Q1 TCP sockets and the TCP state diagram

3.5 Points

Q1.1 TCP connection refused

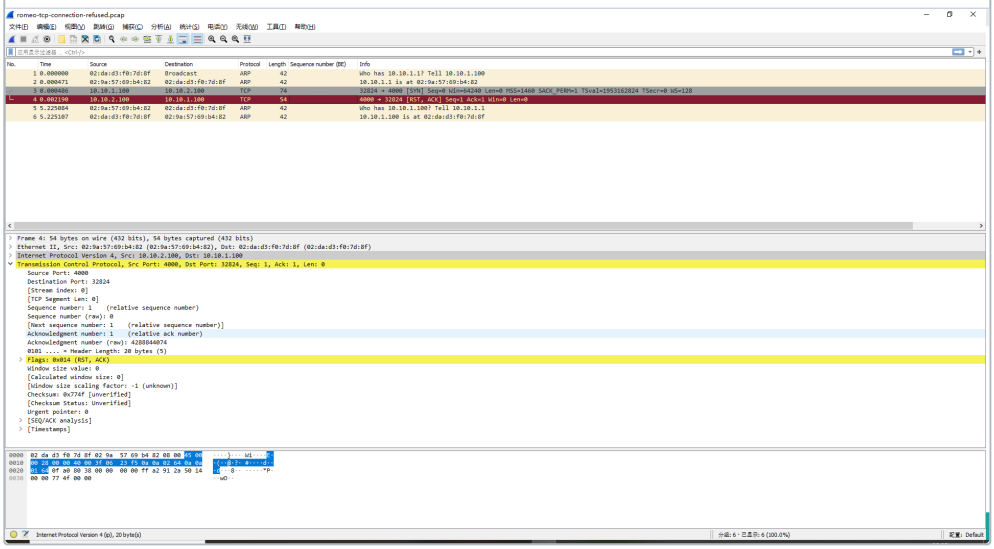
0.5 Points

In Lab 2, you saw what happened when a client tries to reach a UDP port on which no process is listening. In this exercise, you saw what happened when a client tries to reach a TCP port on which no process is listening.

Show the message that is sent when a client tries to reach a TCP port on which no process is listening. (You can show a screenshot from Wireshark or a line from `tcpdump`, but make sure to crop your submission to only include the relevant output.)

screenshot

tcp_connection_refused.png Download



Explain what happened in the TCP case, and note any difference from the UDP case. What kind of message (TCP, UDP, ICMP) is sent from the *destination host* in each case? What header field is used in each case, to indicate that there is no service listening on that port? (Make sure these details are visible in your screenshot or `tcpdump` output above!)

In TCP case. The destination host sent back an TCP ACK back to t he source host. With ethernet, IP and TCP header. In UDP case. Th e destination host sent back an ICMP port unreachable message t o source host, with ethernet, IP header.

Q1.2 TCP connection establishment

0.5 Points

Use the packets you captured to explain TCP connection establishment.

Explain how the client goes from `CLOSED` to `SYN_SENT` to `ESTABLISHED` state, and show the packet that is sent and/or received by the client at each state transition. What flags are set in the TCP header of each packet? (Make sure you show the relevant part of the TCP header.)

Client goes to STN_SENT by sending a TCP packet with flag [S] to the destination host. And then goes to ESTABLISHED by recieving a TCP SYN ACK packet with flag [S.] and sending a TCP ACK pack et with flag [.]

client_ack.png Download

```
16:34:39.750054 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [.], ack 82662040
5, win 502, options [nop,nop,TS val 1965558628 ecr 381426864], length 0
        0x0000:  4500 0034 c93b 4000 4006 59ad 0a0a 0164  E..4.;@.@.Y....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ed1 3145 39f5  ...d.@...~>.1E9.
        0x0020:  8010 01f6 1802 0000 0101 080a 7528 0b64  ............u(.d
        0x0030:  16bc 1cb0                                ....
```

client_recv_ack.png Download

```
16:34:39.750007 IP 10.10.2.100.4000 > 10.10.1.100.32832: Flags [S.], seq 8266204
04, ack 2256420561, win 65160, options [mss 1460,sackOK,TS val 381426864 ecr 196
5558626,nop,wscale 7], length 0
        0x0000:  4500 003c 0000 4000 3f06 23e1 0a0a 0264  E..<..@.?.#....d
        0x0010:  0a0a 0164 0fa0 8040 3145 39f4 867e 3ed1  ...d...@1E9..~>.
        0x0020:  a012 fe88 180a 0000 0204 05b4 0402 080a  ................
        0x0030:  16bc 1cb0 7528 0b62 0103 0307            ....u(.b....
```

