Connor Twohey

Project 2

Problem Set 1:

```
▼ Ethernet II, Src: ArrisGro_b7:1f:e0 (90:3e:ab:b7:1f:e0), Dst: Apple_ee:a3:4f (a4:5e:60:ee:a3:4f)
  ▶ Destination: Apple_ee:a3:4f (a4:5e:60:ee:a3:4f)
  ▶ Source: ArrisGro_b7:1f:e0 (90:3e:ab:b7:1f:e0)
    Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 129.107.56.31, Dst: 192.168.1.194
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x7799 (30617)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 244
    Protocol: TCP (6)
    Header checksum: 0x8d8d [validation disabled]
    [Header checksum status: Unverified]
    Source: 129.107.56.31
    Destination: 192.168.1.194
▼ Transmission Control Protocol, Src Port: 443, Dst Port: 51277, Seq: 11243, Ack: 1249, Len: 1448
    Source Port: 443
    Destination Port: 51277
    [Stream index: 4]
    [TCP Segment Len: 1448]
    Sequence number: 11243    (relative sequence number)
    [Next sequence number: 12691    (relative sequence number)]
    Acknowledgment number: 1249    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x010 (ACK)
    Window size value: 5628
    [Calculated window size: 5628]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x9aac [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```

(For all, except #4)

```
▼ TCP Option - No-Operation (NOP)
    Kind: No-Operation (1)
▼ TCP Option - No-Operation (NOP)
    Kind: No-Operation (1)
▼ TCP Option - Timestamps: TSval 101449914, TSecr 4049884412
    Kind: Time Stamp Option (8)
    Length: 10
    Timestamp value: 101449914
    Timestamp echo reply: 4049884412
```

(For #4)

1. IP: 192.168.1.194

Port #: 51277

Both underlined with green
2. TTL = 244 (underlined with red)
3. IPv4 (underlined with blue)
4. See second screenshot
5. Not fragmented (underlined with orange)
6. TCP Segment Length = 1448 (underlined with purple)
7. Sequence Number = Relative Sequence Number = 11243 (underlined with gray)
8. I can use the two above numbers to calculate the ACK for the next packet:
ACK = TCPLength + SeqNum = 1448 + 11243 = 12691
Next packet ACK confirmed as 12691 (right side):

```
43 1.670775    192.168.1.194    129.107.56.31    TCP    66 51277 → 443 [ACK] Seq=1249 Ack=12691 Win=65
```

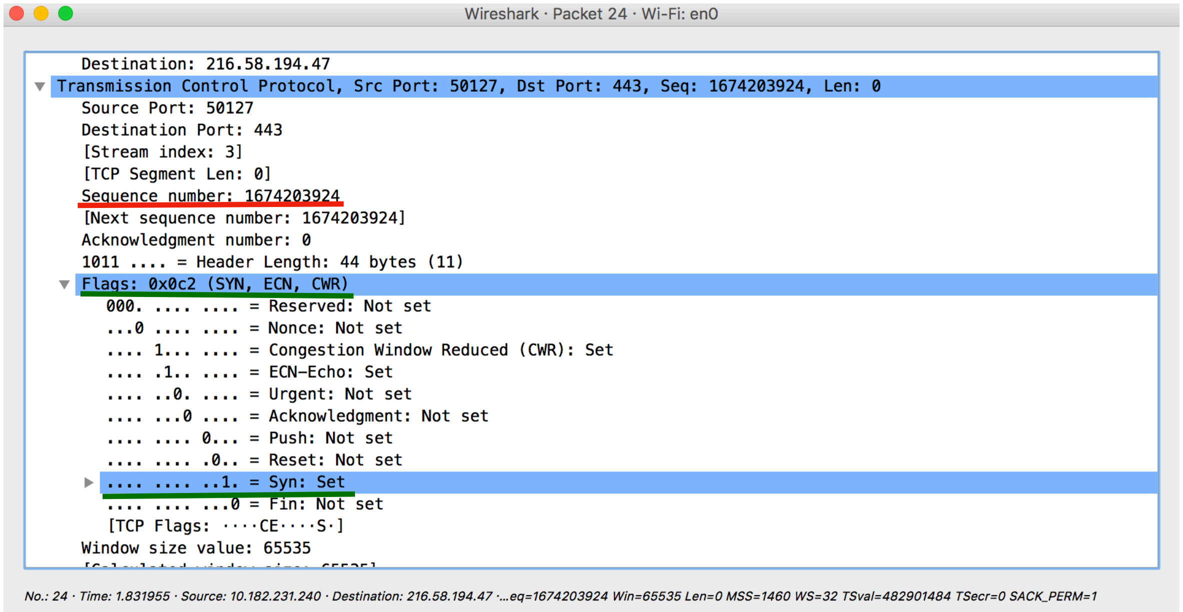
9. Field names are given below:

```
▼ Flags: 0x010 (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
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A····]
```

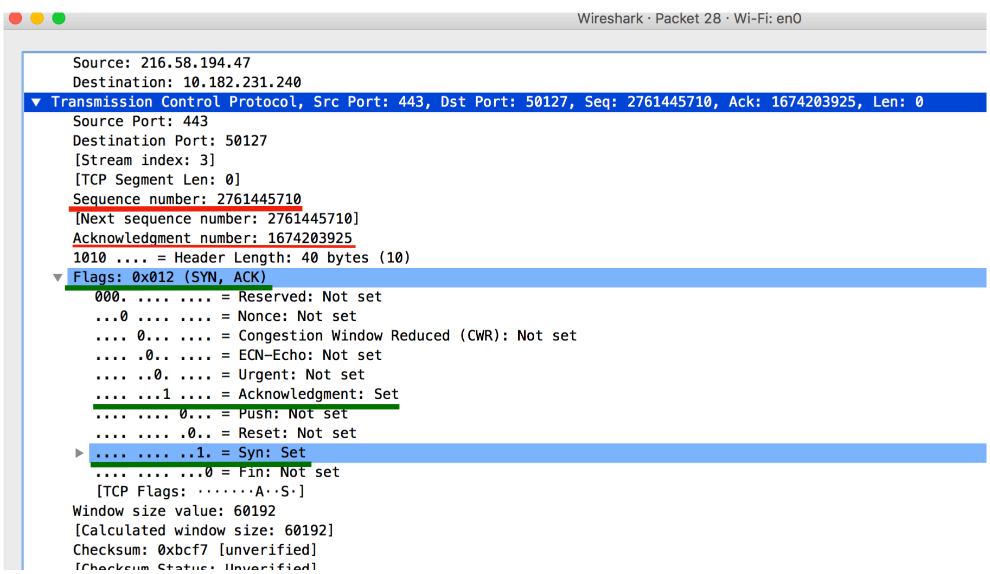
10. UTA IP: 129.107.56.31
UTA Port: 443

Connor Twohey

Problem Set 2:



1. Absolute Sequence Number = 1674203924 (underlined in red)
2. The TCP flags define this segment as a SYN segment (underlined with green)



3. Absolute Seq Number = 2761445710
Absolute ACK Number = 1674203925
(Both underlined in red)

4. When YouTube's (or any TCP) server receives the initial SYN message, it takes that initial message's sequence number, and sets the ACK value of the SYNACK reply equal to the initial sequence number plus one. This is the normal behavior for a TCP server constructing a SYNACK message.

The TCP flags define this segment as a SYNACK message (underlined in green).

Problem Set 3 (images use relative seq/ACK numbers):

