# WEEK 5

NAME : SUNDEEP A
ROLL NO : 48

SRN : PES1UG20CS445
SECTION: H

## Step 1: UDP and DNS

Q1) Open Wireshark and set up our privacy filter so that you display only DNS traffic to or from your computer (Filter: dns && ip.addr==<your IP address>).

```
dns && ip.addr == 192.168.100.137
```

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 4 | 0.000479589 | 192.168.100.137 | 192.168.100.2 | DNS | 81 | Standard query 0xe696 A www.pluralsight.com |
| 9 | 0.246002138 | 192.168.100.2 | 192.168.100.137 | DNS | 165 | Standard query response 0xe696 A www.pluralsight.com CNAME ww… |

```
Frame 4: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.100.137, Dst: 192.168.100.2
User Datagram Protocol, Src Port: 46488, Dst Port: 53
    Source Port: 46488
    Destination Port: 53
    Length: 45
    Checksum: 0x4a1b incorrect, should be 0xfffc (maybe caused by "UDP checksum offload"?)
    [Checksum Status: Bad]
    [Stream index: 3]
    [Timestamps]
Domain Name System (query)
```

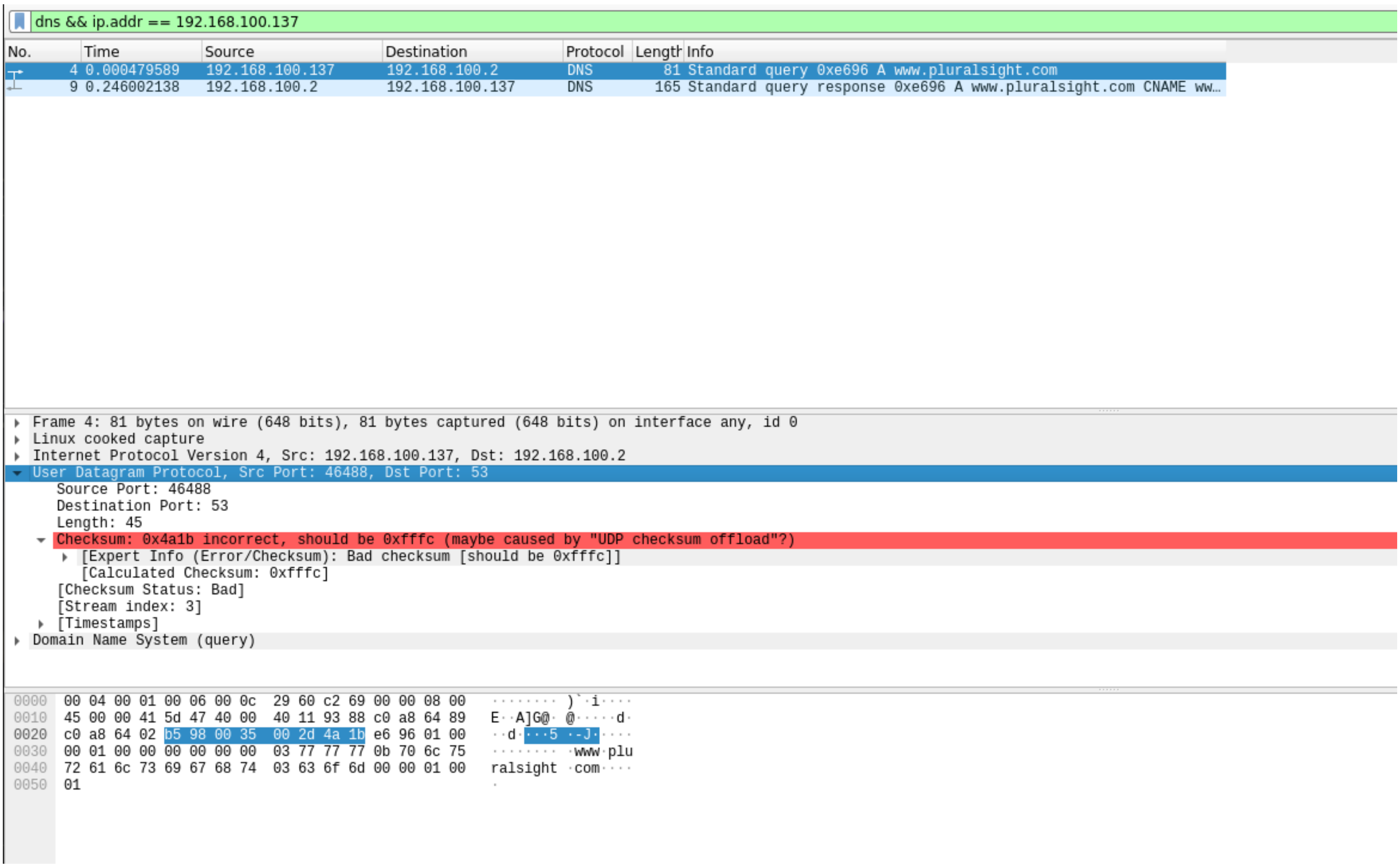
Q2) Use dig to generate a DNS query to lookup the domain name

```
sundeep@sundeep:~$ dig www.pluralsight.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.pluralsight.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9404
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.pluralsight.com.           IN      A

;; ANSWER SECTION:
www.pluralsight.com.    5       IN      CNAME   www.pluralsight.com.cdn.cloudflare.net.
www.pluralsight.com.cdn.cloudflare.net. 4 IN A  104.19.161.127
www.pluralsight.com.cdn.cloudflare.net. 4 IN A  104.19.162.127

;; Query time: 252 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Tue Apr 19 22:41:32 IST 2022
;; MSG SIZE  rcvd: 132
```

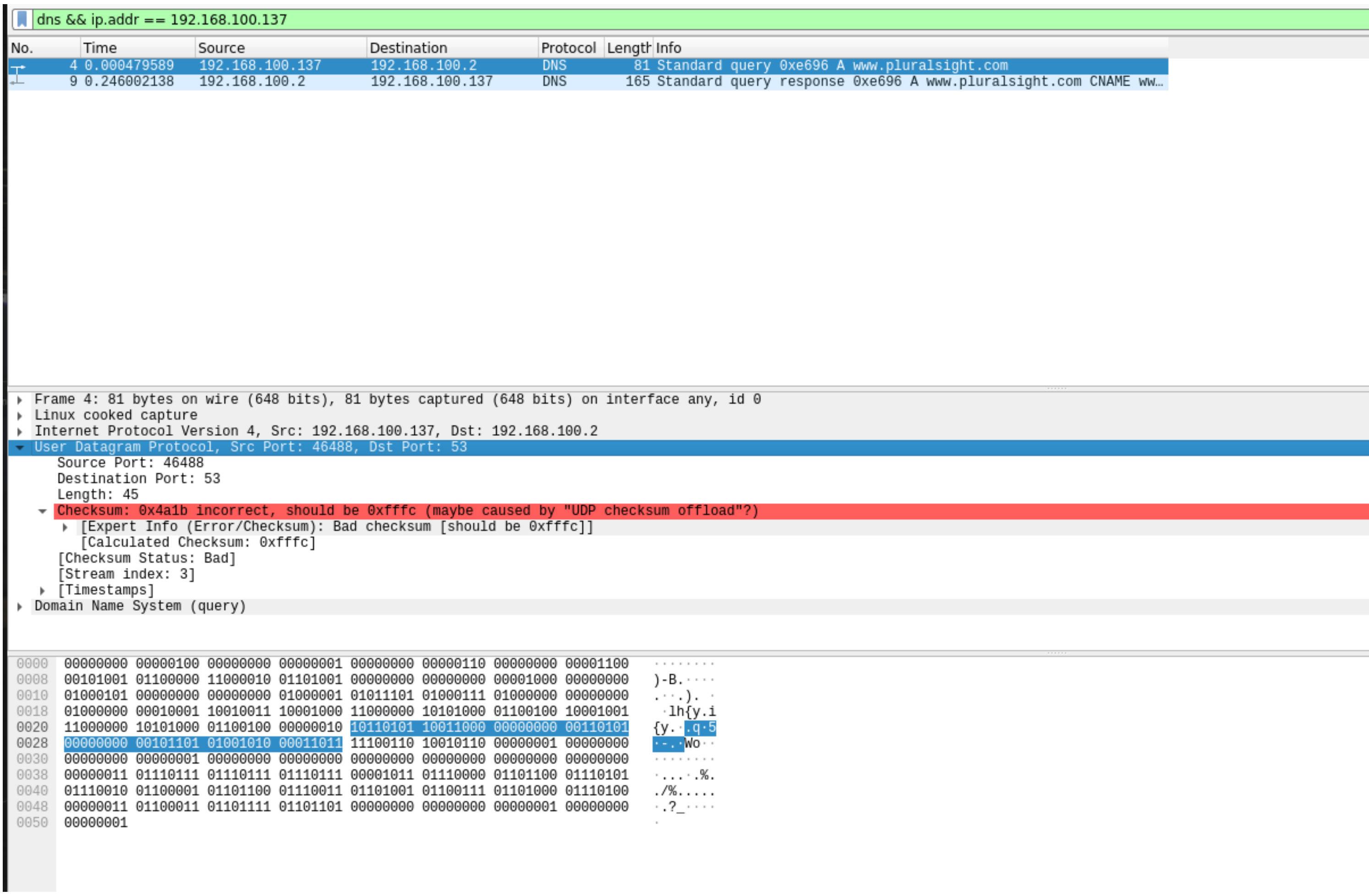
Q3) Before you look at the packets in Wireshark, think for a minute about what you expect to see as the UDP segment headers. What can you reasonably predict, and what could you figure out if you had some time and a calculator handy? Use your knowledge of UDP to inform your predictions.

