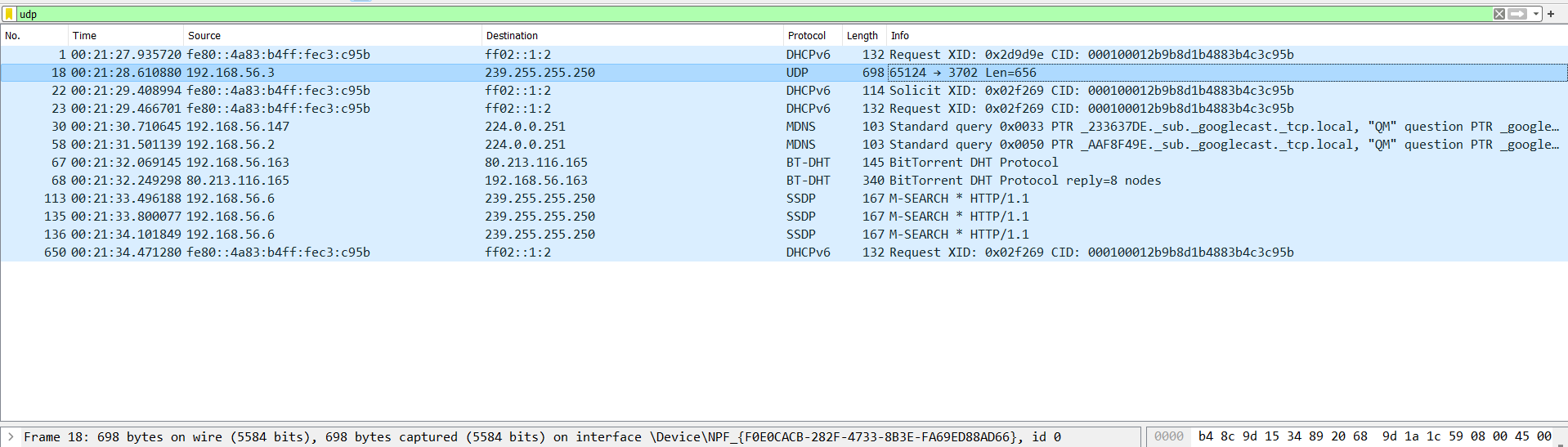
Assignment 2 UDP AND TCP 20K-0244

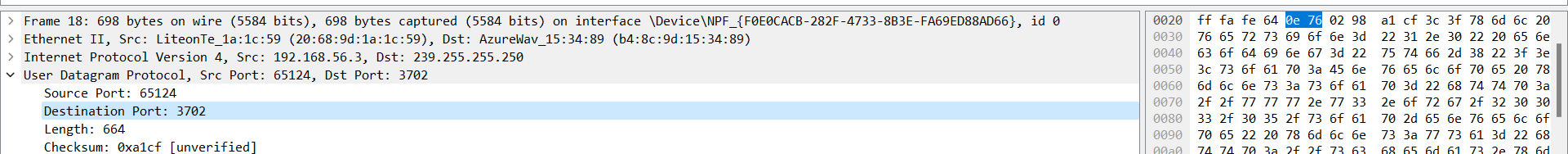
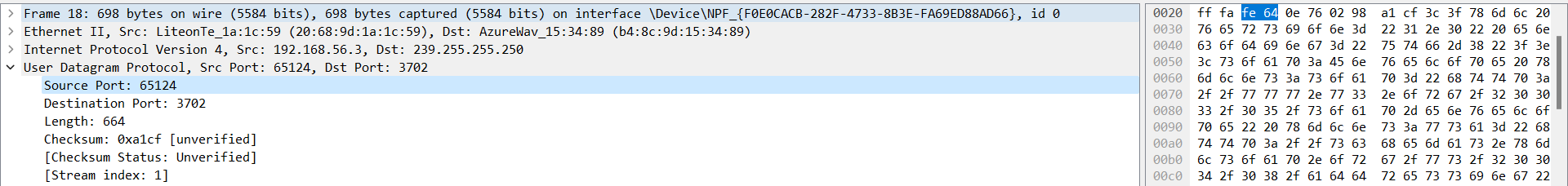
# UDP

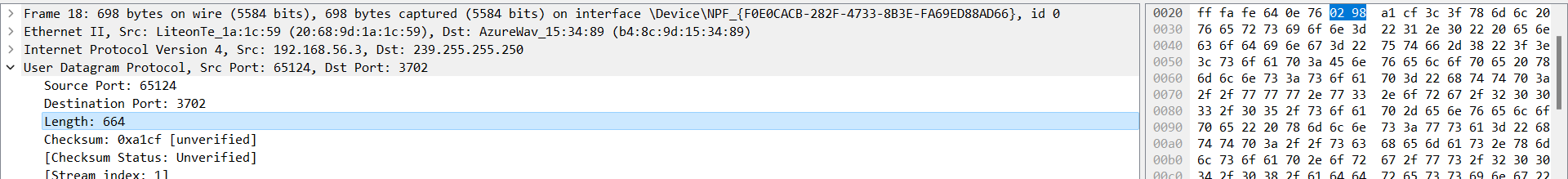
## 1)

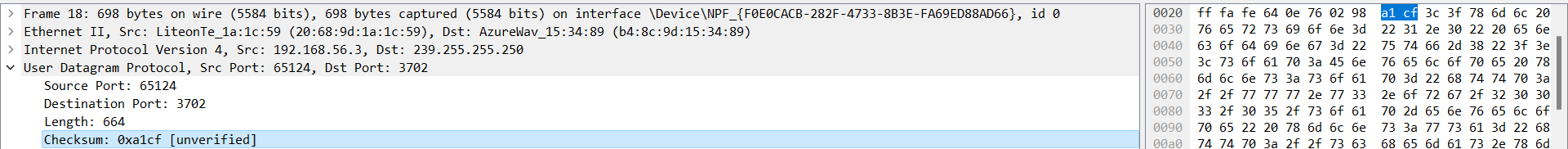


UDP PACKET CHOSEN

I note that in my UDP there is source port, destination port, length, checksum.(had a timestamp but it was not coming in the file like these 4 when hioghlighted)





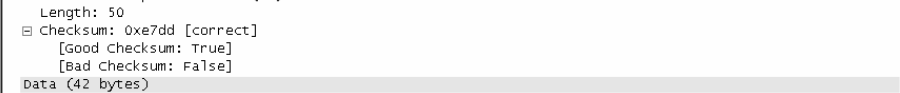


## 2)

Note in 1 that each part of header has 2 bytes, so total header will equal 8 bytes.

## 3)

The length attribute specifies how several bytes belong to the UDP segment (header and data). As the size of a data field might vary from one UDP segments to the next, an explicit length number is required. So 42 data + 8 bytes header.