client_syn.png Download

```
16:34:39.748697 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [S], seq 22564205
60, win 64240, options [mss 1460,sackOK,TS val 1965558626 ecr 0,nop,wscale 7], l
ength 0
        0x0000:  4500 003c c93a 4000 4006 59a6 0a0a 0164  E..<.:@.@.Y....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ed0 0000 0000  ...d.@...~>.....
        0x0020:  a002 faf0 180a 0000 0204 05b4 0402 080a  ................
        0x0030:  7528 0b62 0000 0000 0103 0307            u(.b........
```

Explain how the server goes from `CLOSED` to `LISTEN` to `SYN_RCVD` to `ESTABLISHED` state, and show the packet that is sent and/or received by the server at each state transition. What flags are set in the TCP header of each packet? (Make sure you show the relevant part of the TCP header.)

Server goes to LISTEN by call python function sock.listen() and the n goes to STN_RCVD by receving a TCP packet with flag [S] from the source host. And then goes to ESTABLISHED by sending a TC P SYN ACK packet with flag [S.] and receving a TCP ACK packet wi th flag [.]

server_listen.png Download

```
>>> sock.listen()
```

server_recv_ack.png Download

```
16:34:39.746614 IP (tos 0x0, ttl 63, id 51515, offset 0, flags [DF], proto TCP (
6), length 52)
    10.10.1.100.32832 > 10.10.2.100.4000: Flags [.], cksum 0x1802 (incorrect ->
0xe888), ack 826620405, win 502, options [nop,nop,TS val 1965558628 ecr 38142686
4], length 0
        0x0000:  4500 0034 c93b 4000 3f06 5aad 0a0a 0164  E..4.;@.?.Z....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ed1 3145 39f5  ...d.@...~>.1E9.
        0x0020:  8010 01f6 1802 0000 0101 080a 7528 0b64  ............u(.d
        0x0030:  16bc 1cb0                                ....
```

server_recv_syn.png Download

```
16:34:39.745541 IP (tos 0x0, ttl 63, id 51514, offset 0, flags [DF], proto TCP (
6), length 60)
    10.10.1.100.32832 > 10.10.2.100.4000: Flags [S], cksum 0x180a (incorrect ->
0x5f7a), seq 2256420560, win 64240, options [mss 1460,sackOK,TS val 1965558626 e
cr 0,nop,wscale 7], length 0
        0x0000:  4500 003c c93a 4000 3f06 5aa6 0a0a 0164  E..<.:@.?.Z....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ed0 0000 0000  ...d.@...~>.....
        0x0020:  a002 faf0 180a 0000 0204 05b4 0402 080a  ................
        0x0030:  7528 0b62 0000 0000 0103 0307            u(.b........
```

server_syn_ack.png Download

```
16:34:39.745616 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6),
length 60)
    10.10.2.100.4000 > 10.10.1.100.32832: Flags [S.], cksum 0x180a (incorrect ->
 0xbd2b), seq 826620404, ack 2256420561, win 65160, options [mss 1460,sackOK,TS
val 381426864 ecr 1965558626,nop,wscale 7], length 0
        0x0000:  4500 003c 0000 4000 4006 22e1 0a0a 0264  E..<..@.@."....d
        0x0010:  0a0a 0164 0fa0 8040 3145 39f4 867e 3ed1  ...d...@1E9..~>.
        0x0020:  a012 fe88 180a 0000 0204 05b4 0402 080a  ................
        0x0030:  16bc 1cb0 7528 0b62 0103 0307            ....u(.b....
```