**I expect to see Source-port and Destination-port numbers, Length, Check-sum.**

Q4) Take a look at the query packet on Wireshark. You'll see a bunch of bytes (70-75 bytes) listed as the actual packet contents in the bStom Wireshark window. The bytes at offsets up to number 33-34 are generated by the lower-level prSocols.You will also see Wireshark interpret the header contents. Match up the bytes in the packet contents window with each field of the UDP header. Were your predictions correct?




In bits:



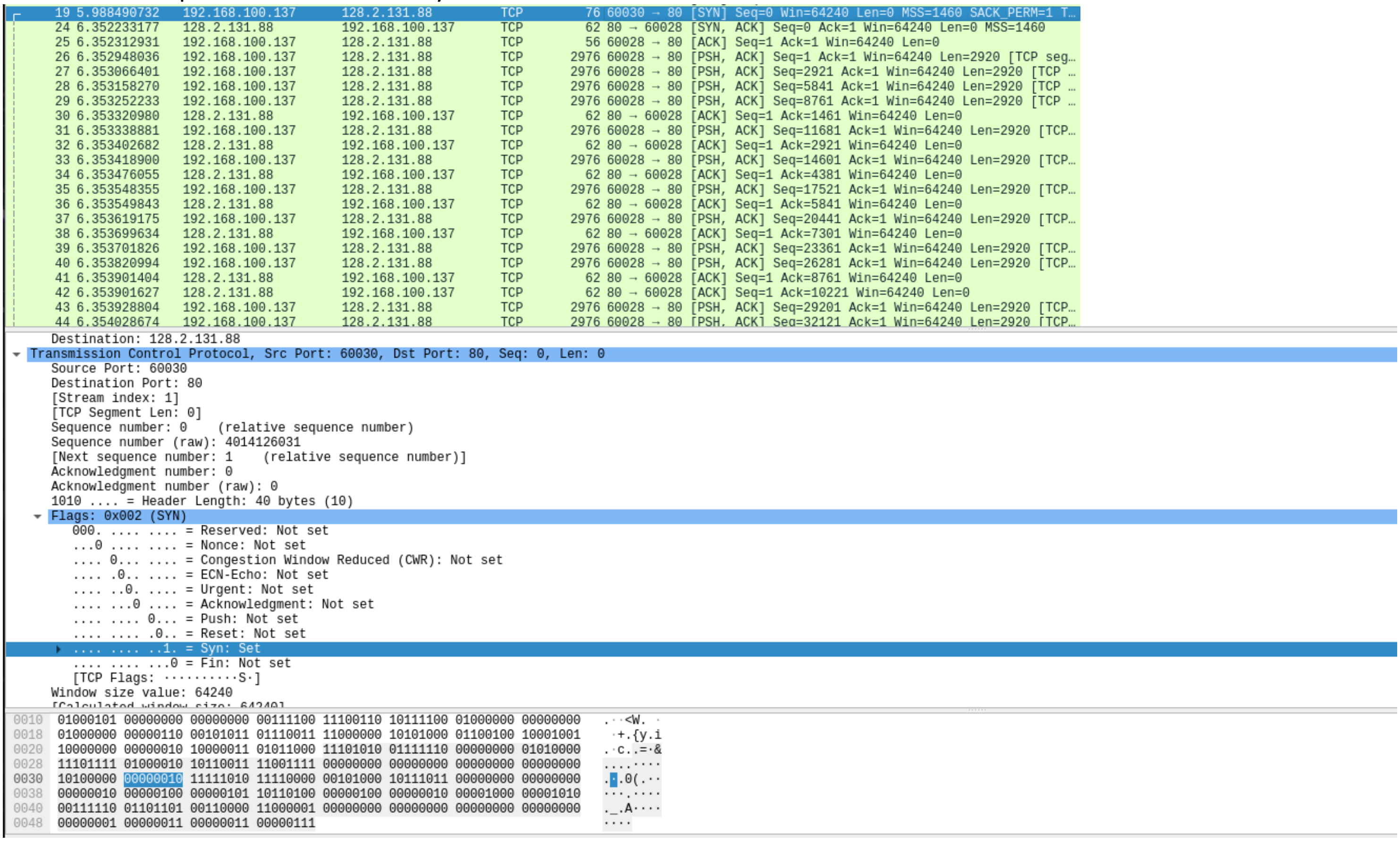

Q5) Continue to examine the DNS request packet. Which fields does the UDP checksum cover? Wireshark probably shows the UDP checksum as “Validation Disabled”. Why is that?

**Due to the prevalence of offloading in modern hardware and operating systems**

# Step 2: TCP

Q11) Take a screenshot showing the three-way handshake.

Here the first 3 packets show the 3 way handshake..




Q12) What is the IP address and TCP port number used by your computer (client) to transfer the file? What is the IP address of the server? On what port number is it sending and receiving TCP segments for this transfer of the file?