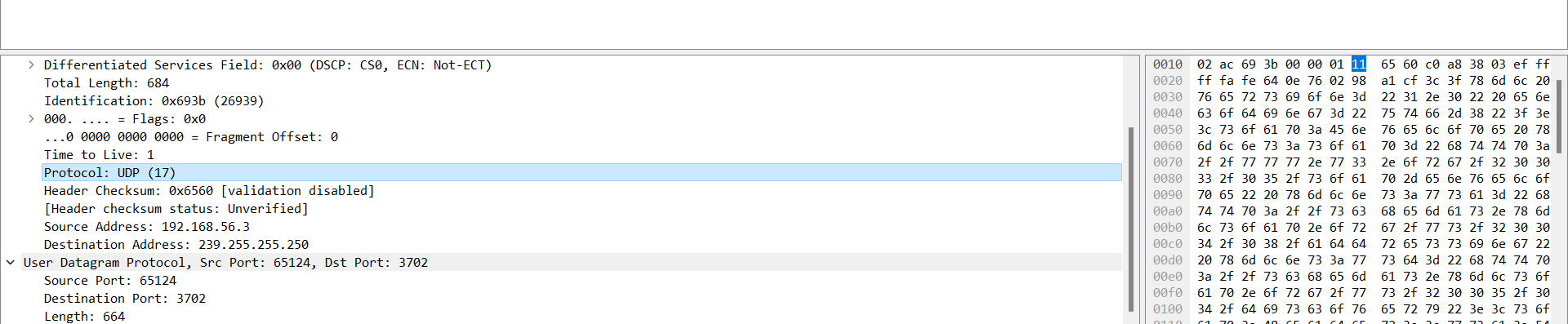
## 4)

A UDP Payload is the data in the segment without the header. UDP has 16 bits, therefore the maximum number of bytes in this will be (2 ^ 16 -1) to get 65535 bytes for range. But now subtract 8 header bytes so we get 65527 bytes.

## 5)

Similiarly the largest possible port number will become one where all bits used -1 for maximum value so 65535. The -1 represents maximum value and the 2 power bits show possible combinations. So 65536 combinations with 65535 as maximum port number.

## 6)



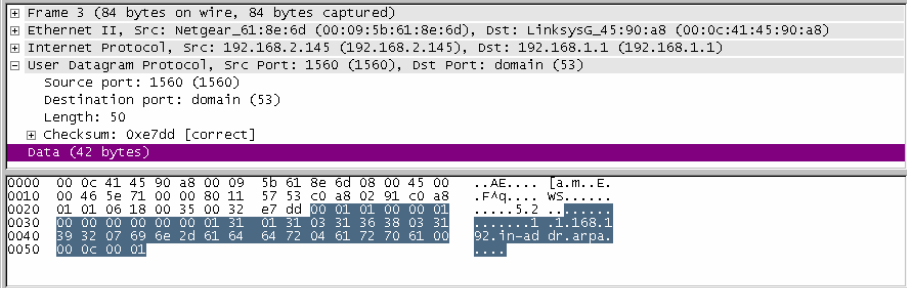
So UDP has 17 as a denary/decimal value but in hexadecimal, it is 0x11 as highlighted. Mind you the 0x is only present to represent a hexadecimal value.

## 7)

Google says it is optional to calculate the checksum. However, it is calculated over the header and data. So 16 bits are involved. To calculate checksum, we take the one’s complement for the one’s complement sum of information from the pseudo header for the UDP and IP header with the data. Then add bytes at the end to make a multiple for two bytes and then computer the checksum. IF the sum is zero, set it to 0xFFFF

The UDP checksum is calculated as the 16-bit one’s complement of the one’s complement sum of a pseudo header of information from the IP header, the UDP header, and the data. This is padded as needed with zero bytes at the end to make a multiple of two bytes. If the checksum is computed to be 0, it must be set to 0xFFFF.

## 8)



IP header: Source IP address c0a8 0291

IP header: Destination IP address c0a8 0101

IP header: Protocol number(zero padded on left) 0011

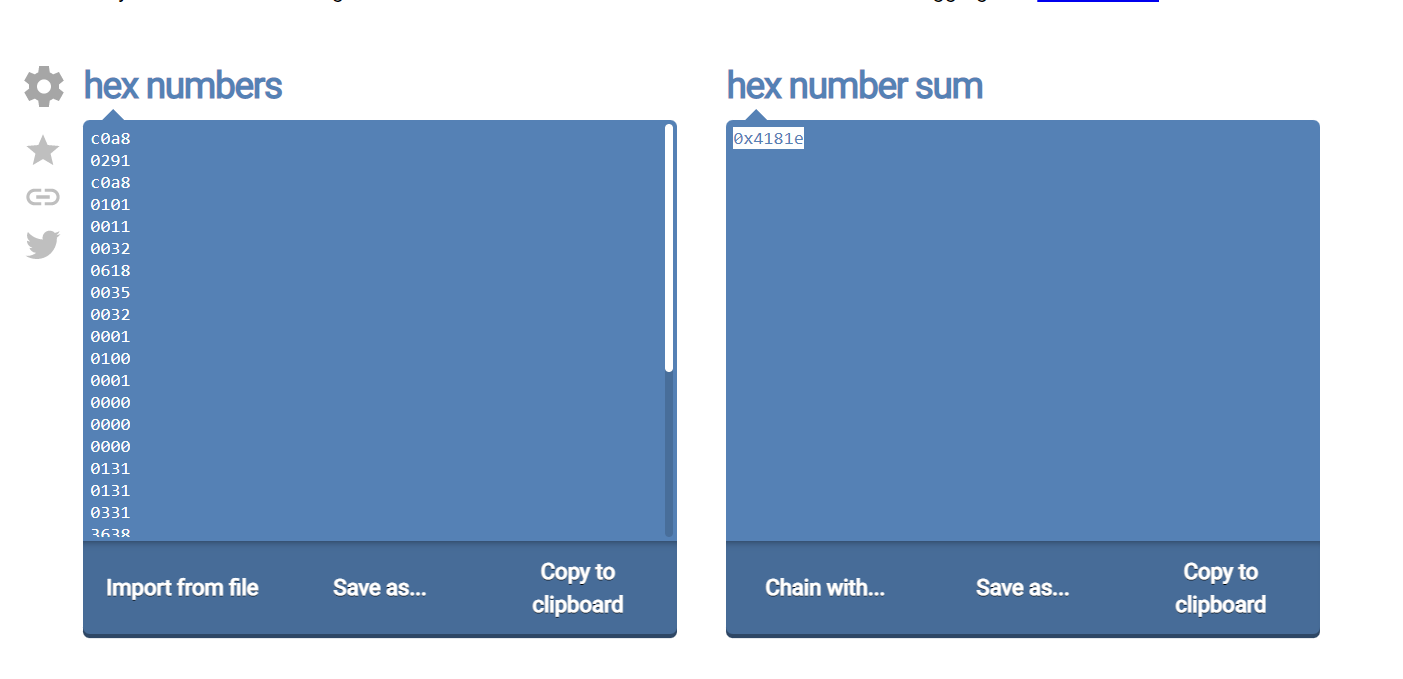
16 bit UDP Length 0032

UDP header: source port 0618

UDP header: destination port 0035

UDP header: length 0032

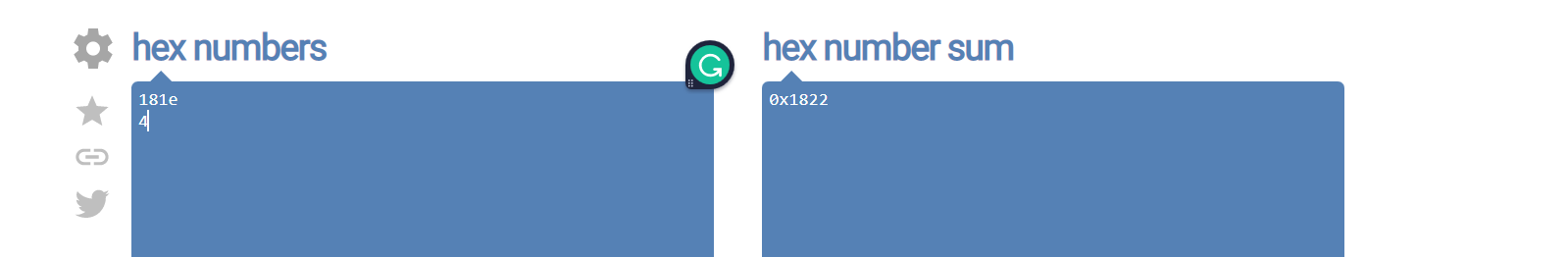
UDP Data 0001 0100 0001 0000 0000 0000 0131 0131 0331 3638 0331 3932 0769 6e2d 6164 6472 0461 7270 6100 000c 0001



