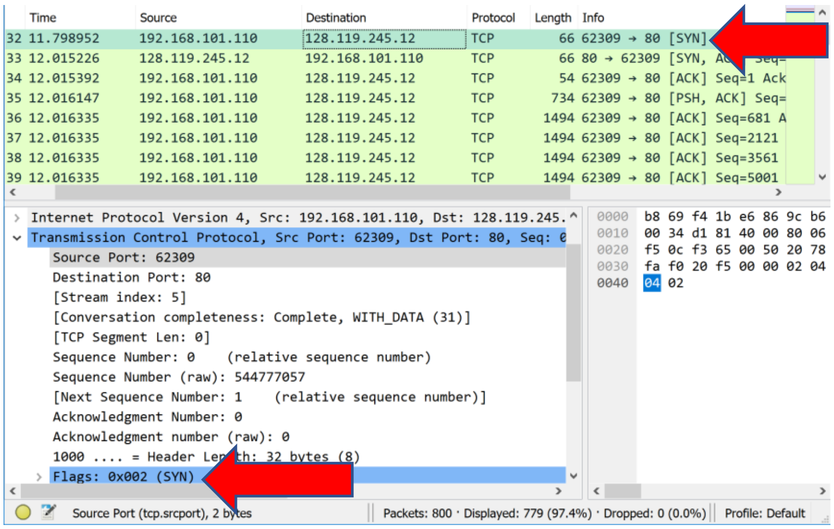
KTHYUS001 – Yusuf Kathrada

Assignment 2

Manual Marked Section

3(b). What is it in this TCP segment that identifies the segment as a SYN segment?

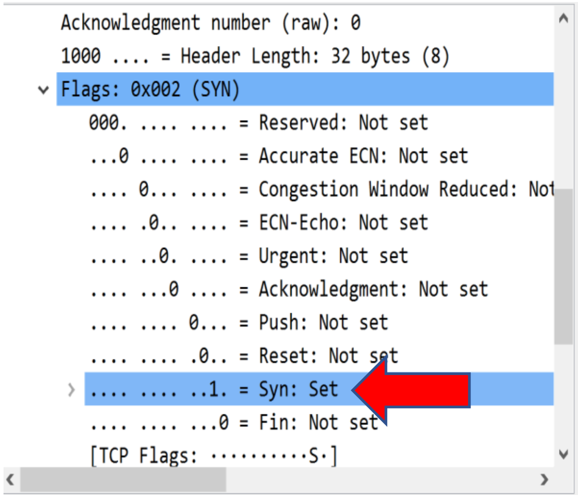


```
   Time        Source            Destination       Protocol  Length  Info
32 11.798952   192.168.101.110   128.119.245.12    TCP          66 62309 → 80 [SYN]
33 12.015226   128.119.245.12    192.168.101.110   TCP          66 80 → 62309 [SYN, AC   Seq=
34 12.015392   192.168.101.110   128.119.245.12    TCP          54 62309 → 80 [ACK] Seq=1 Ack
35 12.016147   192.168.101.110   128.119.245.12    TCP         734 62309 → 80 [PSH, ACK] Seq=
36 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=681 A
37 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=2121
38 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=3561
39 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=5001
```

```
> Internet Protocol Version 4, Src: 192.168.101.110, Dst: 128.119.245.
v Transmission Control Protocol, Src Port: 62309, Dst Port: 80, Seq: 0
    Source Port: 62309
    Destination Port: 80
    [Stream index: 5]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 544777057
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
```

```
0000  b8 69 f4 1b e6 86 9c b6
0010  00 34 d1 81 40 00 80 06
0020  f5 0c f3 65 00 50 20 78
0030  fa f0 20 f5 00 00 02 04
0040  04 02
```