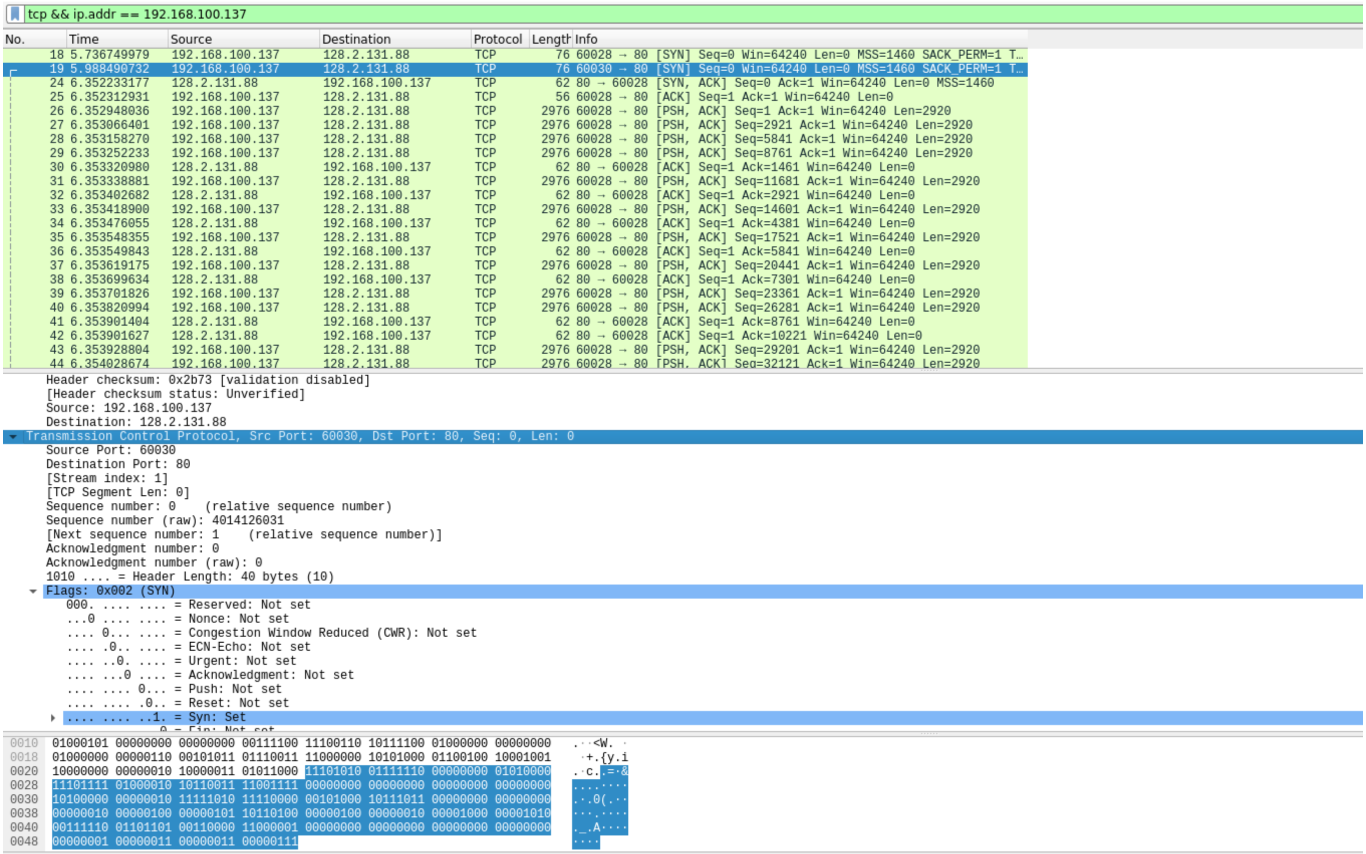
Client:

IP : 192.168.100.137 port:60030

Server:

IP: 128.2.131.88 port: 80

Q13) After disabling HTTP protocol in the Wireshark:



## Step 2b: TCP Basics

Q14) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection?

**Relative Sequence Number:0**

**Raw Sequence number:4014126031**

What element of the segment identifies it as a SYN segment?

**From the above figure, we can see that the SYN bit is set to 1 in the flags section .So, it can be identified as a SYN segment.**

Q15) What is the sequence number of the SYNACK segment sent by the server in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did the server determine that value? What element in the segment identifies it as a SYNACK segment?

**From the below figure we can see that,**

sequence number of the SYNACK segment sent by the server in reply to the SYN = 0

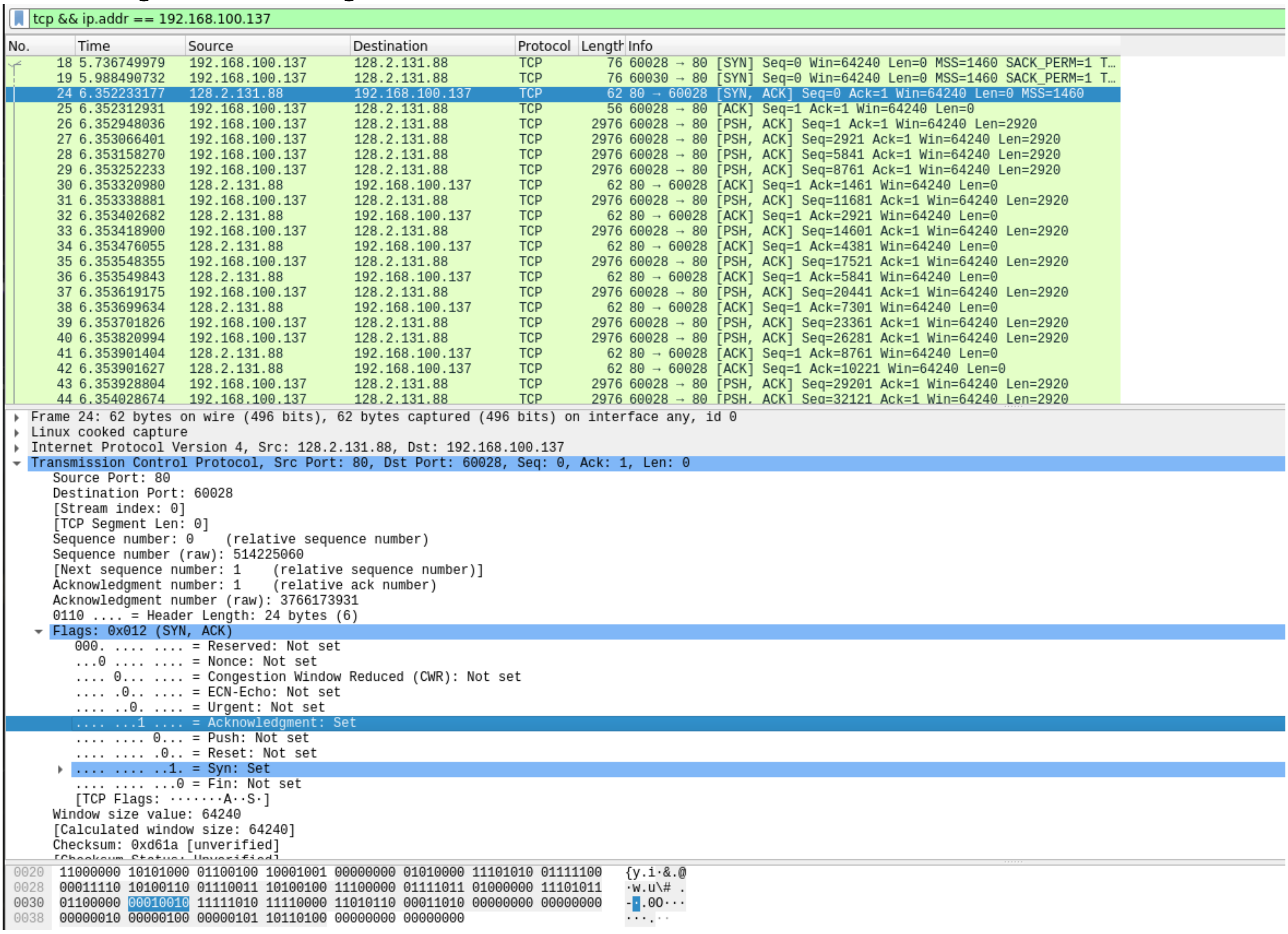
value of the Acknowledgement field in the SYNACK segment = 1

How did the server determine that value?

○ The server adds 1 to the initial sequence number of SYN segment form the client computer. For this case, the initial sequence number of SYN segment from the client computer is 0, thus the value of the ACKnowledgement field in the SYNACK segment is 1.

What element in the segment identifies it as a SYNACK segment?