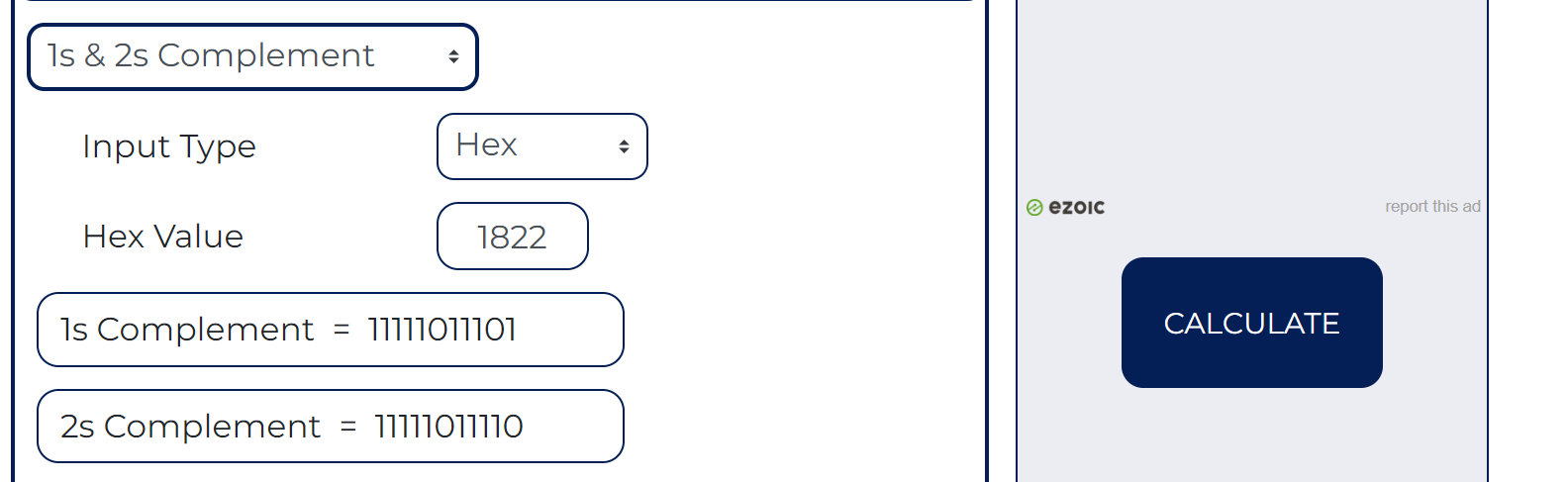
Sum all hex values to get 0x4181e

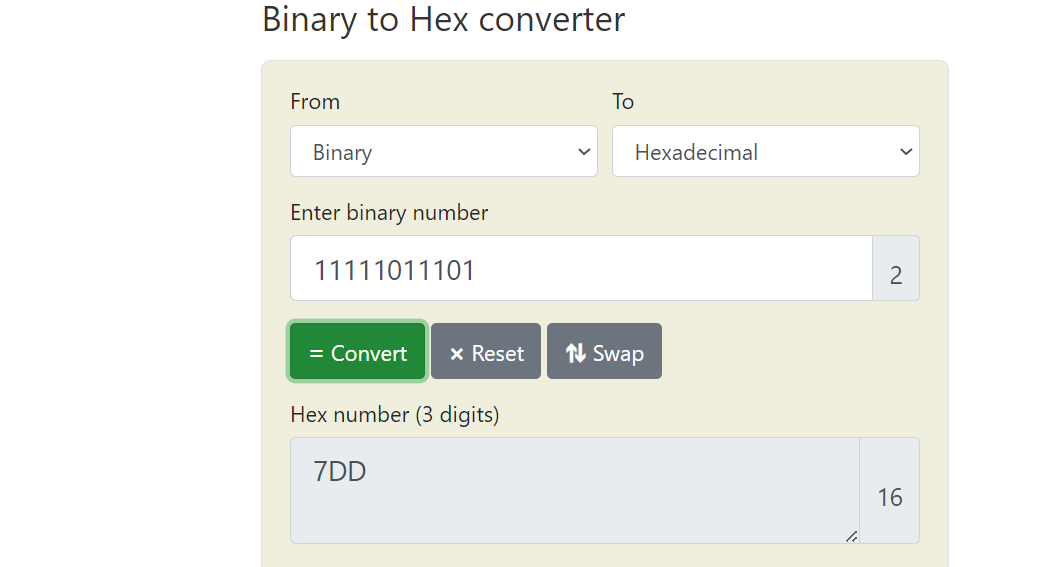
Carry the “4”