```
▶ Frame 140: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits) on interface 0
▶ Ethernet II, Src: Apple_8b:6e:80 (6c:40:08:8b:6e:80), Dst: fa:cf:9c:21:5f:64 (fa:cf:9c:21:5f:64)
▶ Internet Protocol Version 4, Src: 172.20.10.2, Dst: 23.235.44.231
▼ Transmission Control Protocol, Src Port: 55790, Dst Port: 80, Seq: 1415, Ack: 56130, Len: 1388
    Source Port: 55790
    Destination Port: 80
    [Stream index: 4]
    [TCP Segment Len: 1388]
    Sequence number: 1415    (relative sequence number)
    [Next sequence number: 2803    (relative sequence number)]
    Acknowledgment number: 56130    (relative ack number)
0000  fa cf 9c 21 5f 64 6c 40  08 8b 6e 80 08 00 45 00   ···!_dl@ ··n···E·
0010  05 a0 d5 66 40 00 40 06  65 09 ac 14 0a 02 17 eb   ···f@·@· e·······
0020  2c e7 d9 ee 00 50 12 fc  0d cf 3f 57 a5 79 80 10   ,····P·· ··?W·y··
0030  10 00 36 9a 00 00 01 01  08 0a 06 4f 76 33 03 0d   ··6····· ···Ov3··
0040  d1 4e 50 4f 53 54 20 2f  76 73 2f 70 61 67 65 2f   ·NPOST / vs/page/
0050  68 6f 74 65 6c 2f 72 65  73 75 6c 74 73 20 48 54   hotel/re sults HT
0060  54 50 2f 31 2e 31 0d 0a  48 6f 73 74 3a 20 77 77   TP/1.1·· Host: ww
0070  77 2e 6b 61 79 61 6b 2e  63 6f 6d 0d 0a 41 63 63   w.kayak. com··Acc
0080  65 70 74 3a 20 2a 2f 2a  0d 0a 58 2d 52 65 71 75   ept: */* ··X-Requ
0090  65 73 74 65 64 2d 57 69  74 68 3a 20 58 4d 4c 48   ested-Wi th: XMLH
00a0  74 74 70 52 65 71 75 65  73 74 0d 0a 41 63 63 65   ttpReque st··Acce
00b0  70 74 2d 4c 61 6e 67 75  61 67 65 3a 20 65 6e 2d   pt-Langu age: en-
00c0  75 73 0d 0a 41 63 63 65  70 74 2d 45 6e 63 6f 64   us··Acce pt-Encod
```

1. Relative Seq Num = 1415 (underlined in red)

```
140 4.081502   172.20.10.2     23.235.44.231   TCP     1454 55790 → 80 [ACK] Seq=1415 Ack=56130 Win=4096 Len=1388 TSval=105870899 TSecr=51237198
141 4.081503   172.20.10.2     23.235.44.231   TCP      201 55790 → 80 [PSH, ACK] Seq=2803 Ack=56130 Win=4096 Len=135 TSval=105870899 TSecr=51237:
142 4.081603   172.20.10.2     23.235.44.231   HTTP      75 POST /vs/page/hotel/results HTTP/1.1  (application/x-www-form-urlencoded)
143 4.082853   172.20.10.2     209.105.248.3   TLSv1    583 Client Hello
144 4.135363   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2803 Win=72 Len=0 TSval=51237230 TSecr=105870899
145 4.135713   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2938 Win=77 Len=0 TSval=51237230 TSecr=105870899
146 4.135716   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2947 Win=77 Len=0 TSval=51237230 TSecr=105870899
▶ Frame 142: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
▶ Ethernet II, Src: Apple_8b:6e:80 (6c:40:08:8b:6e:80), Dst: fa:cf:9c:21:5f:64 (fa:cf:9c:21:5f:64)
▶ Internet Protocol Version 4, Src: 172.20.10.2, Dst: 23.235.44.231
▼ Transmission Control Protocol, Src Port: 55790, Dst Port: 80, Seq: 2938, Ack: 56130, Len: 9
```

2. i. 1415 (140), 2803 (141), 2938 (142), 56130 (144) (underlined in red above)
ii. 4.081502 s (140), 4.081503 s (141), 4.081603 s (142), 4.135363 s (144)
iii. Segment 144 is the segment that ACKs segment 140
Segments 145 and 146 ACK segments 141 and 142 respectively
ACK for Segment 140 received at 4.135363 s
ACK for Segment 141 received at 4.135713 s
ACK for Segment 142 received at 4.135716 s
Segment 144 is an ACK so N/A
See Screenshot below