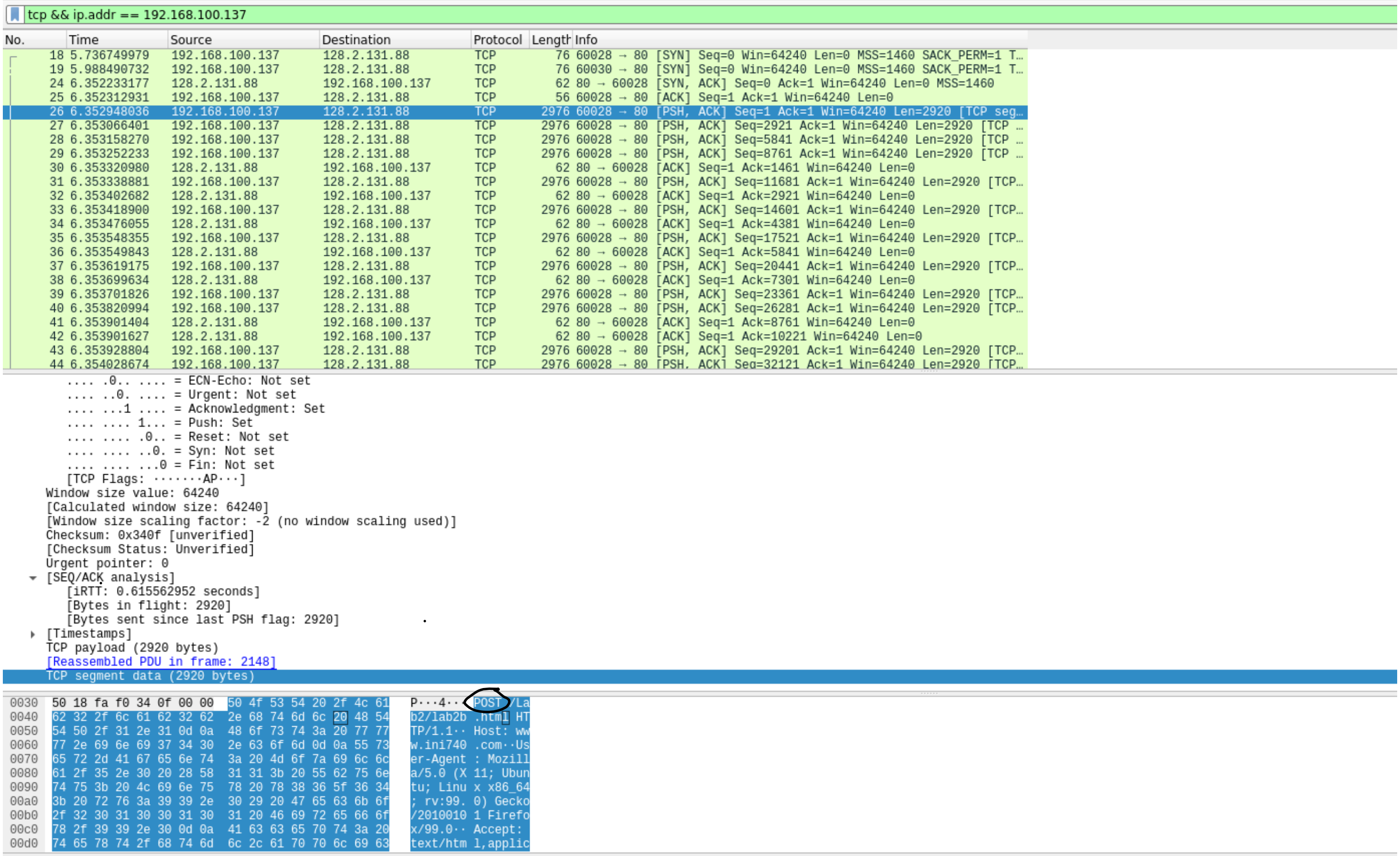
○ A segment will be identified as a SYNACK segment if both SYN flag and Acknowledgement in the segment are set to 1.



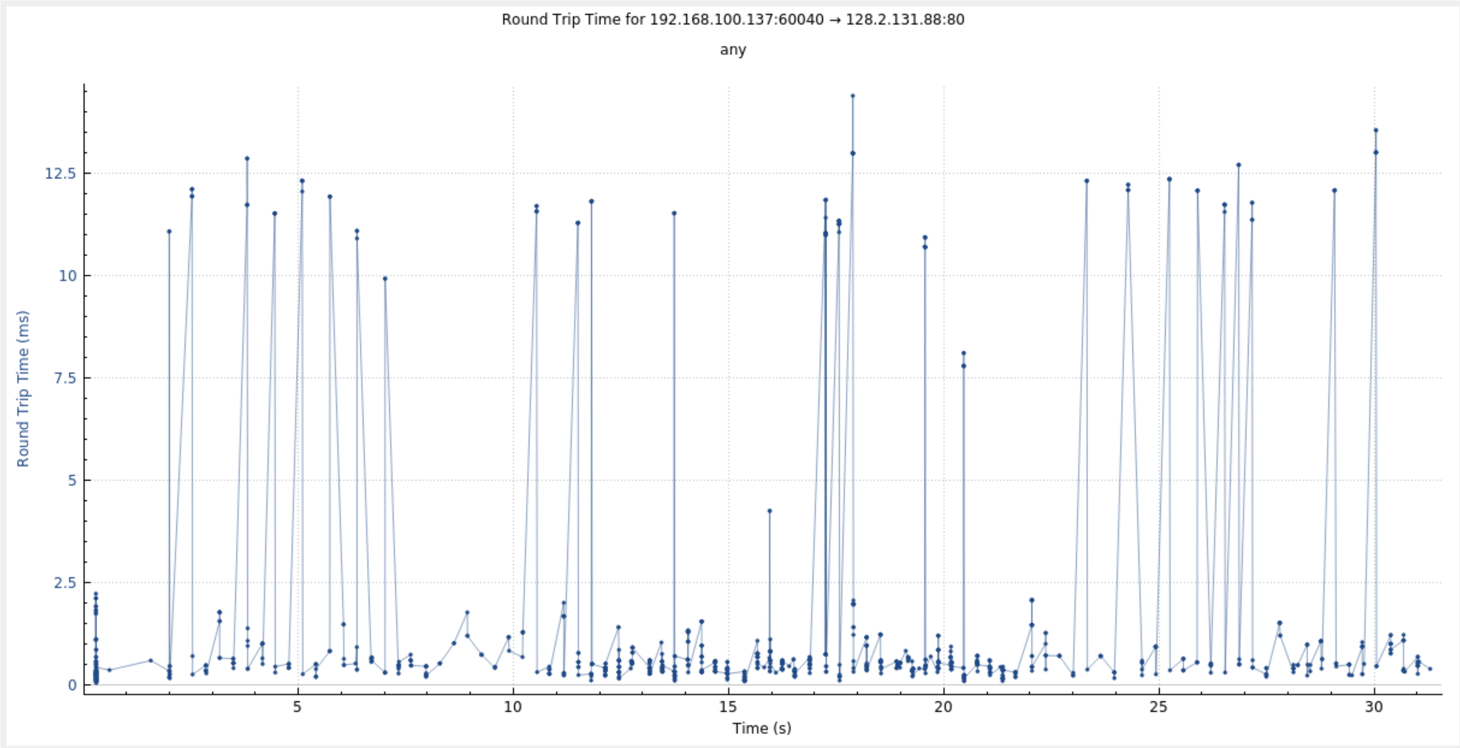
Q16) What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.

**In the below figure we can see that the data field starts with the word " POST".**

The sequence number of the TCP segment containing the HTTP POST is seq = 1




Q17) Consider the TCP segment containing the HTTP POST as the first segment in the non-overhead part of the TCP connection. For the segments which follow, put together a table with one row per segment (and columns for whatever data you think is useful) until you have enough segments to calculate four SampleRTT values according to the RTT estimation techniques discussed in class. Calculate what those SampleRTT values are, as well as the EstimatedRTT after each Sample is collected. Discuss this calculation, including what your initial EstimatedRTT was, your choice of parameters, and any segments that weren't used in the calculation. Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the server. Then select: Statistics → TCP Stream Graph → Round Trip Time Graph.