Add in the carry to get 1822



1s complement = checksum! E7dd

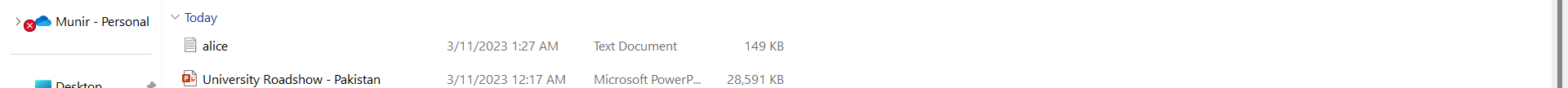


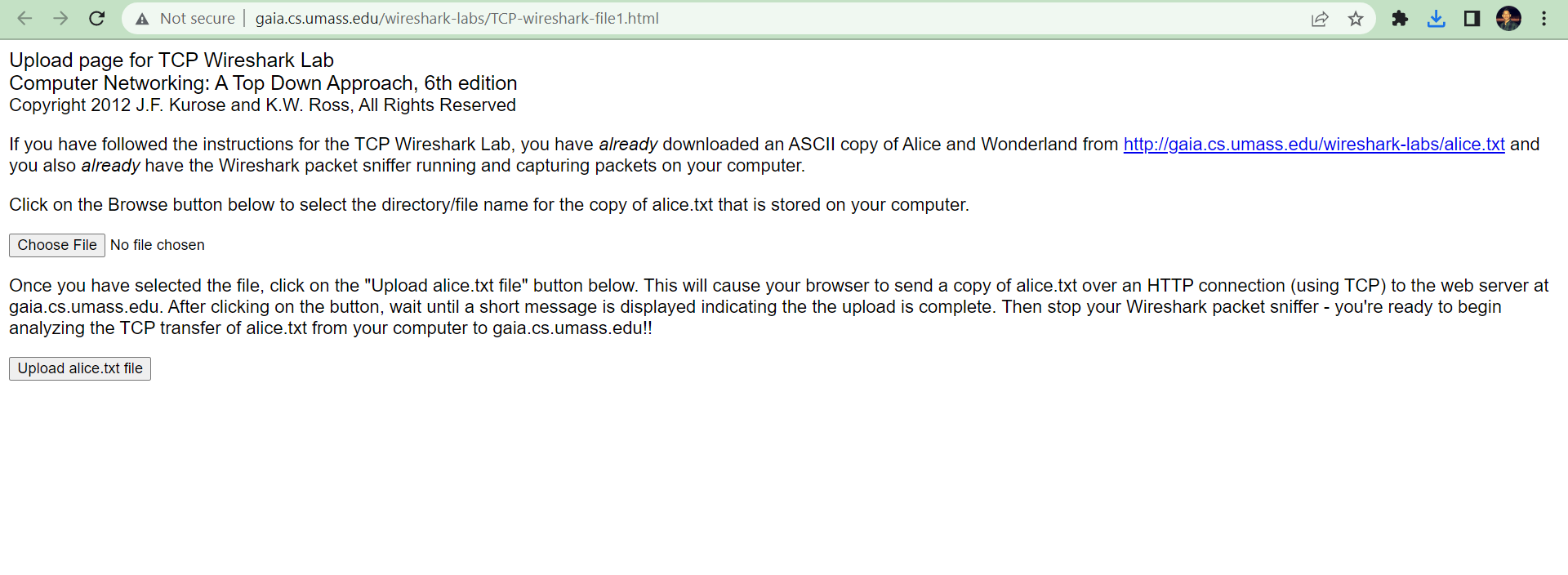


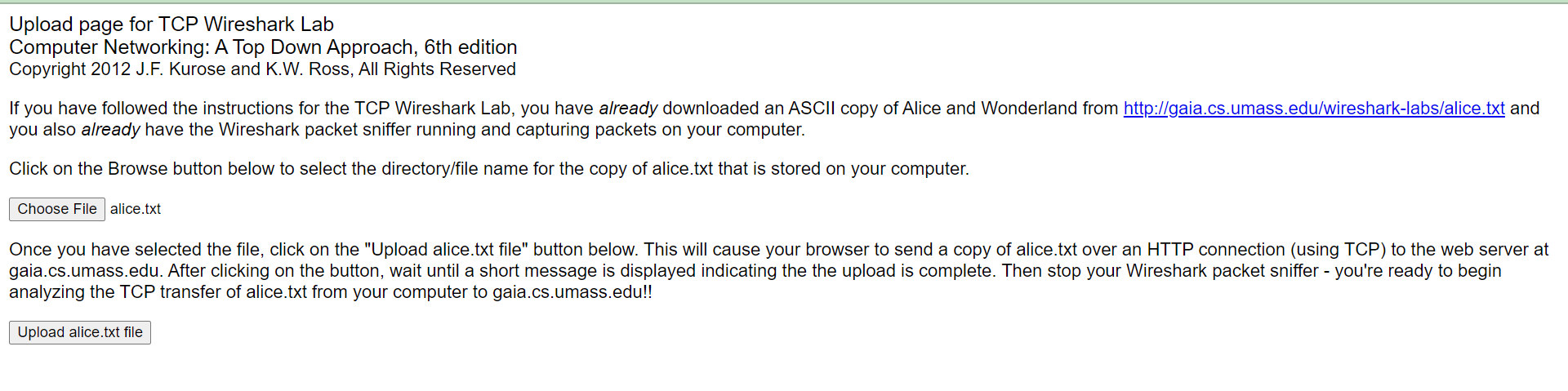


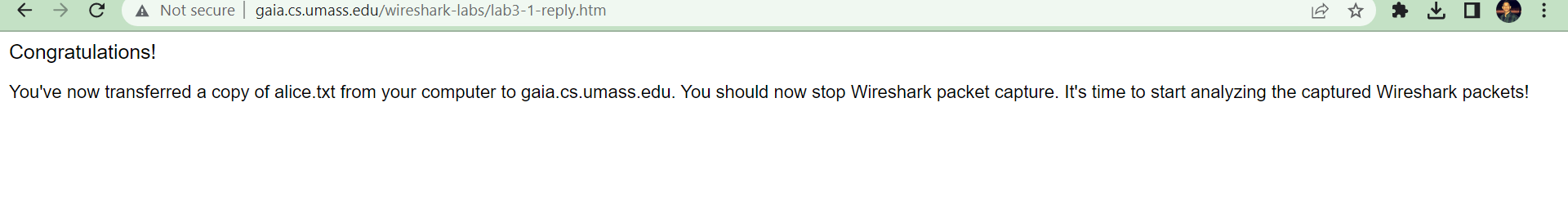
# TCP

Following steps









## 1)

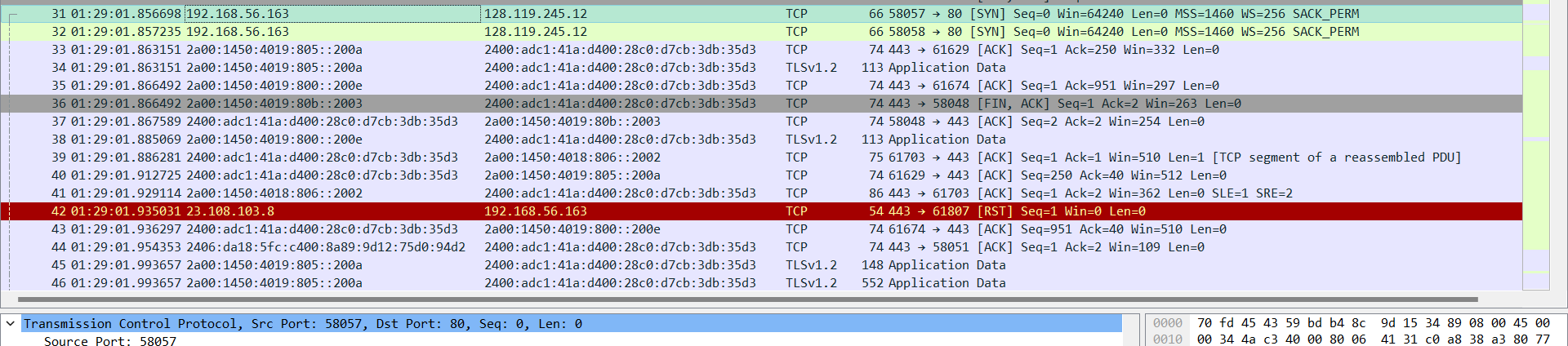
I noted the SYN,SYN ACK and ACK message for a three-way handshake as well.

Source port is 1161, and my source ip is 192.168.1.102.