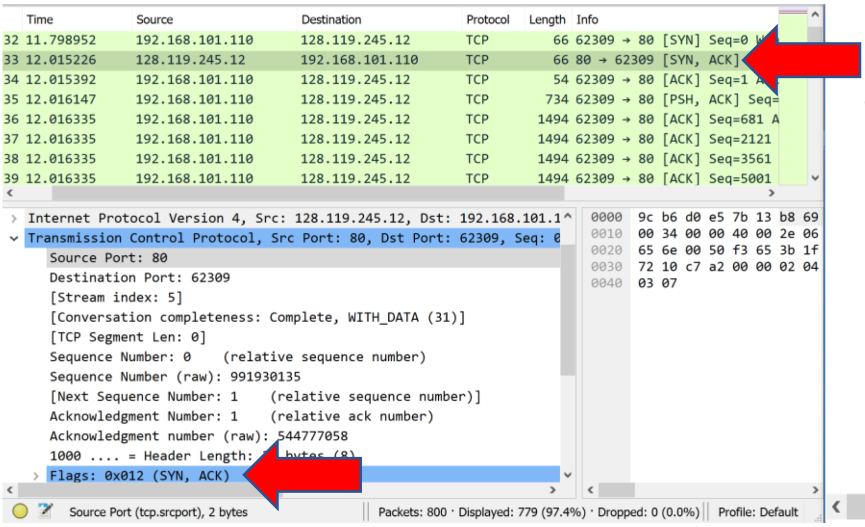
```
Source Port (tcp.srcport), 2 bytes   Packets: 800 · Displayed: 779 (97.4%) · Dropped: 0 (0.0%)   Profile: Default
```



```
  Acknowledgment number (raw): 0
  1000 .... = Header Length: 32 bytes (8)
v Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  > .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ··········S·]
```

The arrows in the images above indicate that in this TCP segment there is a flag which identifies this TCP segment as a TCP SYN segment. This is further confirmed by the Syn: Set to 1 (…. …. ..1. = Syn: Set).

4(b). What is it in the segment that identifies the segment as a SYNACK segment?

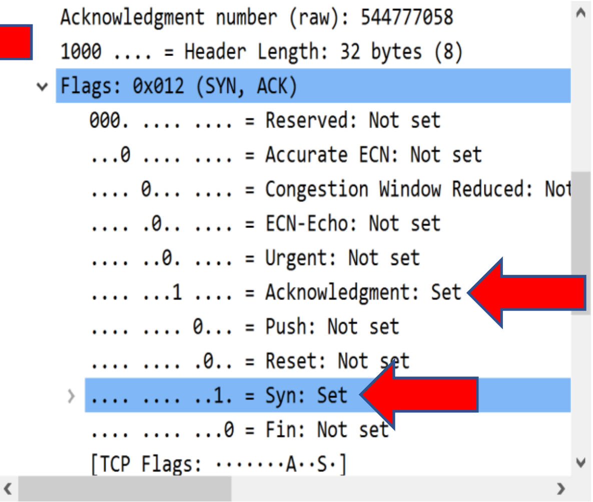


```
   Time        Source            Destination       Protocol  Length  Info
32 11.798952   192.168.101.110   128.119.245.12    TCP          66 62309 → 80 [SYN] Seq=0 W
33 12.015226   128.119.245.12    192.168.101.110   TCP          66 80 → 62309 [SYN, ACK]
34 12.015392   192.168.101.110   128.119.245.12    TCP          54 62309 → 80 [ACK] Seq=1 A
35 12.016147   192.168.101.110   128.119.245.12    TCP         734 62309 → 80 [PSH, ACK] Seq=
36 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=681 A
37 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=2121
38 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=3561
39 12.016335   192.168.101.110   128.119.245.12    TCP        1494 62309 → 80 [ACK] Seq=5001
```

```
> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.101.1
v Transmission Control Protocol, Src Port: 80, Dst Port: 62309, Seq: 0
    Source Port: 80
    Destination Port: 62309
    [Stream index: 5]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 991930135
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 544777058
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
```

```
0000  9c b6 d0 e5 7b 13 b8 69
0010  00 34 00 00 40 00 2e 06
0020  65 6e 00 50 f3 65 3b 1f
0030  72 10 c7 a2 00 00 02 04
0040  03 07
```

```
Source Port (tcp.srcport), 2 bytes   Packets: 800 · Displayed: 779 (97.4%) · Dropped: 0 (0.0%)   Profile: Default
```



```
  Acknowledgment number (raw): 544777058
  1000 .... = Header Length: 32 bytes (8)
v Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not
    .... .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·]
```

The arrows in the images above indicate that in this TCP segment there is flag which identifies this TCP segment as a TCP SYNACK segment. This is further confirmed by the Syn: Set to 1 (…. …. ..1. = Syn: Set) and the Acknowledgement: Set to 1 (…. …1 …. = Acknowledgement: Set).

4(d). What is it in the segment that identifies the segment as a SYNACK segment?

The acknowledgement number (544777058) is 1 more than the sequence number of the TCP SYN segment (544777057) which suggests that the packet was successfully delivered and that the acknowledgment number represents the next byte it is expecting.