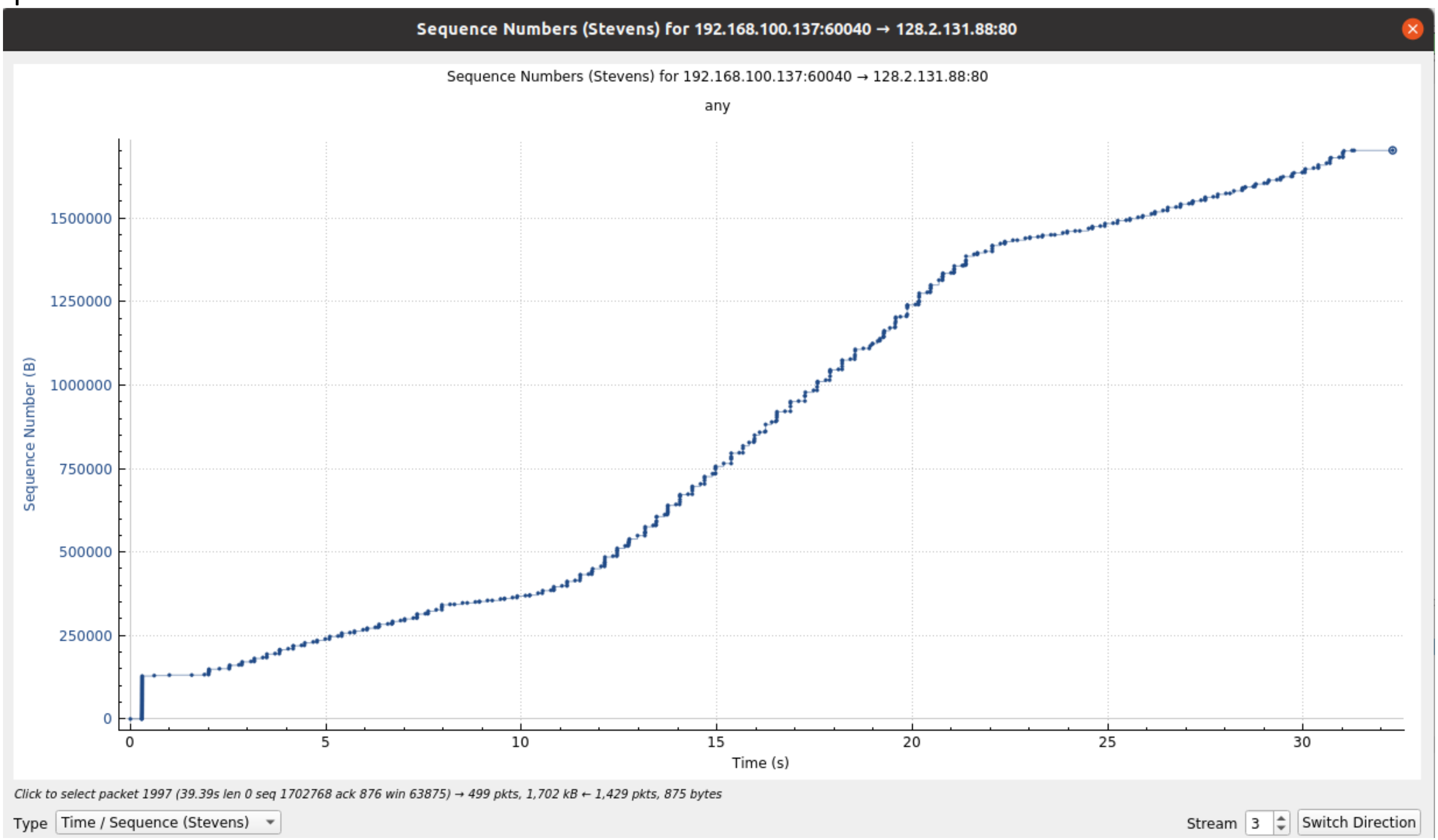



Q18) What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

```
▾ Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▸ .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A··S·]
  Window size value: 64240
  [Calculated window size: 64240]
```

The minimum amount of available buffer space advertised at the received for the entire trace is indicated in the first ACK from the server, its value is 64240 bytes (shown in above figure).

Q19) Are there any retransmitted segments? What did you check for (in the trace) to answer this question?



As the sequence number-time graph is monotonically increasing from the above graph,we can conclude that no packet is retransmitted.
OR

tcp.analysis.retransmission

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|

Q20) How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is delayed ACKing segments? Explain how or why not

| | | | | | | |
|---|---|---|---|---|---|---|
| 50 | 6.354191561 | 128.2.131.88 | 192.168.100.137 | TCP | 62 | 80 → 60028 [ACK] Seq=1 Ack=16061 Win=64240 Len=0 |
| 51 | 6.354191657 | 128.2.131.88 | 192.168.100.137 | TCP | 62 | 80 → 60028 [ACK] Seq=1 Ack=17521 Win=64240 Len=0 |
| 52 | 6.354302366 | 128.2.131.88 | 192.168.100.137 | TCP | 62 | 80 → 60028 [ACK] Seq=1 Ack=18981 Win=64240 Len=0 |
| 53 | 6.354302477 | 128.2.131.88 | 192.168.100.137 | TCP | 62 | 80 → 60028 [ACK] Seq=1 Ack=20441 Win=64240 Len=0 |

The difference between the acknowledged sequence numbers of two consecutive ACKs indicates the data received by the server between these two ACKs.
The receiver is ACKing every other segment. For example, segment of Number. 51 acknowledged data with 18981-17521=1460 bytes.

Q21) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value
The file on the hard drive is 17,01,981 bytes, and the download time is 51.057952 (last TCP segment) - 48.441203 (last ACK) = 2.616749 second. Therefore, the throughput for the TCP connection is computed as 17,01,981/2.616749=650418.133340 bytes/second.

## Step 2c: Statistics

Q22) What is the most common TCP packet length range? What is the second most common TCP packet length range? Why is the ratio of TCP packets of length < 40 bytes equal to zero? Describe what actions you took to get answers to these questions from Wireshark

Wireshark · Packet Lengths · any

| Topic / Item | Count | Average | Min val | Max val | Rate (ms) | Percent | Burst rate | Burst start |
|---|---|---|---|---|---|---|---|---|
| Packet Lengths | 2255 | 827.28 | 56 | 14656 | 0.0501 | 100% | 1.3700 | 6.352 |
| 0-19 | 0 | - | - | - | 0.0000 | 0.00% | - | - |
| 20-39 | 0 | - | - | - | 0.0000 | 0.00% | - | - |
| 40-79 | 1675 | 62.17 | 56 | 78 | 0.0372 | 74.28% | 0.9400 | 6.352 |
| 80-159 | 76 | 93.89 | 83 | 159 | 0.0017 | 3.37% | 0.0800 | 42.000 |
| 160-319 | 13 | 175.92 | 162 | 194 | 0.0003 | 0.58% | 0.0400 | 42.815 |
| 320-639 | 2 | 440.00 | 414 | 466 | 0.0000 | 0.09% | 0.0100 | 41.691 |
| 640-1279 | 1 | 931.00 | 931 | 931 | 0.0000 | 0.04% | 0.0100 | 40.256 |
| 1280-2559 | 199 | 1514.08 | 1292 | 1516 | 0.0044 | 8.82% | 0.1100 | 8.445 |
| 2560-5119 | 176 | 3442.86 | 2976 | 4843 | 0.0039 | 7.80% | 0.4200 | 6.353 |
| 5120 and greater | 113 | 7459.36 | 5896 | 14656 | 0.0025 | 5.01% | 0.0700 | 10.773 |

Display filter: Apply

Copy Save as... Close

the most common TCP packet length range: 40-79
is the second most common TCP packet length range : 1280-2559

The header length is 40 bytes in handshaking stage as it consists of 10 headers so as the minimum packet length is 40 bytes without any data in it, the ratio of tcp packets of length <40 is 0.