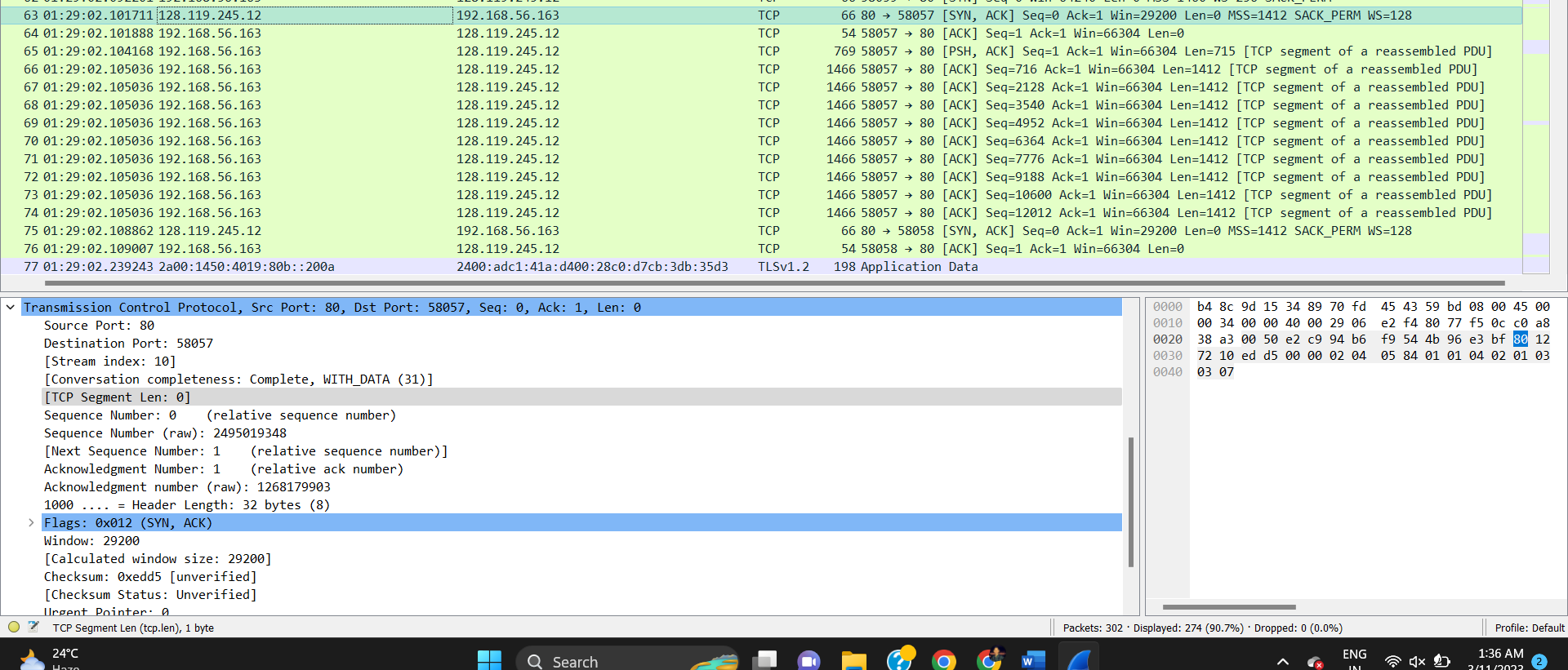
## 2)

Gaia source port is 80 and ip is 128.119.245.12. we are sure it is 80 as client sends to port 80 thus it has to be gaia using same port.

## 3)



Source port is 58057, and my source ip is 192.168.56.163.