```
140 4.081502   172.20.10.2     23.235.44.231   TCP     1454 55790 → 80 [ACK] Seq=1415 Ack=56130 Win=4096 Len=1388 TSval=105870899 TSecr=51237198
141 4.081503   172.20.10.2     23.235.44.231   TCP      201 55790 → 80 [PSH, ACK] Seq=2803 Ack=56130 Win=4096 Len=135 TSval=105870899 TSecr=5123
142 4.081603   172.20.10.2     23.235.44.231   HTTP      75 POST /vs/page/hotel/results HTTP/1.1  (application/x-www-form-urlencoded)
143 4.082853   172.20.10.2     209.105.248.3   TLSv1    583 Client Hello
144 4.135363   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2803 Win=72 Len=0 TSval=51237230 TSecr=105870899
145 4.135713   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2938 Win=77 Len=0 TSval=51237230 TSecr=105870899
146 4.135716   23.235.44.231   172.20.10.2     TCP       66 80 → 55790 [ACK] Seq=56130 Ack=2947 Win=77 Len=0 TSval=51237230 TSecr=105870899
```

iv. & v. RTT(140) = 4.135363 s - 4.081502 s = 0.053861 s
RTT(141) = 4.135713 s - 4.081503 s = 0.05421 s
RTT(142) = 4.135716 s - 4.081603 s = 0.054113 s
No RTT for 144 (144 is an ACK)

vi. & vii. Given that EstimatedRTT(140) = RTT(140) = 0.053861 s
alpha = 0.125
EstimatedRTT(141) = (0.875)(0.053861) + (0.125)(0.05421) = 0.047128 + 0.0067763 = 0.053904 s
EstimatedRTT(142) = (0.875)(0.053904) + (0.125)(0.054113) = 0.047166 + 0.0067641 = 0.0539301 s
No EstimatedRTT for 144 (144 is an ACK)

3. Length(140) = 1388, Length(141) = 135, Length(142) = 9, Length(144) = 0

```
▶ Frame 59: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Ethernet II, Src: fa:cf:9c:21:5f:64 (fa:cf:9c:21:5f:64), Dst: Apple_8b:6e:80 (6c:40:08:8b:6e:80)
▶ Internet Protocol Version 4, Src: 23.235.44.231, Dst: 172.20.10.2
▼ Transmission Control Protocol, Src Port: 80, Dst Port: 55790, Seq: 5039, Ack: 1415, Len: 2
    Source Port: 80
    Destination Port: 55790
    [Stream index: 4]
    [TCP Segment Len: 2]
    Sequence number: 5039    (relative sequence number)
    [Next sequence number: 5041    (relative sequence number)]
    Acknowledgment number: 1415    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x018 (PSH, ACK)
    Window size value: 66
    [Calculated window size: 66]
```

4. 66 (in first ACK received from server, port 55790, underlined in red above)
5. Yes. The sender is throttled by the receiver every time the sender tries to send a longer segment than the receiver's window can currently hold. An example is given below, where the length of the received segment is larger (68) than the size of the window (66):

```
59 3.931066    23.235.44.231    172.20.10.2    TCP    68 80 → 55790 [PSH, ACK] Seq=5039 Ack=1415 Win=66
```

6. Yes, there are some. There are none on port 55790, but there are retransmissions on other ports/connections. If there is a retransmission, my version of WireShark lets me know (example given):