This information can be obtained by navigating to ‘statistics<packet lenths’ in wireshark menu.

Q23) What average throughput did you use in Mbps? How many packets were captured in the packet capture session? How many bytes in total? Explain your methods.

Average throughput = 1865523/45.046 = 0.041Mbps
Number of packets captured = 2255
Total number of bytes = 1865523

For this information navigate to ‘statistics->capture file properties’

**Statistics**

| Measurement | Captured | Displayed | Marked |
|---|---|---|---|
| Packets | 2255 | 2116 (93.8%) | — |
| Time span, s | 45.046 | 39.309 | — |
| Average pps | 50.1 | 53.8 | — |
| Average packet size, B | 827 | 875 | — |
| Bytes | 1865523 | 1852540 (99.3%) | 0 |
| Average bytes/s | 41 k | 47 k | — |
| Average bits/s | 331 k | 377 k | — |

Q24) A conversation represents a traffic between two hosts. With which remote host did your local host converse the most (in bytes)? How many packets were sent from your host? How many packets were sent from the remote host?

Wireshark · Conversations · any

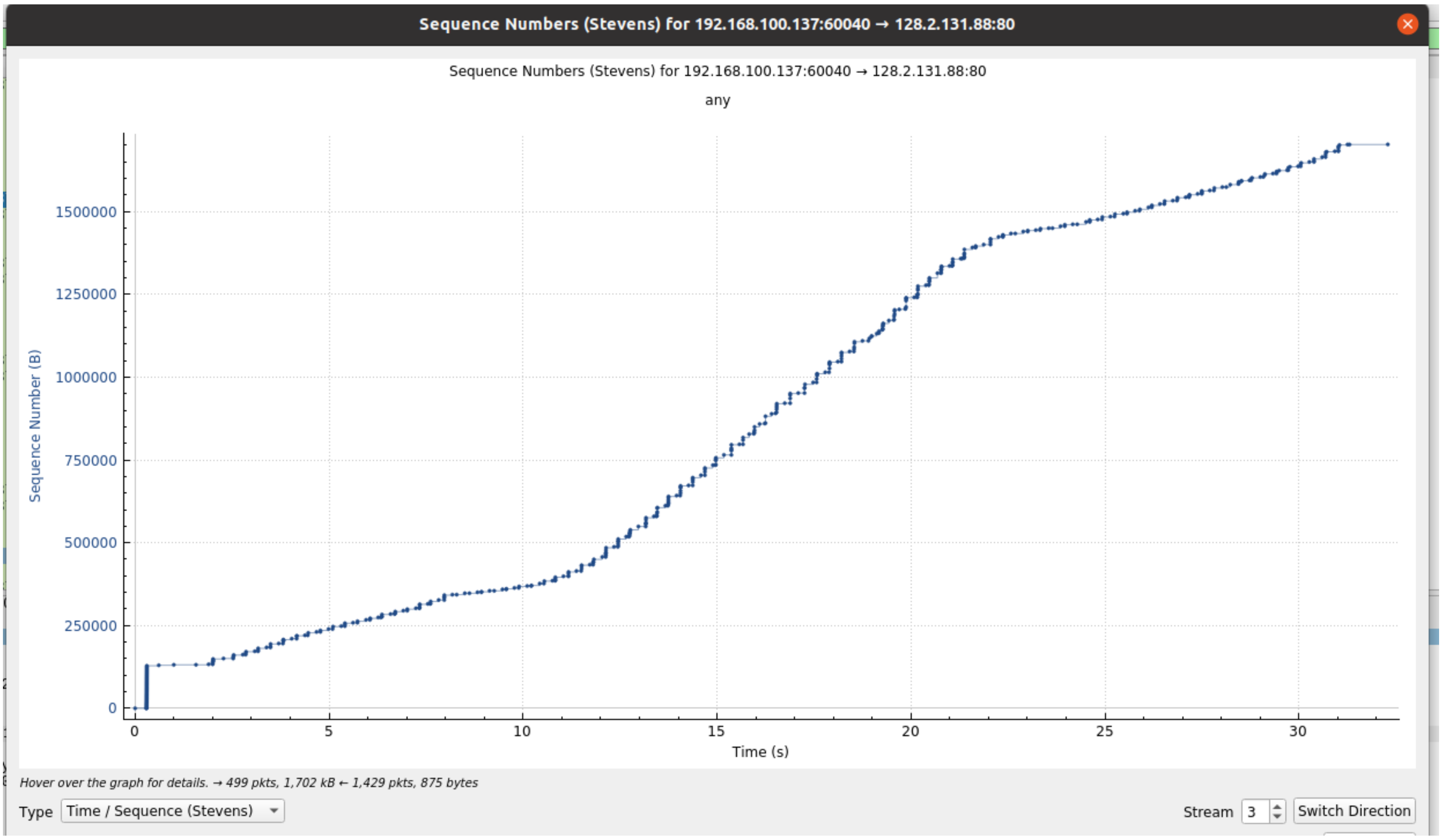
Ethernet | IPv4 · 6 | IPv6 | TCP · 5 | UDP · 7

| Address A | Address B | Packets | Bytes | Packets A → B | Bytes A → B | Packets B → A | Bytes B → A | Rel Start | Duration | Bits/s A → B | Bits/s B → A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 99.86.20.72 | 192.168.100.137 | 8 | 472 | 4 | 248 | 4 | 224 | 5.886420 | 30.7198 | 64 | 58 |
| 127.0.0.1 | 127.0.0.53 | 12 | 2,006 | 6 | 542 | 6 | 1,464 | 0.000000 | 39.5379 | 109 | 296 |
| 128.2.131.88 | 192.168.100.137 | 1,935 | 1,820 k | 1,432 | 89 k | 503 | 1,730 k | 7.096355 | 32.2918 | 22 k | 428 k |
| 142.250.71.10 | 192.168.100.137 | 21 | 8,094 | 10 | 6,536 | 11 | 1,558 | 0.763596 | 0.2766 | 189 k | 45 k |
| 142.250.77.138 | 192.168.100.137 | 21 | 14 k | 10 | 12 k | 11 | 1,995 | 0.063412 | 1.6465 | 62 k | 9,693 |
| 192.168.100.2 | 192.168.100.137 | 8 | 1,186 | 4 | 870 | 4 | 316 | 0.000261 | 39.5372 | 176 | 63 |

With 128.2.131.88 my localhost conversed the most.(1730Kbytes were transferred)
503 packets were sent from my localhost
1432 packets were sent from the remote host.