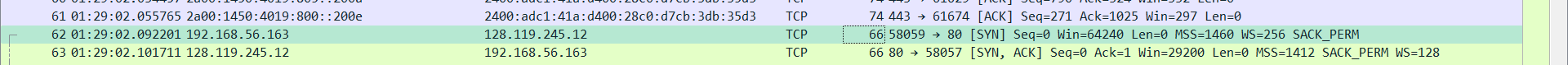


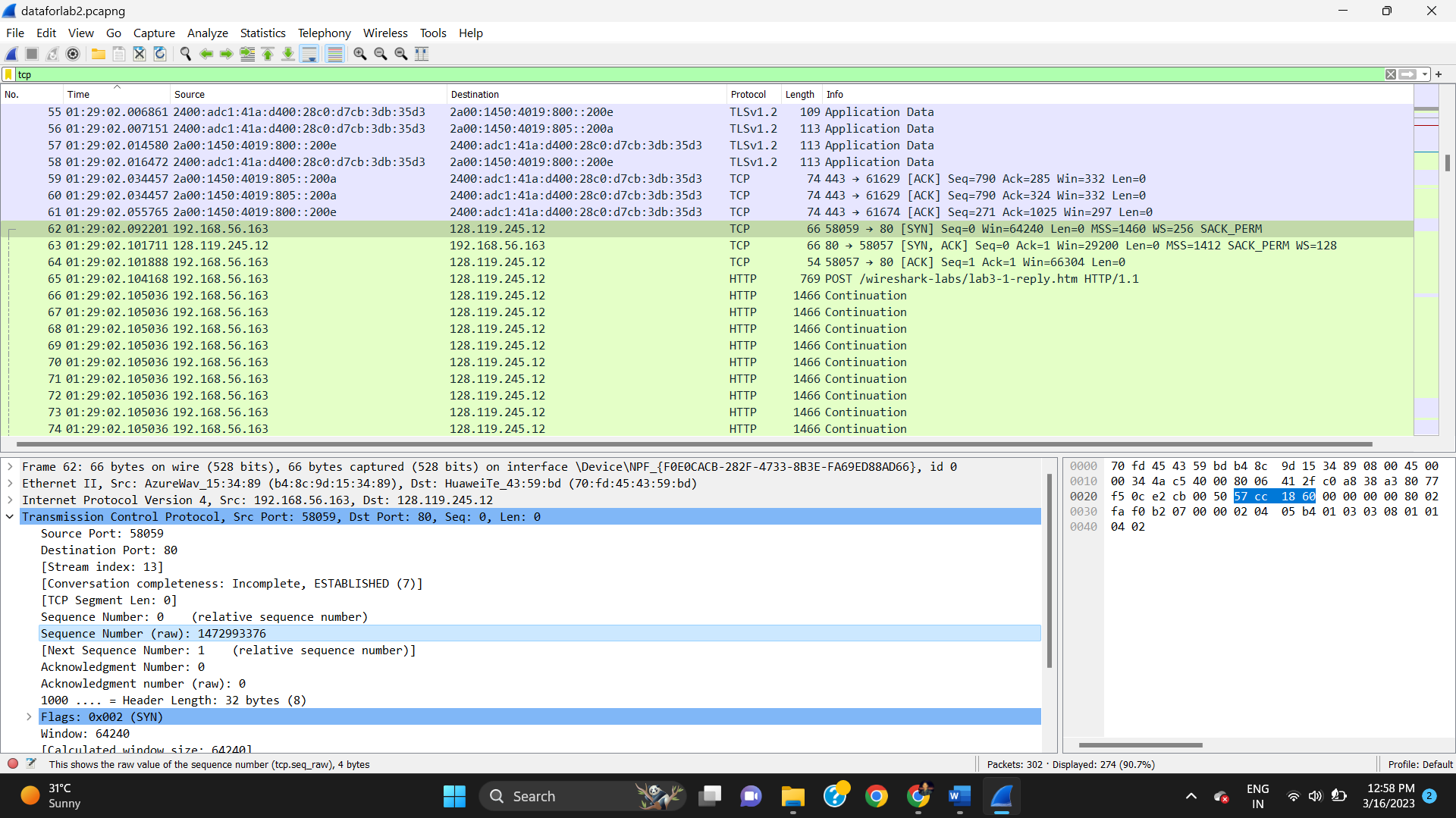
# TCP Basics

## 4)

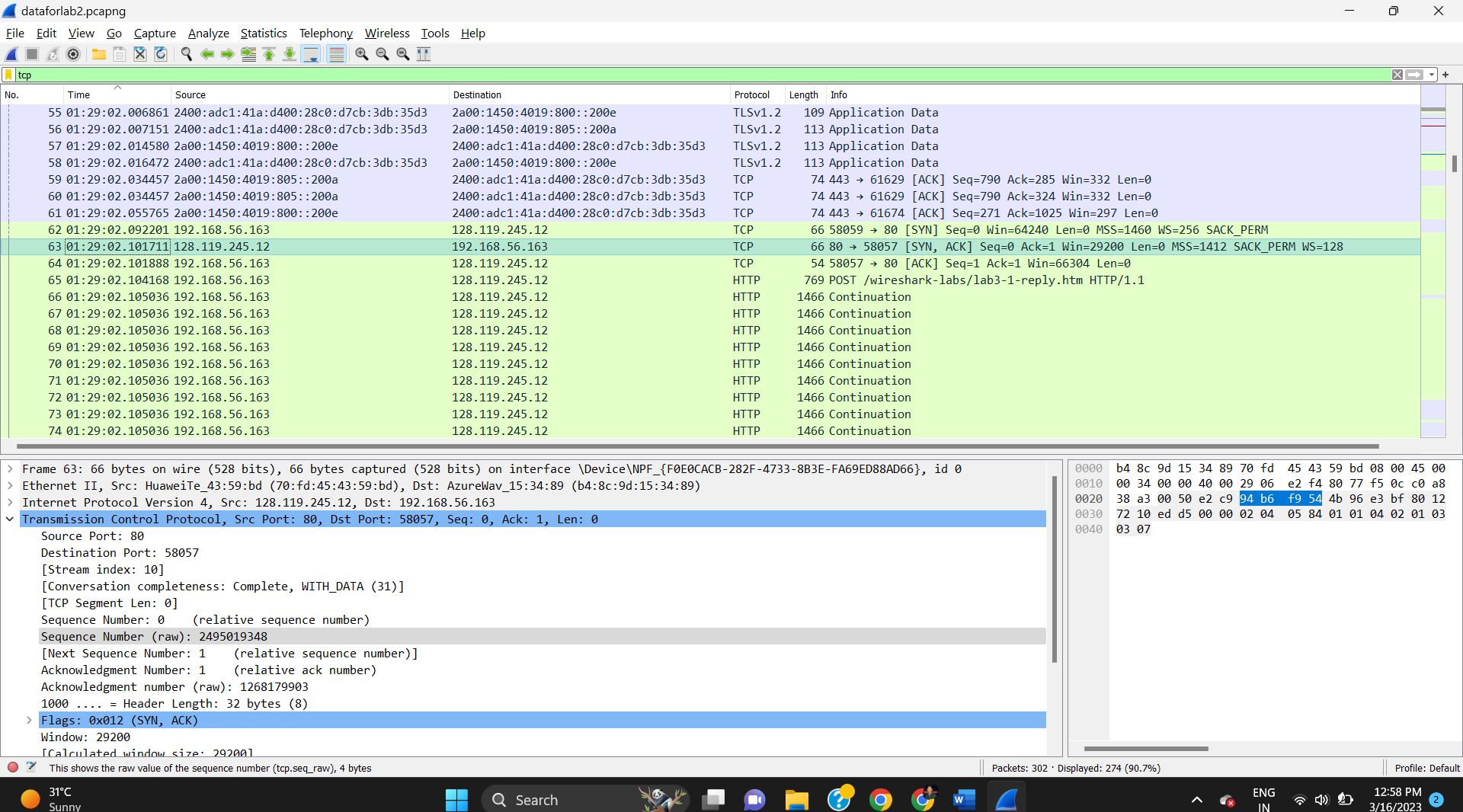
The sequence number used to begin the TCP was 0. We can clearly see a SYN flag which lets the users know that the following is a SYN segment. This is for the beginning for the 3 way handshake.

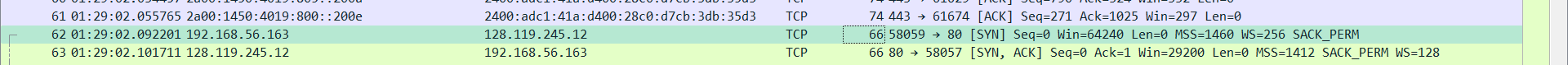
(Have included raw sequence number as well)





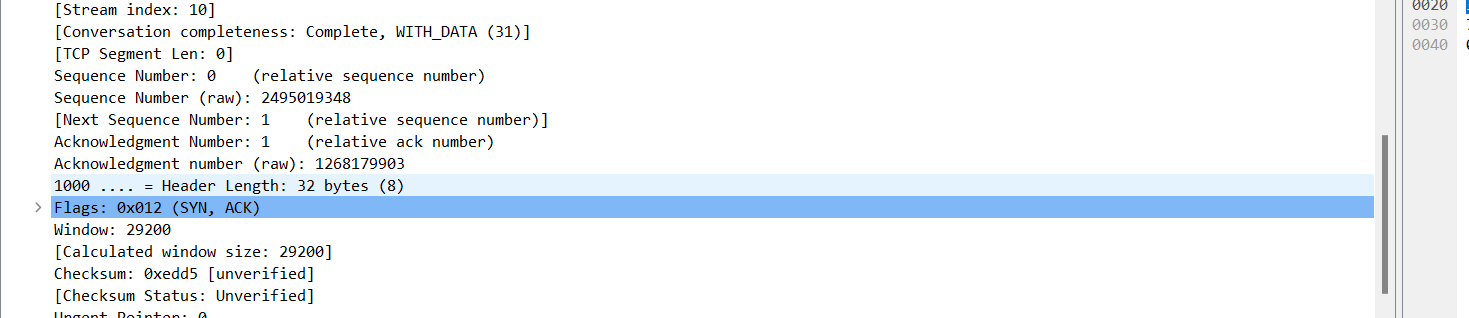
## 5)



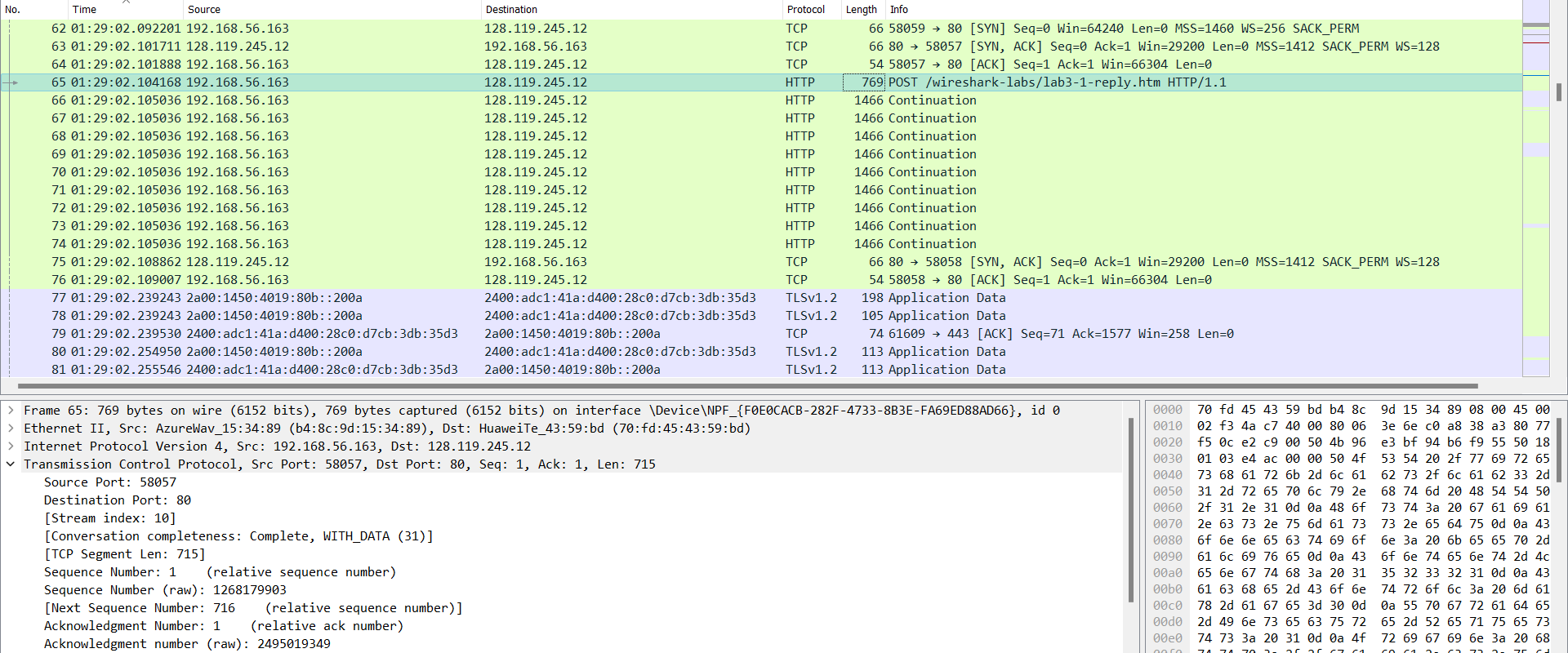


The sequence number for the SYN-ACK segment is 0. Then we see that the ack value is = 1. This is as sequence number plus 1 so now it is 1. We can also see a clear SYN-ACK flag.