Q1.3 TCP options

0.5 Points

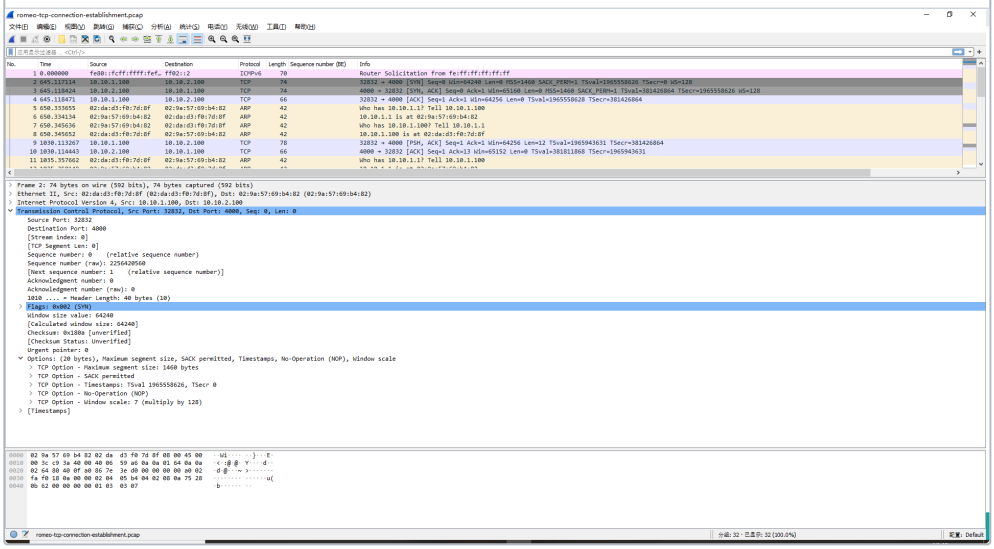
As part of the three-way handshake used for connection establishment, hosts exchange information about optional capabilities that they support - for example, if they support a maximum segment size (MSS) greater than the default value of 536 byte.

Which of the following options are used by both endpoints in this exchange? (Here is a list of TCP options, with links to more information about each one.)

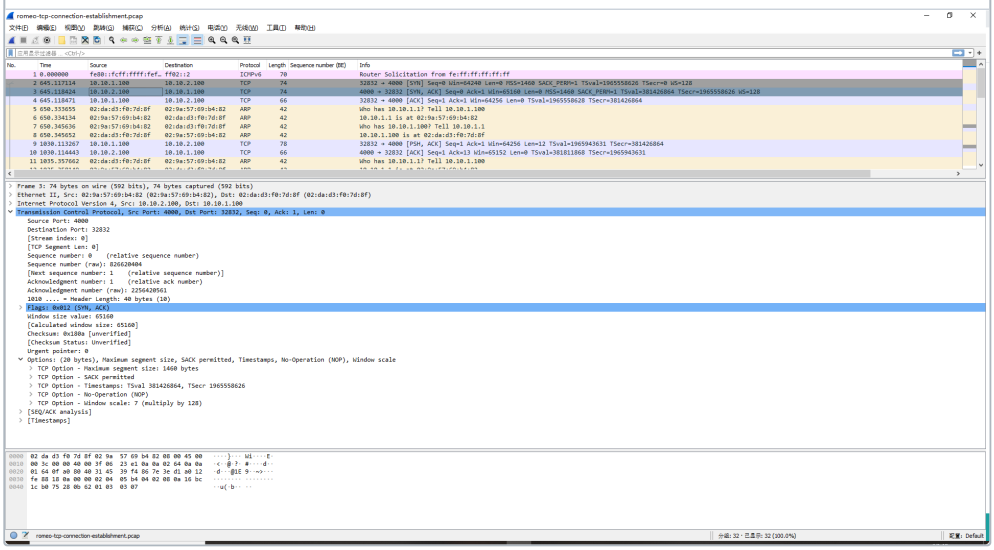
- [x] Advertise a MSS greater than 536 bytes
- [ ] Use Multipath TCP, to simultaneously use multiple paths between the hosts.
- [x] Support selective acknowledgments (SACK)
- [ ] Use TCP Authentication Option, to protect against an attack in which a malicious eavesdropper captures and then plays back the TCP connection.
- [x] Set a window scale factor, which the other endpoint should use as a multiplier on the advertised receive window size field to compute the actual RWND.

Show a screenshot of both of the connection establishment packets from Wireshark, with the options section of the TCP header visible and expanded.

▼ syn.png Download



syn_ack.png Download



Q1.4 TCP Socket API calls

0.5 Points

You would expect to see TCP packets appear on a network link directly following which socket API calls?

