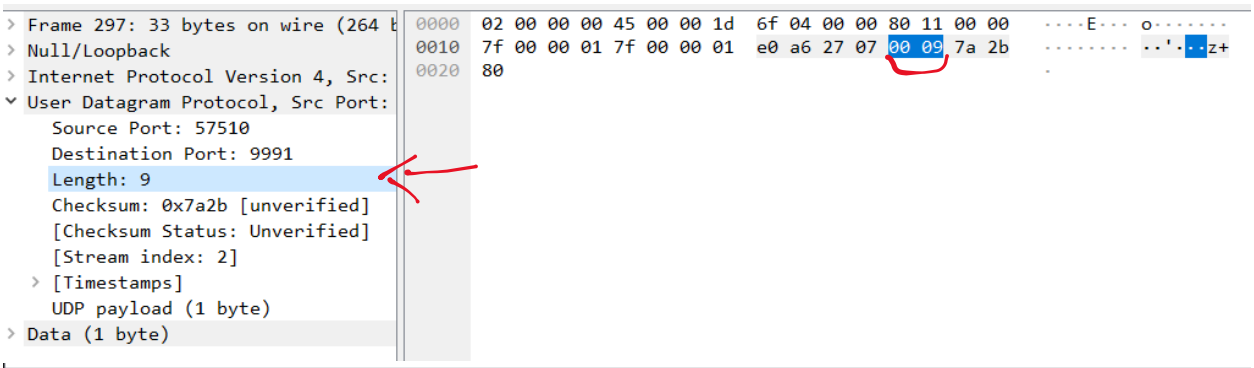
• Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the server.

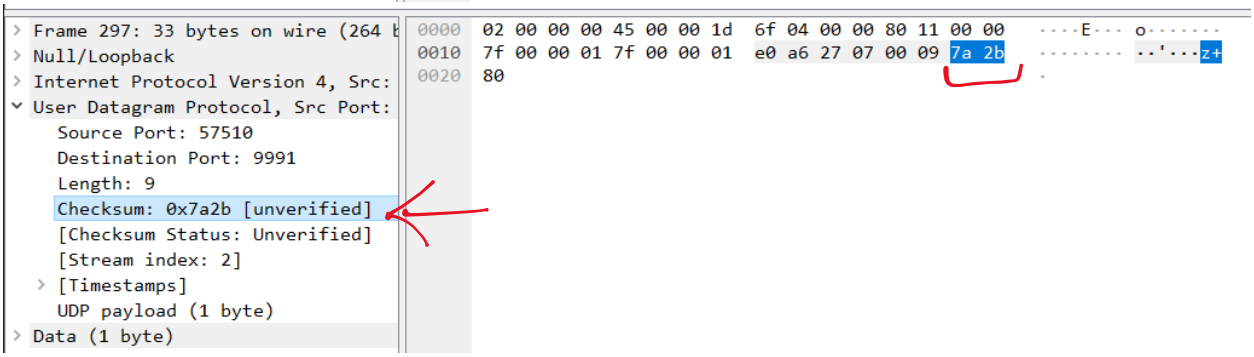


For UDP:

- Select one UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. Name these fields.