(included the raw sequence number as well) 2495019348

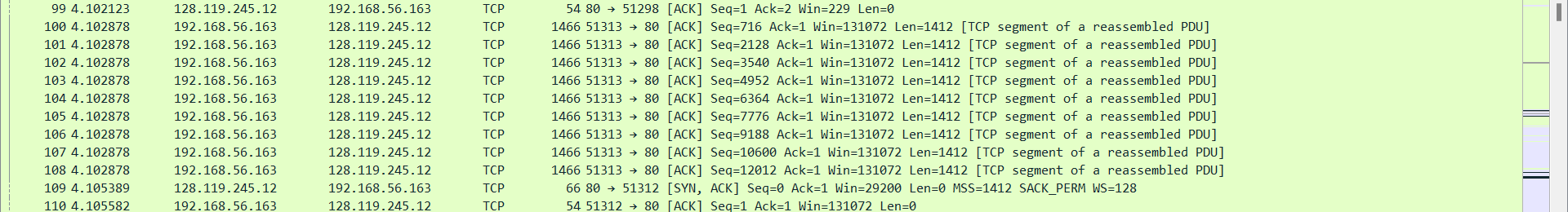


## 6)



Sequence number is 1. (also included raw one) 1258177903.

## 7)



Lets use the flow graph to make calculation easy.

Note that my segments travel as a single PDU, so we can deduce these segments are sent at the same times, and then the acknowledgment follows for each of the one.

1. 716 acknowledged at 
2. 3540 acknowledged at
3. 7776 acknowledged at
4. 13424 acknowledged at 
5. 19702 sent and acknowledged at 
6. 21896 sent and ack at

## 8) RTT times

Seq 1 => 4.344172-4.102878=0.241294 ms

Seq 2=>4.344172-4.102878=0.241294 ms

Seq 3=>4.344172-4.102878=0.241294 ms

Seq 4=>4.344172-4.102878=0.241294 ms

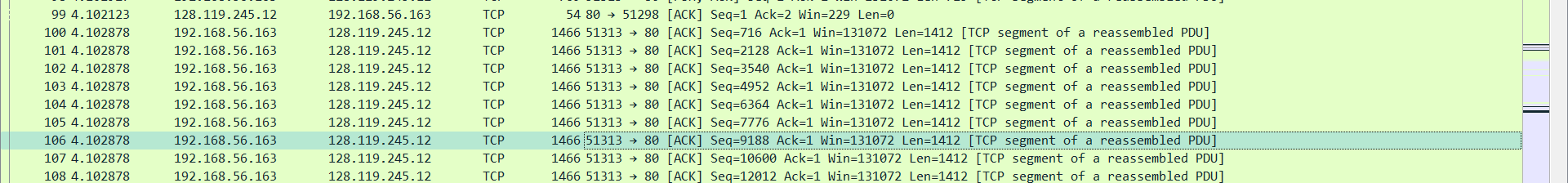
Sent in a TCP segment with reassembled PDU hence same