- [ ] bind
- [ ] listen
- [ ] accept
- [x] connect
- [x] send
- [ ] recv
- [x] shutdown

Q1.5 TCP socket connection establishment

0.25 Points

When a server receives a SYN over a TCP connection, it sends a SYN ACK in response. When is the SYN ACK sent?

- [x] If the socket is in the `LISTEN` state, the SYN ACK will be sent by the operating system as soon as it receives the SYN.
- [ ] The SYN ACK is sent only after the server calls `accept`.

Q1.6 TCP socket ACKs

0.25 Points

When a host receives a data segment over a TCP connection, it sends an ACK. When is this ACK sent?

- [x] When the data is received by the operating system and placed in a buffer.
- [ ] Only once the data is processed by the application, following a call to `recv`.

Q1.7 TCP sequence numbers

0.5 Points

Use the packets you captured to show how TCP sequence numbers and ACK numbers are computed and incremented, by answering the following questions.

(Make sure to show *raw* sequence numbers, not the relative sequence numbers computed by Wireshark, or that you would see in `tcpdump` if you don't use the `-S` display option.)

Show the SYN sent from romeo to juliet (copy and paste from `tcpdump` output, but make sure to use the `-S` option to show "raw" sequence numbers!) Also, answer this question: What is the initial sequence number?

```
16:34:39.748697 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [S], s
eq 2256420560, win 64240, options [mss 1460,sackOK,TS val 196
5558626 ecr 0,nop,wscale 7], length 0
        0x0000:  4500 003c c93a 4000 4006 59a6 0a0a 0164  E..<.:
@.@.Y....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ed0 0000 0000  ...d.
@...~>.....
        0x0020:  a002 faf0 180a 0000 0204 05b4 0402 080a
 ................
        0x0030:  7528 0b62 0000 0000 0103 0307            u(.b........

initial sequence number: 2256420560
```

Show the ACK sent from juliet to romeo in response to the SYN. Also, answer these questions: What is the ACK number? What does this indicate?

```
16:34:39.750007 IP 10.10.2.100.4000 > 10.10.1.100.32832: Flags [S.],
 seq 826620404, ack 2256420561, win 65160, options [mss 1460,
sackOK,TS val 381426864 ecr 1965558626,nop,wscale 7], length
 0
        0x0000:  4500 003c 0000 4000 3f06 23e1 0a0a 0264  E..<..
@.?.#....d
        0x0010:  0a0a 0164 0fa0 8040 3145 39f4 867e 3ed1  ...d...@1
E9..~>.
        0x0020:  a012 fe88 180a 0000 0204 05b4 0402 080a
 ................
        0x0030:  16bc 1cb0 7528 0b62 0103 0307            ....u(.b....

ack number: 2256420561
the ack number equals to initial sequence number incremnets by 1
```

Show the *first* data segment sent from romeo to juliet. Also, answer these questions: What is the sequence number? What is the TCP segment length? (Note: `tcpdump` may show you the sequence number as a range, with a start and end value. Only the lower value is actually in the packet header; report this value here.)

16:41:04.744850 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [P.], seq 2256420561:2256420573, ack 826620405, win 502, options [nop,nop,TS val 1965943631 ecr 381426864], length 12
0x0000: 4500 0040 c93c 4000 4006 59a0 0a0a 0164 E..@.<@.@.Y....d
0x0010: 0a0a 0264 8040 0fa0 867e 3ed1 3145 39f5 ...d.@...~>.1E9.
0x0020: 8018 01f6 180e 0000 0101 080a 752d eb4f ....u-.O
0x0030: 16bc 1cb0 4865 6c6c 6f20 6a75 6c69 6574 ....Hello.juliet

initial sequnece number: 2256420561

Show the ACK sent from juliet to romeo in response to this data segment. Also, answer these questions:
What is the ACK number? What does this indicate?

16:41:04.746026 IP 10.10.2.100.4000 > 10.10.1.100.32832: Flags [.], ack 2256420573, win 509, options [nop,nop,TS val 381811868 ecr 1965943631], length 0
0x0000: 4500 0034 f634 4000 3f06 2db4 0a0a 0264 E..4.4@.?.-....d
0x0010: 0a0a 0164 0fa0 8040 3145 39f5 867e 3edd ...d...@1E9..~>.
0x0020: 8010 01fd 1802 0000 0101 080a 16c1 fc9c ....
0x0030: 752d eb4f u-.O

ACK number: 2256420573
By comparing the ACK number with the initial sequnence number we know it receive data with size 12 byte

Q1.8 TCP connection termination

0.5 Points

Use the packets you captured to explain TCP connection termination.