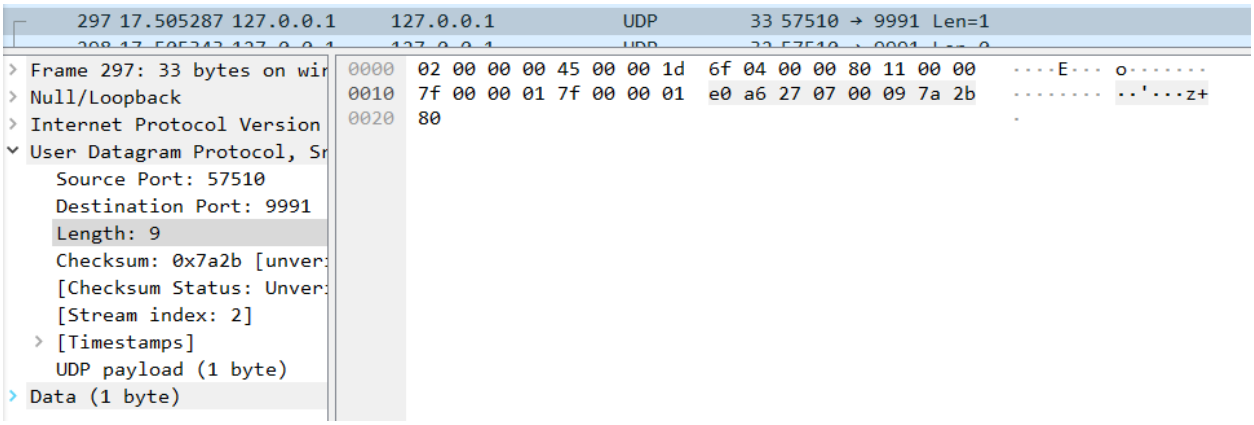


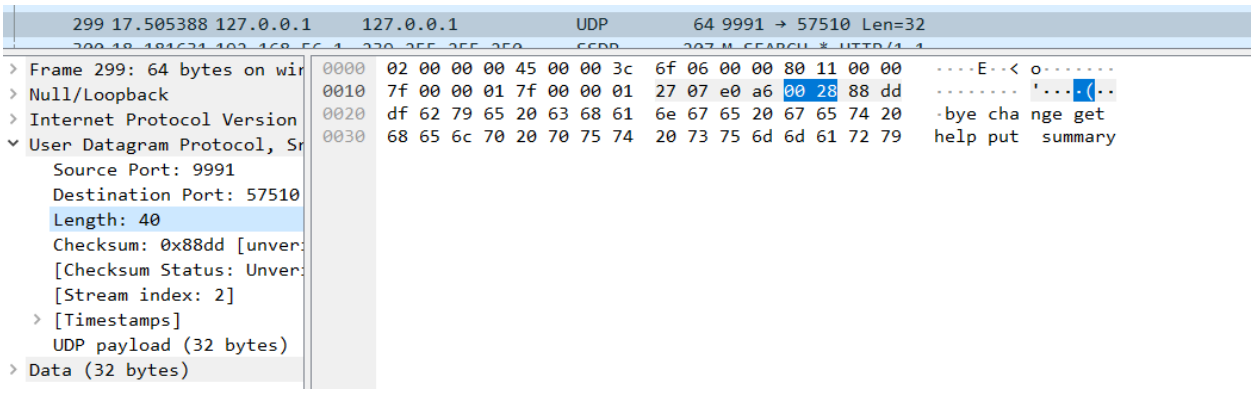
For a UDP packet, the fields in “User Datagram Protocol” are the source port, the destination port, the length and the checksum.

• By consulting the displayed information in Wireshark’s packet content field for this packet, determine the length (in bytes) of each of the UDP header fields

Based on the four images above, the length in bytes of each of the UDP header fields 2 bytes. This is because each field takes up 4 hexadecimal (4 bits), meaning 4 times 4 bits or 16 bits which is 2 bytes. In other words, each field is 2 bytes and the entire UDP header is 8 bytes.

• The value in the Length field is the length of what? Verify your claim with your captured UDP packet.





The field “length” represents the total number of bytes in the entire UDP segment (header + payload). This can be confirmed by trying to find the number of bytes in the UDP header and see if it is equal to 8 bytes.

For the first image, length is equal to 9 and the UDP payload is 1 byte. This is expected as this means the UDP header takes up 8 bytes as mentioned in question 2. The calculation is 9-1=8 bytes. We can also do the same for the second image where the length is 40 and the payload is 32. This means that the UDP header is 40-32=8 bytes which is also the expect answer.

Implementation Design of the Client Program:

-Code Structure:

-Function Prototype: handle server response

-Function Prototype: send data to server

-Functions Prototype: put, get, change, summary, help, bye commands

-Prompt user information

-Setup client connection (TCP or UDP)

-Loop to allow user to perform commands

Implementation Design of the Server Program:

-Code Structure:

-Function Prototype: handle client request

-Prompt user information

-Setup server connection (TCP or UDP)