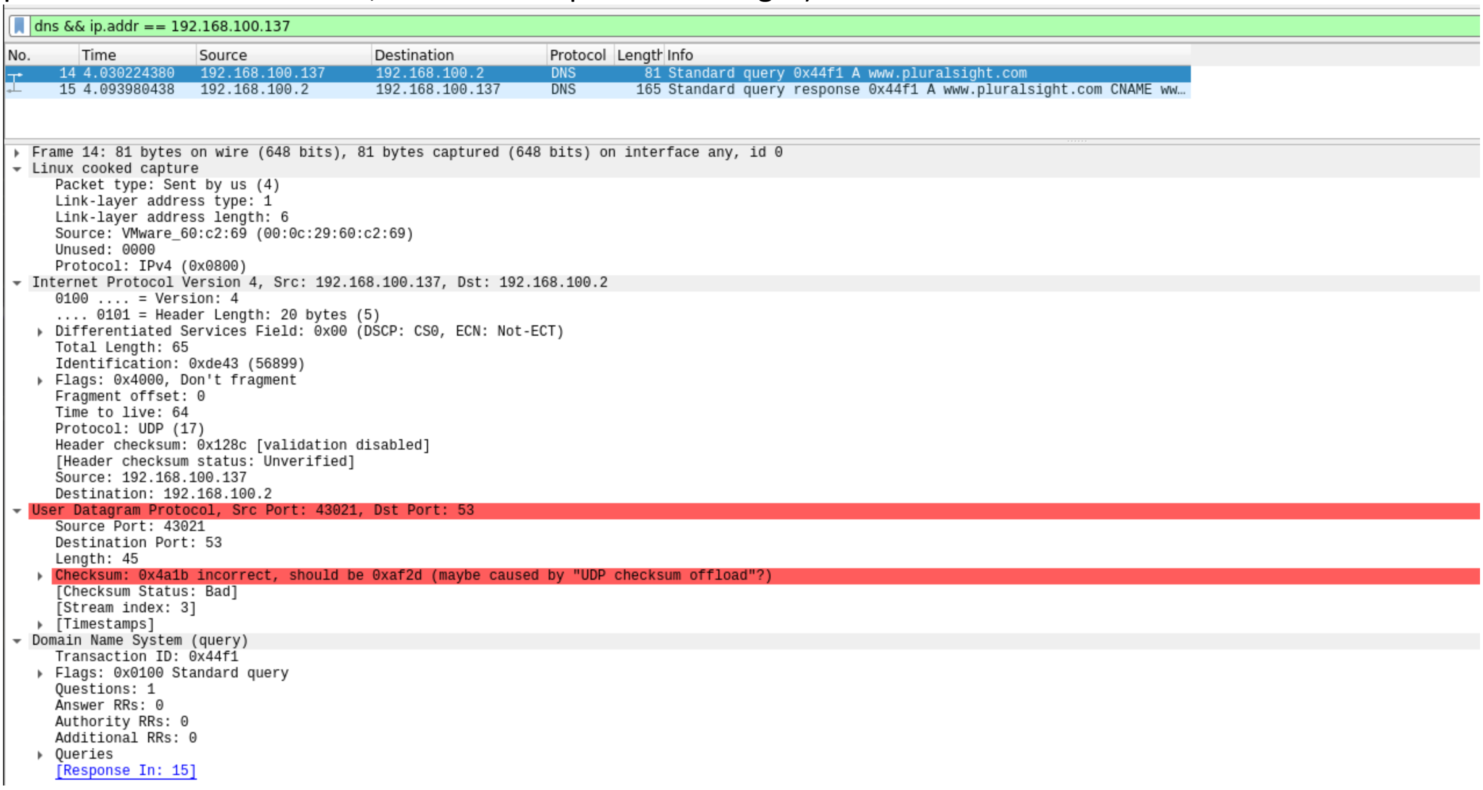
## Step 3: Congestion Control

Q25). Select a TCP segment in the Wireshark’s “listing of captured-packets” window. Then select the menu: Statistics → TCP Stream Graph → Time-Sequence- Graph (Stevens). You should see a plot that looks like the following plot (though the individual plotted values may differ quite a bit).




## Step 4: The Network Layer

Q28) Take a look at the IP section of the DNS query (the packet that was generated when you used dig to request the address of www.pluralsight.com). Match up the header fields with the format we discussed in class (don't just look through Wireshark's display -- instead, match the raw bytes with the pictures we saw in lecture, which I've copied on the right).




Q29) Datagram length: 45
Upper-Layer protocol-IP
Source IP address : 192.168.100.137
Destination IP address : 192.168.100.2

Q31) we discussed the TTL field and determined that we didn't know a good way to set this. What does your OS set this field to?

TTL value = 64

OS = Ubuntu 20.04

## Step 5: ICMP

Q33)

```
sundeep@sundeep:~$ traceroute www.cmuj.jp
traceroute to www.cmuj.jp (122.17.163.205), 30 hops max, 60 byte packets
 1  _gateway (192.168.100.2)  0.502 ms  0.576 ms  0.433 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Q32)

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 127.0.0.1 | 127.0.0.53 | DNS | 84 | Standard query 0x35ab A www.cmuj.jp OPT |
| 2 | 0.000054675 | 127.0.0.1 | 127.0.0.53 | DNS | 84 | Standard query 0xcea0 AAAA www.cmuj.jp OPT |
| 3 | 0.000280993 | 192.168.100.137 | 192.168.100.2 | DNS | 73 | Standard query 0xb3c7 A www.cmuj.jp |
| 4 | 0.000542686 | 192.168.100.137 | 192.168.100.2 | DNS | 73 | Standard query 0xa38e AAAA www.cmuj.jp |
| 5 | 0.725109521 | 192.168.100.2 | 192.168.100.137 | DNS | 155 | Standard query response 0xb3c7 A www.cmuj.jp CNAME cmuj.jp A … |
| 6 | 0.726083712 | 127.0.0.53 | 127.0.0.1 | DNS | 114 | Standard query response 0x35ab A www.cmuj.jp CNAME cmuj.jp A … |
| 7 | 1.298569536 | 192.168.100.2 | 192.168.100.137 | DNS | 161 | Standard query response 0xa38e AAAA www.cmuj.jp CNAME cmuj.jp… |
| 8 | 1.299283086 | 192.168.100.137 | 192.168.100.2 | DNS | 69 | Standard query 0x4644 AAAA cmuj.jp |
| 9 | 1.307883279 | 192.168.100.2 | 192.168.100.137 | DNS | 69 | Standard query response 0x4644 AAAA cmuj.jp |
| 10 | 1.308342982 | 127.0.0.53 | 127.0.0.1 | DNS | 98 | Standard query response 0xcea0 AAAA www.cmuj.jp CNAME cmuj.jp… |
| 11 | 1.308963909 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 44802 → 33434 Len=32 [UDP CHECKSUM INCORRECT] |
| 12 | 1.309271081 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 49237 → 33435 Len=32 [UDP CHECKSUM INCORRECT] |
| 13 | 1.309444820 | 192.168.100.2 | 192.168.100.137 | ICMP | 104 | Time-to-live exceeded (Time to live exceeded in transit) |
| 14 | 1.309671330 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 40120 → 33436 Len=32 [UDP CHECKSUM INCORRECT] |
| 15 | 1.309828921 | 192.168.100.2 | 192.168.100.137 | ICMP | 104 | Time-to-live exceeded (Time to live exceeded in transit) |
| 16 | 1.309918129 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 57824 → 33437 Len=32 [UDP CHECKSUM INCORRECT] |
| 17 | 1.310087505 | 192.168.100.2 | 192.168.100.137 | ICMP | 104 | Time-to-live exceeded (Time to live exceeded in transit) |
| 18 | 1.310177506 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 57927 → 33438 Len=32 [UDP CHECKSUM INCORRECT] |
| 19 | 1.310358562 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 59798 → 33439 Len=32 [UDP CHECKSUM INCORRECT] |
| 20 | 1.310641675 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 55119 → 33440 Len=32 [UDP CHECKSUM INCORRECT] |
| 21 | 1.311871811 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 59103 → 33441 Len=32 [UDP CHECKSUM INCORRECT] |
| 22 | 1.312161293 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 45384 → 33442 Len=32 [UDP CHECKSUM INCORRECT] |
| 23 | 1.312442937 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 37430 → 33443 Len=32 [UDP CHECKSUM INCORRECT] |
| 24 | 1.312727490 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 49077 → 33444 Len=32 [UDP CHECKSUM INCORRECT] |
| 25 | 1.312992874 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 41266 → 33445 Len=32 [UDP CHECKSUM INCORRECT] |
| 26 | 1.313214126 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 33419 → 33446 Len=32 [UDP CHECKSUM INCORRECT] |
| 27 | 1.313532240 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 39264 → 33447 Len=32 [UDP CHECKSUM INCORRECT] |
| 28 | 1.313762325 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 39824 → 33448 Len=32 [UDP CHECKSUM INCORRECT] |
| 29 | 1.314004682 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 47151 → 33449 Len=32 [UDP CHECKSUM INCORRECT] |
| 30 | 1.314842282 | 127.0.0.1 | 127.0.0.53 | DNS | 99 | Standard query 0xe3ad PTR 2.100.168.192.in-addr.arpa OPT |
| 31 | 1.315572030 | 192.168.100.137 | 192.168.100.2 | DNS | 88 | Standard query 0xb65f PTR 2.100.168.192.in-addr.arpa |
| 32 | 1.355722359 | 192.168.100.2 | 192.168.100.137 | DNS | 88 | Standard query response 0xb65f No such name PTR 2.100.168.192… |
| 33 | 1.356656805 | 127.0.0.53 | 127.0.0.1 | DNS | 121 | Standard query response 0xe3ad PTR 2.100.168.192.in-addr.arpa… |
| 34 | 1.359084540 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 47660 → 33450 Len=32 [UDP CHECKSUM INCORRECT] |
| 35 | 1.359869920 | 192.168.100.137 | 122.17.163.205 | UDP | 76 | 38160 → 33451 Len=32 [UDP CHECKSUM INCORRECT] |