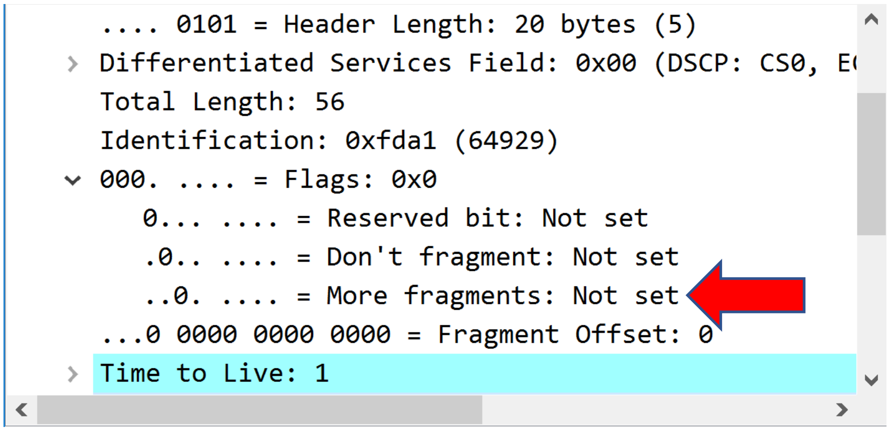
8(b). Does the lack of receiver buffer space ever throttle the sender for these first four data-carrying segments?

No, the sender is not throttled as the first four data-carrying segments all fall within the available buffer space.

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

No, this was checked by looking for any repeating sequence numbers and none were found.

15. Explain how you determined whether or not the datagram has been fragmented.



```
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, E
    Total Length: 56
    Identification: 0xfda1 (64929)
  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: 1
```

The More fragments is equal to 0 (..0. … = More fragments: Not set).

16. Which fields in the IP datagram always change from one datagram to the next within this series of UDP segments sent by your computer destined to 128.119.245.12, via traceroute?

The fields which always change from one datagram to the next is the Identification, Time to Live and Header Checksum.

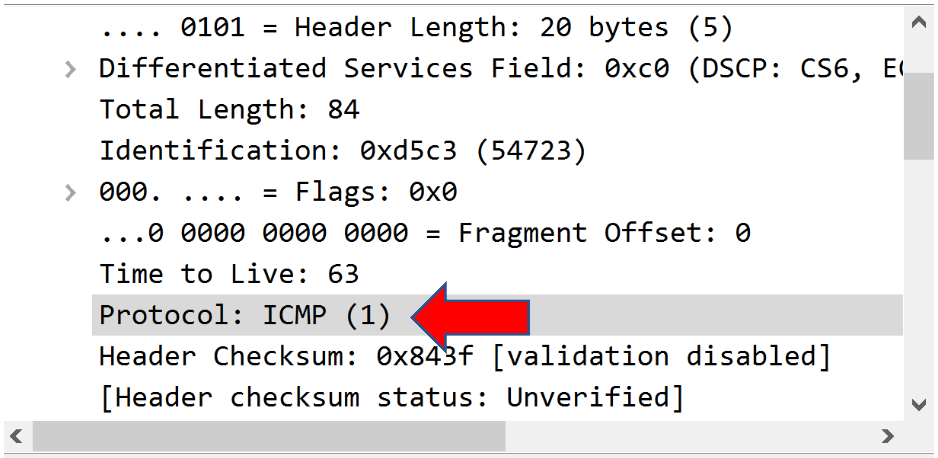
17. Which fields in this sequence of IP datagrams (containing UDP segments) stay constant? Why?

- Version, all versions are equal to 4 as we are using IPv4 across all datagrams.
- Source IP, the source from which we send does not change and thus it stays the same.
- Destination IP, we are sending to same source.
- Header Length, this is due to the fact that across all datagrams they are all ICMP packets.
- Differentiated Services, all the packets are the same (ICMP) and thus they all require the same services.
- Upper Layer Protocol, this is also due to the fact that all the packets are ICMP.

18. Describe the pattern you see in the values in the Identification field of the IP datagrams being sent by your computer.

The identification field whose value is in hexadecimal (and decimal in brackets) increments each time there is a ICMP request.

19. What is the upper layer protocol specified in the IP datagrams returned from the routers?



```
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0xc0 (DSCP: CS6, E
    Total Length: 84
    Identification: 0xd5c3 (54723)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 63
    Protocol: ICMP (1)
    Header Checksum: 0x843f [validation disabled]
    [Header checksum status: Unverified]
```

The upper layer protocol is ICMP.