```
9 0.059976    172.20.10.2    173.194.115.90    TCP    78 [TCP Retransmission] 55720 → 443 [FIN, ACK] Seq=1 Ack=65 Win=4096 Len=0
```

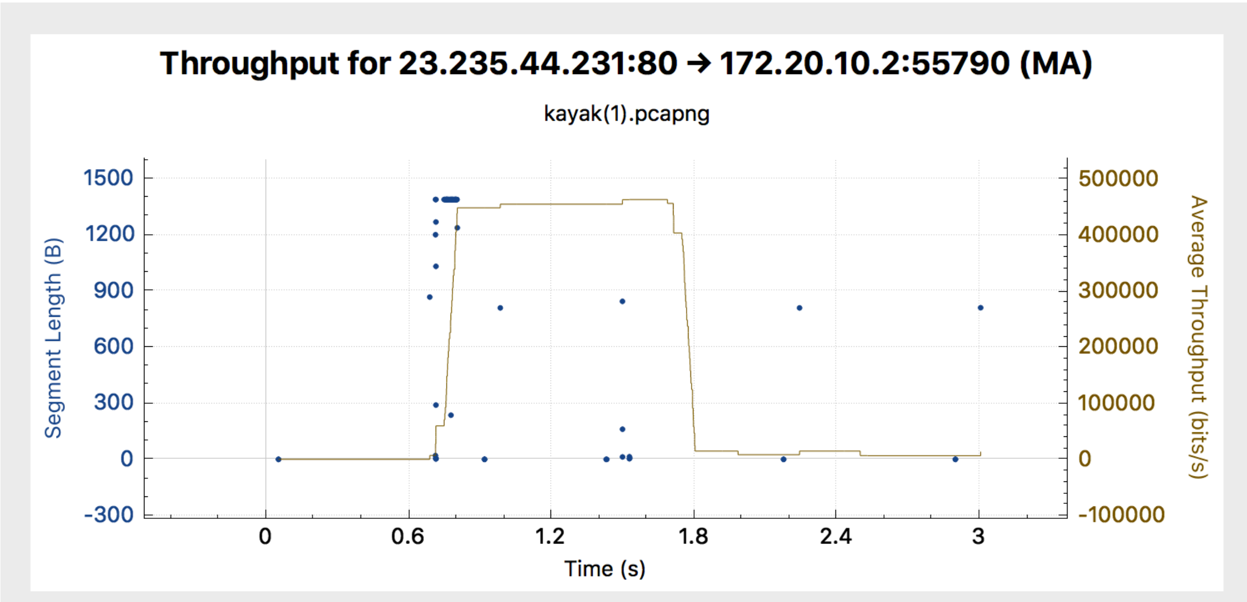
7. Typically, the amount of data received is equal to the difference between the ACK numbers of two consecutive ACKs (from sender to receiver). Below is a table with each pair of consecutive ACK segments (all on port 55790):

Consecutive ACK Pairs (by number)	**Difference in ACK # (= Data Received)**
35 - 36	26
144 - 145	135
145 - 146	9
356 - 357	29
444 - 445	29

Perhaps it has to do with the specific connection being used for analysis, but there weren't very many consecutive ACK pairs where the sender was sending data to the receiver. I didn't include pairs of consecutive [PSH, ACK] segments, because every single one had the same ACK number. However, out of the pairs that do exist, the typical amount of data received/acknowledged per ACK seems to be around 29.

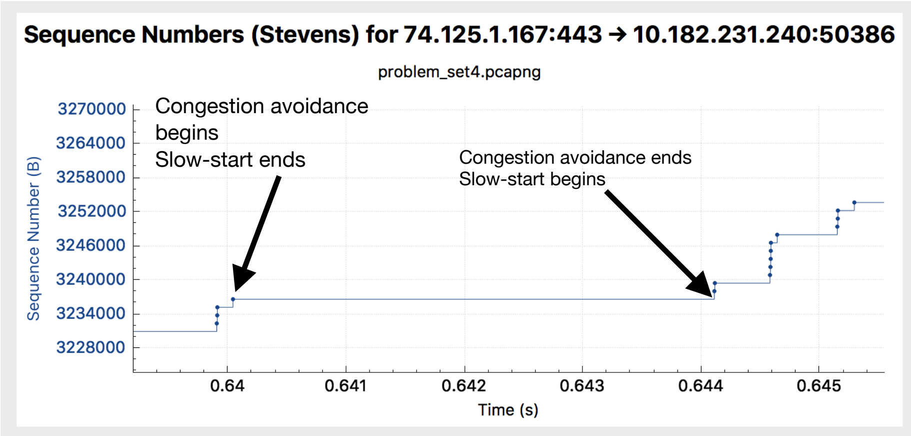
8. The average throughput of the connection on port 55790 is around 455500 bits/sec.

9. I used the TCP Stream Graph for Throughput, zoomed in on the curve to see better where it lies. A picture of the chart is given below:



Problem Set 4:

For this set of problems, I took only a portion of the Stevens graph, since you can't see the details of the graph unless you zoom in very closely:



1. The slow-start phase begins whenever the sequence number is increasing (meaning that the sender is sending out data) and ends when the sequence number stops increasing (meaning that the receiver can't take anymore data in it's buffer).
2. Congestion avoidance takes place whenever the sequence number is not increasing (or rather, when it is constant). This means that the sender is not sending out any more data. In this case, this will only happen when the receiver buffer is too full. So it is at these times that congestion avoidance takes control on the receiver's end (the sender has no need for congestion avoidance). In the picture above, the points at which congestion avoidance takes over are labeled. Also, for the purposes of the given screenshot, congestion avoidance begins whenever slow-start ends, and congestion avoidance ends whenever slow-start begins, so the labels can be used for both procedures.
3. For one thing, the ideal TCP behavior seems to act as if no data will be lost, when in reality, there were some segments that were lost and had to be resent. It also seems that, with realistic TCP behavior, there are quite a few other types of errors that can happen that don't even seem to get mention with the idealized TCP we've been studying. Another big difference is that the amount of data sent/received each time is not guaranteed to be constant (or even close, sometimes), unlike in the idealized version of TCP we've studied, where we usually assume that our data transmits exactly the amount we tell it to at one time.