```
Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.53
User Datagram Protocol, Src Port: 35679, Dst Port: 53
Domain Name System (query)
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 00 00 44 f2 5b 40 00  40 11 4a 17 7f 00 00 01   E··D·[@· @·J·····
0020  7f 00 00 35 8b 5f 00 35  00 30 fe 77 35 ab 01 20   ···5·_·5 ·0·w5··
0030  00 01 00 00 00 00 00 01  03 77 77 77 04 63 6d 75   ········ ·www·cmu
0040  6a 02 6a 70 00 00 01 00  01 00 00 29 04 b0 00 00   j·jp···· ···)····
0050  00 00 00 00                                        ····
```

```
No.  Time         Source           Destination      Protocol  Leng  Info
  13 1.309444820  192.168.100.2    192.168.100.137  ICMP      104   Time-to-live exceeded (Time to live exceeded in transit)
  15 1.309828921  192.168.100.2    192.168.100.137  ICMP      104   Time-to-live exceeded (Time to live exceeded in transit)
  17 1.310087505  192.168.100.2    192.168.100.137  ICMP      104   Time-to-live exceeded (Time to live exceeded in transit)
```

```
    Total Length: 88
    Identification: 0x4900 (18688)
  ▾ Flags: 0x0000
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..0. .... .... .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (1)
    Header checksum: 0xa7c8 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.2
    Destination: 192.168.100.137
▾ Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0x384a [correct]
    [Checksum Status: Good]
    Unused: 00000000
  ▾ Internet Protocol Version 4, Src: 192.168.100.137, Dst: 122.17.163.205
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 60
      Identification: 0x8900 (35072)
    ▾ Flags: 0x0000
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
      Fragment offset: 0
    ▸ Time to live: 1
      Protocol: UDP (17)
      Header checksum: 0xeda0 [validation disabled]
      [Header checksum status: Unverified]
      Source: 192.168.100.137
      Destination: 122.17.163.205
  ▾ User Datagram Protocol, Src Port: 44802, Dst Port: 33434
      Source Port: 44802
      Destination Port: 33434
      Length: 40
    ▾ Checksum: 0x95eb [correct]
        [Calculated Checksum: 0x95eb]
      [Checksum Status: Good]
      [Stream index: 4]
```

Q35) What are the transmitted segments like? Describe the important features of the segments you observe. In particular, examine the destination port field. What characteristics do you observe about this port number and why would it be chosen so?

Transmitted segments are of type ICMP which are mainly used to carry error messages Commonly used ICMP types are echo request and echo reply (used for ping) and time to live exceeded in transit (used for traceroute).

The destination  port number is 33434. And The dest port increases by one in every sent packet, indicating that traceroute tries to reach the server in multiple ports, as seen here:

```
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 192.168.100.137, Dst: 122.17.163.205
  ▸ User Datagram Protocol, Src Port: 44802, Dst Port: 33434
  ▸ Data (32 bytes)
```

```
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 192.168.100.137, Dst: 122.17.163.205
  ▸ User Datagram Protocol, Src Port: 49237, Dst Port: 33435
  ▸ Data (32 bytes)
```

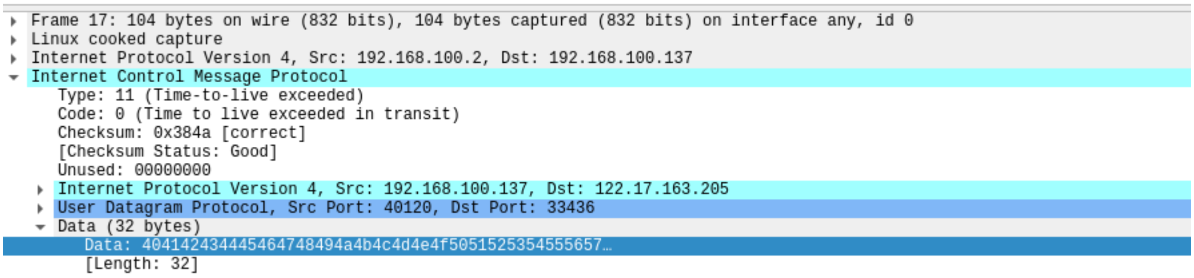
```
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 192.168.100.137, Dst: 122.17.163.205
  ▸ User Datagram Protocol, Src Port: 40120, Dst Port: 33436
  ▸ Data (32 bytes)
```

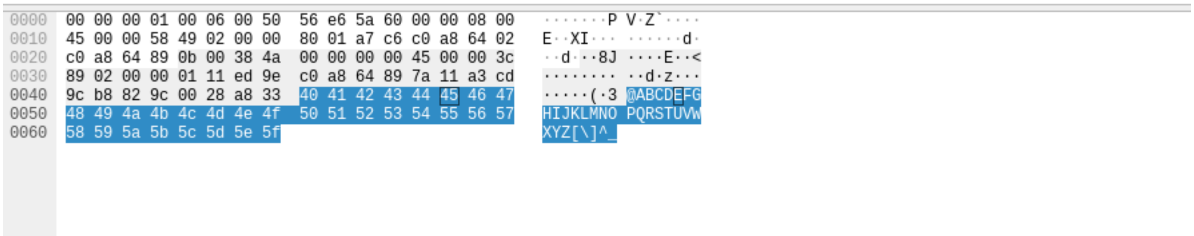
Q36) What about the return packets? What are the values of the various header fields?

```
▾ Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0x384a [correct]
    [Checksum Status: Good]
    Unused: 00000000
```

Q37) The ICMP packets carry some interesting data. What is it? Can you show the relationship to the sent packets?

ICMP packet do contain very intresting data values, The alphabets as seen here:

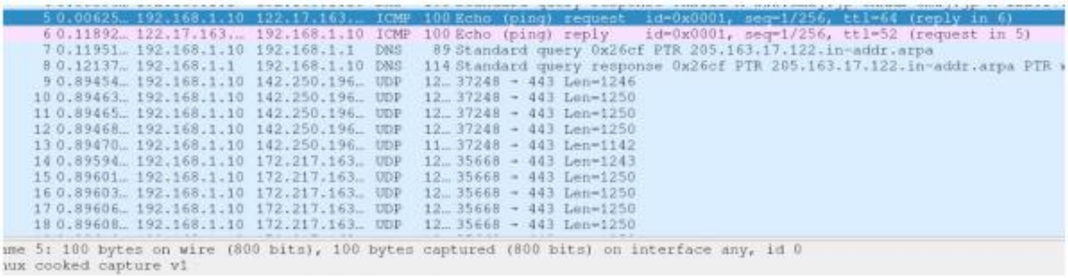



Q38) Lab1 asserted that ping operates in a similar fashion to traceroute. Use Wireshark to show the degree to which this is true. What differences and similarities are there between the network traffic of ping versus traceroute?

Traceroute can’t tell you what happened in the past.

if you experience a slow connection, the Traceroute command that you subsequently issue might not reveal what happened because by that time. The problem that caused the delay may have been fixed and your Traceroute path may not be the same path that the slow connection used.




We can observe the ping request and response followed by a bunch of UDP packets to a fixed port of 443 with random data.