20. Are the values in the Identification fields (across the sequence of all of ICMP packets from all of the routers) similar in behaviour to your answer to question 18 above?

No, they are not incrementing in constant amounts with the identification of one packet to the next varying substantially.

21. Are the values of the TTL fields similar, across all of ICMP packets from all of the routers?

Yes, the TTL fields across all ICMP packets are all 1.

22. Devise an experiment that will falsify, or validate the hypothesis "The network is slower during higher stages of loadshedding (stage > 4) than lower ones (stage < 4)". That is, write down a procedure about what needs to be done to investigate this to obtain a conclusion 'yes this does happen' or 'no, the performance is the same'. This procedure should be written in such a way that it can be given to a classmate and they'd be able to carry out the experiment you devised.

Hypothesis:

The network is slower during higher stages of loadshedding (stage > 4) than the lower ones (stage < 4)

Experiment:

1. Evaluate when and what stage loadshedding your area receives on the day. Loadshedding can be unpredictable at times so by using an app such as ESP (Eskom Se Push) you are able to view the stages and schedule for the day.

2. Pick a time in which you have no loadshedding (This will be your control test), a time you have stage 5 or higher and a time you have between stage 1 and 4. You will run the next steps for each of the before mentioned periods.

3. Start your web browser and visit: http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.

4. Select the browse button on the form and select a file to upload. This same file must be used for all tests and should preferably be larger than 150k bytes.

5. Start Wireshark and start packet capture.

6. Return to the browser and select the "Upload <filename>" button. You should receive a congratulations message once it has successfully been uploaded.

7. Stop the Wireshark packet capture.

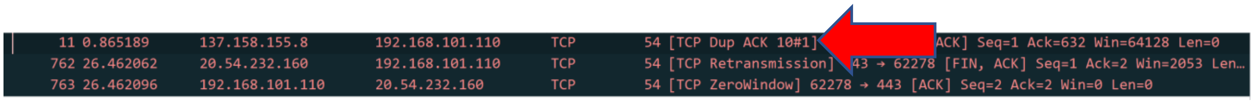
8. Calculate the RTT and tabulate your findings. Repeat steps 3-8 for each period 5 times (i.e., a total of 5x3 = 15 tests) in order to get more conclusive evidence.

9. Calculate the Estimated RTT (using $\alpha = 0.125$) for each period.

10. From these calculations, draw your conclusions based on whether there are noteworthy differences in the RTT found in the three periods.

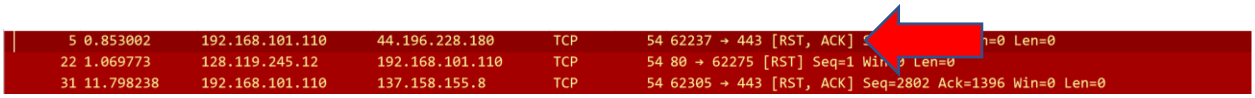
23. Did your network have problems when uploading the Gutenberg file? Try the filters from the table below and report whether there was anything out of the ordinary. For instance, applying the tcp.analysis.flags may show a "TCP Dup ACK", a duplicate ACK received, tcp.analysis.ack_rtt > 0.1 may show packets as well (look at the bottom-right for number of packets and percentage of slow RTT packets).

Yes there were problems when uploading the files.