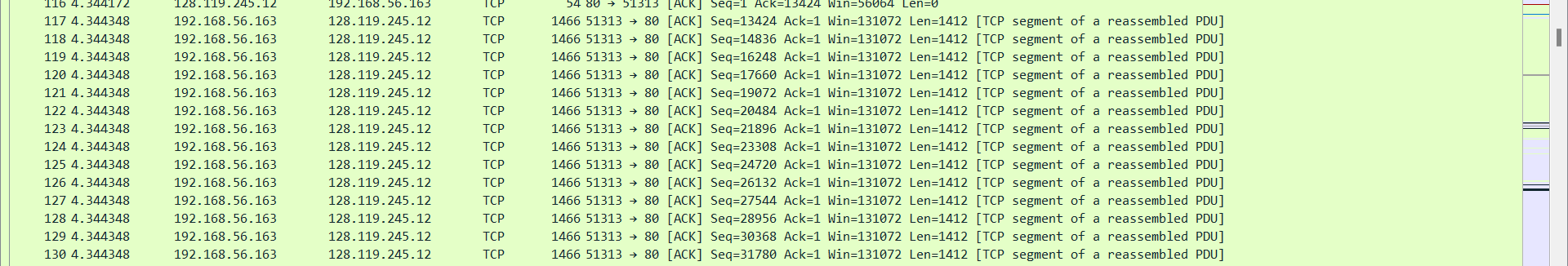
Seq 5=>4.591145-4.34438=0.246765 ms

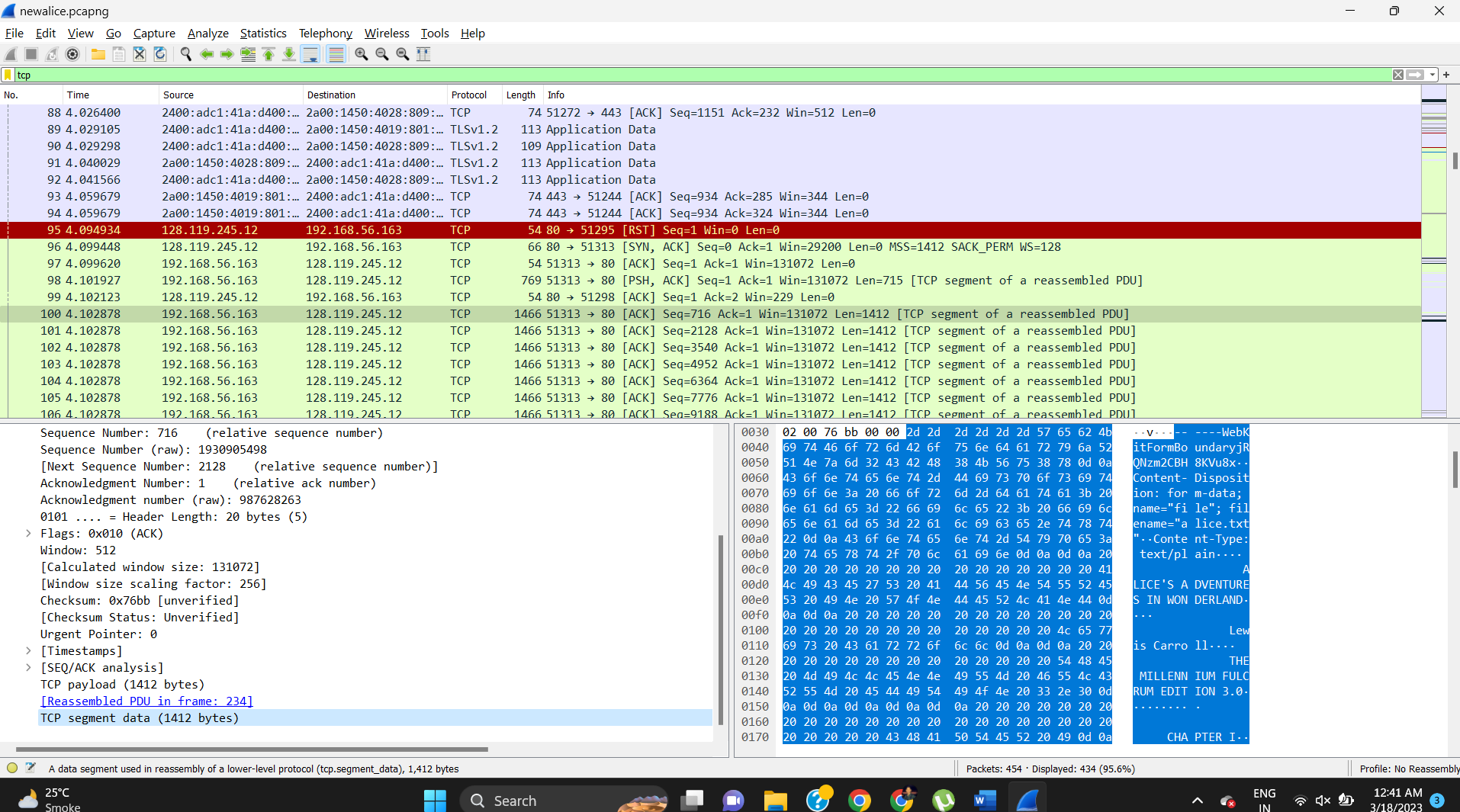
Seq 6=>4.591145-4.34438=0.246765 ms

## 9)

Lengths are 1466 with header, but content is 1412.







In terms of the acknowledgments are high.

Seq 1=716

Seq 2=2824

Seq 3= 4236

Seq 4=5648

Seq 5= 6278

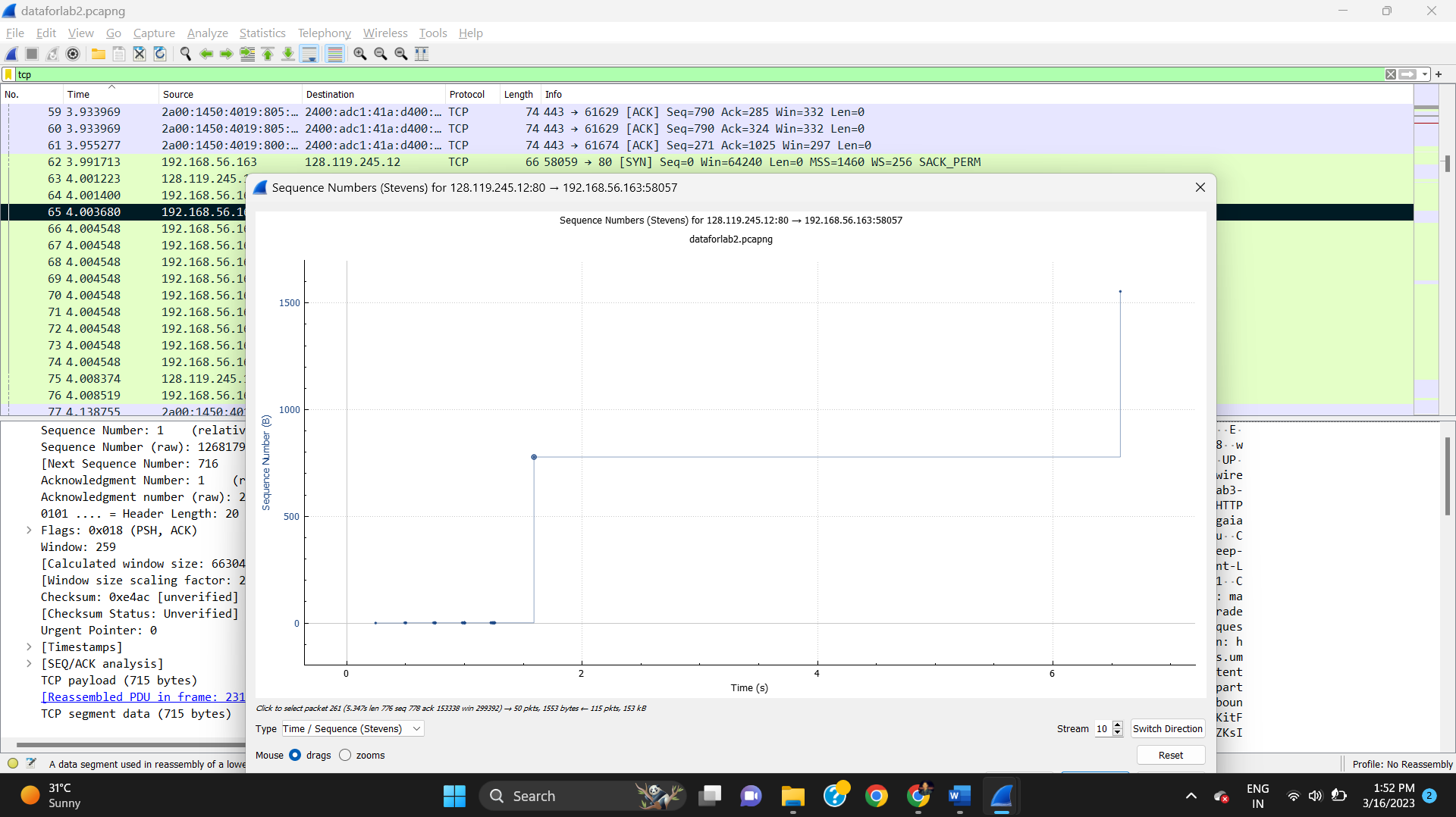
Seq 6 = 2194

Still each difference of content length is still around 1412 for same segment packages.

## 10)

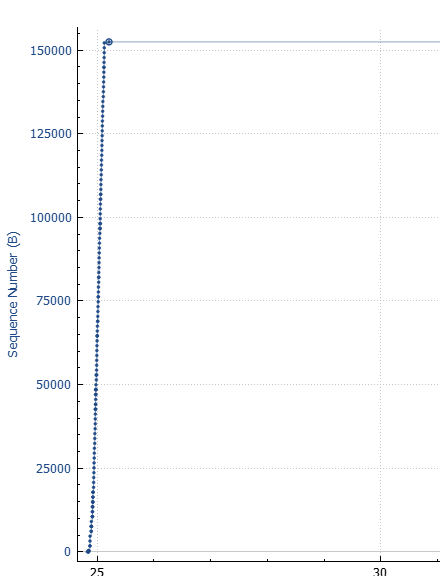
For the minimum buffer space, we can see that the space listed is for 65535 (also previously discussed why). We dont undergo throttling since before reaching full capacity, we get our acknowledgements.

## 11)



No retransmission as the sequence graph has no dots stacked vertically so this means no re transmission when sending data.

Default file



## 12)

The receiver tends to ack around 432 bits first. We do note that the ack dramatically increases and decreases as well. The increase is notable when there are more than 1 acknowledgment at a time as we note.

## 13)

FIN ACK comes at 153338, so these bytes are acknowledged.

Time of ending was 10.329611. time beginning was 4.001400

This means throughput is bits sent= 153338 divided by

Finack 10.329611 - first sequence send 4.001400 ==6.33ms so 0.00633

153338/0.00633 🡺24244012 bits per second.

24.244012 Megabits so /8 to get MegaBytes so 3.028 MBps.

So 3.028 MBps is my throughput.