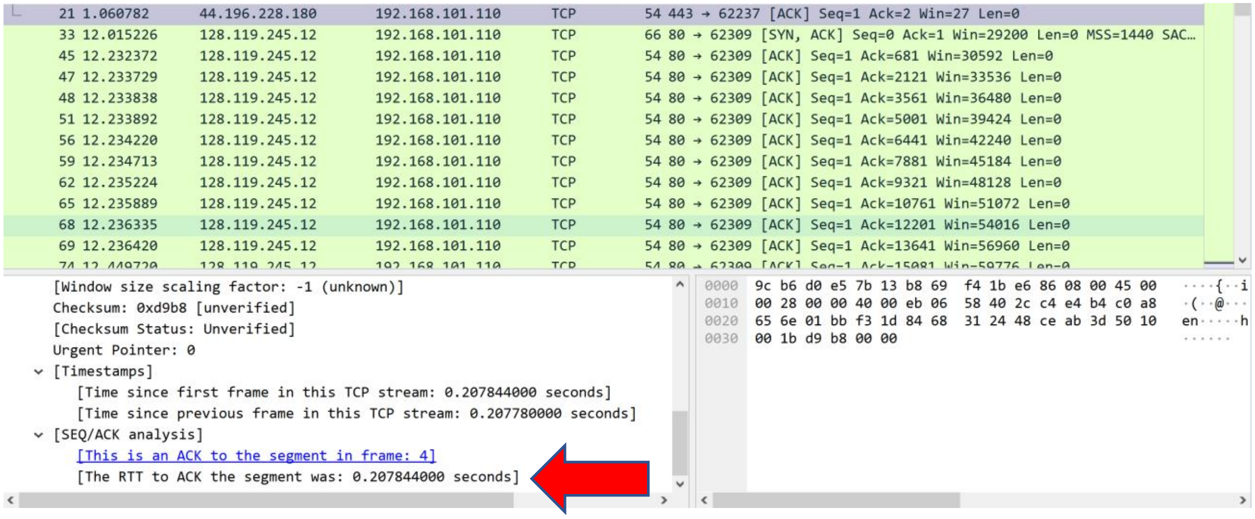
```
 11 0.865189   137.158.155.8    192.168.101.110  TCP   54 [TCP Dup ACK 10#1]              ACK] Seq=1 Ack=632 Win=64128 Len=0
762 26.462062  20.54.232.160    192.168.101.110  TCP   54 [TCP Retransmission]  43 → 62278 [FIN, ACK] Seq=1 Ack=2 Win=2053 Len…
763 26.462096  192.168.101.110  20.54.232.160    TCP   54 [TCP ZeroWindow] 62278 → 443 [ACK] Seq=2 Ack=2 Win=0 Len=0
```

The above image shows that there was a duplicate ACK received. Filter: tcp.analysis.flags && !tcp.analysis.window_update



```
 5 0.853002   192.168.101.110  44.196.228.180   TCP   54 62237 → 443 [RST, ACK] S              n=0 Len=0
22 1.069773   128.119.245.12   192.168.101.110  TCP   54 80 → 62275 [RST] Seq=1 Wi    Len=0
31 11.798238  192.168.101.110  137.158.155.8    TCP   54 62305 → 443 [RST, ACK] Seq=2802 Ack=1396 Win=0 Len=0
```

The above image shows a network failure or timeout that happened mid conversation that caused connection reset. Filter: tcp.flags.reset==1

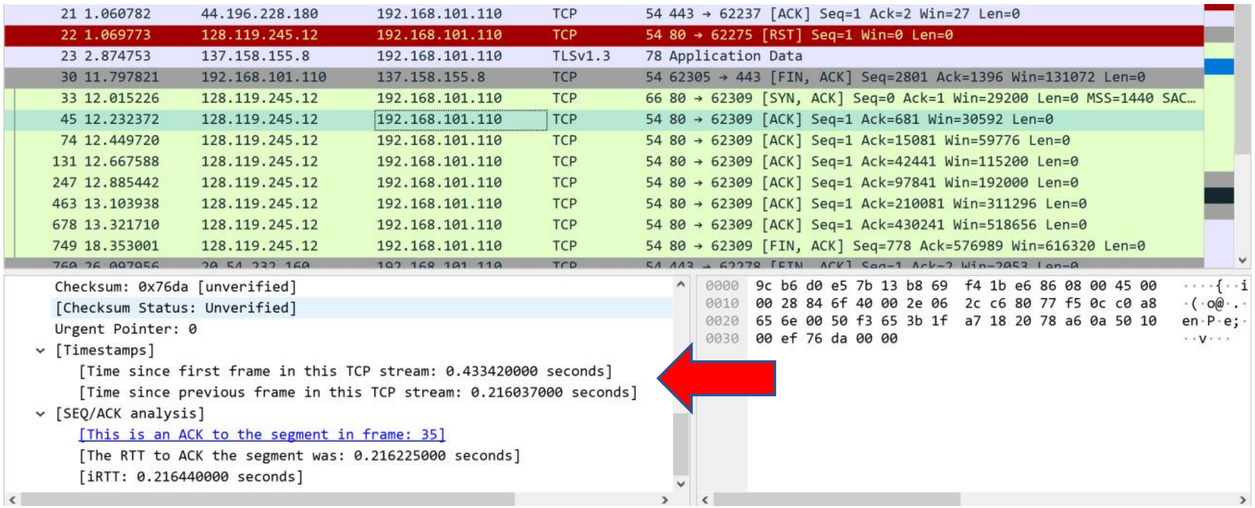


```
21 1.060782   44.196.228.180   192.168.101.110  TCP   54 443 → 62237 [ACK] Seq=1 Ack=2 Win=27 Len=0
33 12.015226  128.119.245.12   192.168.101.110  TCP   66 80 → 62309 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1440 SAC…
45 12.232372  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=681 Win=30592 Len=0
47 12.233729  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=2121 Win=33536 Len=0
48 12.233838  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=3561 Win=36480 Len=0
51 12.233892  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=5001 Win=39424 Len=0
56 12.234220  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=6441 Win=42240 Len=0
59 12.234713  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=7881 Win=45184 Len=0
62 12.235224  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=9321 Win=48128 Len=0
65 12.235889  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=10761 Win=51072 Len=0
68 12.236335  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=12201 Win=54016 Len=0
69 12.236420  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=13641 Win=56960 Len=0
74 12.449720  128.119.245.12   192.168.101.110  TCP   54 80 → 62309 [ACK] Seq=1 Ack=15081 Win=59776 Len=0
```

```
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xd9b8 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ⌄ [Timestamps]
      [Time since first frame in this TCP stream: 0.207844000 seconds]
      [Time since previous frame in this TCP stream: 0.207780000 seconds]
  ⌄ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 4]
      [The RTT to ACK the segment was: 0.207844000 seconds]
```

```
0000  9c b6 d0 e5 7b 13 b8 69  f4 1b e6 86 08 00 45 00   ····{··i
0010  00 28 00 00 40 00 eb 06  58 40 2c c4 e4 b4 c0 a8   ·(··@···
0020  65 6e 01 bb f3 1d 84 68  31 24 48 ce ab 3d 50 10   en·····h
0030  00 1b d9 b8 00 00                                  ······
```

The above image shows slow RTT's which there was many. This may be a sigtn of congestion as it climbs through the trace. Filter: tcp.analysis.ack_rtt > 0.1



```
 21 1.060782    44.196.228.180    192.168.101.110   TCP      54 443 → 62237 [ACK] Seq=1 Ack=2 Win=27 Len=0
 22 1.069773    128.119.245.12    192.168.101.110   TCP      54 80 → 62275 [RST] Seq=1 Win=0 Len=0
 23 2.874753    137.158.155.8     192.168.101.110   TLSv1.3  78 Application Data
 30 11.797821   192.168.101.110   137.158.155.8     TCP      54 62305 → 443 [FIN, ACK] Seq=2801 Ack=1396 Win=131072 Len=0
 33 12.015226   128.119.245.12    192.168.101.110   TCP      66 80 → 62309 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1440 SAC…
 45 12.232372   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=681 Win=30592 Len=0
 74 12.449720   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=15081 Win=59776 Len=0
131 12.667588   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=42441 Win=115200 Len=0
247 12.885442   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=97841 Win=192000 Len=0
463 13.103938   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=210081 Win=311296 Len=0
678 13.321710   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [ACK] Seq=1 Ack=430241 Win=518656 Len=0
749 18.353001   128.119.245.12    192.168.101.110   TCP      54 80 → 62309 [FIN, ACK] Seq=778 Ack=576989 Win=616320 Len=0
760 26.097956   20.54.232.160     192.168.101.110   TCP      54 443 → 62278 [FIN, ACK] Seq=1 Ack=2 Win=2053 Len=0
```

```
    Checksum: 0x76da [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ⌄ [Timestamps]
      [Time since first frame in this TCP stream: 0.433420000 seconds]
      [Time since previous frame in this TCP stream: 0.216037000 seconds]
  ⌄ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 35]
      [The RTT to ACK the segment was: 0.216225000 seconds]
      [iRTT: 0.216440000 seconds]
```

```
0000  9c b6 d0 e5 7b 13 b8 69  f4 1b e6 86 08 00 45 00   ····{··i
0010  00 28 84 6f 40 00 2e 06  2c c6 80 77 f5 0c c0 a8   ·(·o@·.·
0020  65 6e 00 50 f3 65 3b 1f  a7 18 20 78 a6 0a 50 10   en·P·e;·
0030  00 ef 76 da 00 00                                  ··v···
```

The above image shows that there were instances were the server was too busy to respond. Filter: tcp.time_delta > 0.1

There were no evidence that suggests that there was slow DNS times present. Filter: dns.time > 1