Explain how the client goes from `ESTABLISHED` to `FIN_WAIT_1` to `FIN_WAIT_2` to `TIME_WAIT` to `CLOSED` state, and show the packet that is sent and/or received by the client at each state transition. What flags are set in the TCP header of each packet? (Make sure you show the relevant part of the TCP header.)

The client goes from ESTABLISHED to FIN_WAIT_1 by sending a TCP FIN packet with flag [F.] and goes from FIN_WAIT_2 by receving a TCP ACK packet with flag [.]. Then it goes to TIME_WAIT by receiving a TCP FIN packet and sending a TCP ACK packet. After 2MSL timeout it will goes to CLOSED state.

client_f1.png Download

```
17:03:46.089900 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [F.], seq 2256420
573, ack 826620416, win 502, options [nop,nop,TS val 1967305007 ecr 382969405],
length 0
        0x0000:  4500 0034 c93e 4000 4006 59aa 0a0a 0164  E..4.>@.@.Y....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3edd 3145 3a00  ...d.@...~>.1E:.
        0x0020:  8011 01f6 1802 0000 0101 080a 7542 b12f  ............uB./
        0x0030:  16d3 a63d                                ...=
```

client_f2.png Download

```
17:05:07.168131 IP 10.10.2.100.4000 > 10.10.1.100.32832: Flags [F.], seq 8266204
16, ack 2256420574, win 509, options [nop,nop,TS val 383254323 ecr 1967305007],
length 0
        0x0000:  4500 0034 f637 4000 3f06 2db1 0a0a 0264  E..4.7@.?.-....d
        0x0010:  0a0a 0164 0fa0 8040 3145 3a00 867e 3ede  ...d...@1E:..~>.
        0x0020:  8011 01fd 1802 0000 0101 080a 16d7 ff33  ...............3
        0x0030:  7542 b12f                                uB./
```

client_recv_f1_ack.png Download

```
17:03:46.134145 IP 10.10.2.100.4000 > 10.10.1.100.32832: Flags [.], ack 22564205
74, win 509, options [nop,nop,TS val 383173287 ecr 1967305007], length 0
        0x0000:  4500 0034 f636 4000 3f06 2db2 0a0a 0264  E..4.6@.?.-....d
        0x0010:  0a0a 0164 0fa0 8040 3145 3a00 867e 3ede  ...d...@1E:..~>.
        0x0020:  8010 01fd 1802 0000 0101 080a 16d6 c2a7  ................
        0x0030:  7542 b12f                                uB./
```

client_recv_f2_ack.png Download

```
17:05:07.168169 IP 10.10.1.100.32832 > 10.10.2.100.4000: Flags [.], ack 82662041
7, win 502, options [nop,nop,TS val 1967386087 ecr 383254323], length 0
        0x0000:  4500 0034 c93f 4000 4006 59a9 0a0a 0164  E..4.?@.@.Y....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ede 3145 3a01  ...d.@...~>.1E:.
        0x0020:  8010 01f6 1802 0000 0101 080a 7543 ede7  ............uC..
        0x0030:  16d7 ff33                                ...3
```

Explain how the server goes from `ESTABLISHED` to `CLOSE_WAIT` to `LAST_ACK` to `CLOSED` state, and show the packet that is sent and/or received by the server at each state transition. What flags are set in the TCP header of each packet? (Make sure you show the relevant part of the TCP header.)

The server goes from ESTABLISHED to CLOSE_WAIT by receiving a TCP FIN packet with flag [F.] and seding a TCP ACK packet with flag [.]. It goes to LAST_ACK by sending a TCP FIN packet and go es to CLOSED state by recieving a TCP ACK packet.

server_f1.png Download

```
17:03:46.086275 IP (tos 0x0, ttl 63, id 51518, offset 0, flags [DF], proto TCP (
6), length 52)
    10.10.1.100.32832 > 10.10.2.100.4000: Flags [F.], cksum 0x1802 (incorrect ->
 0xb8e6), seq 2256420573, ack 826620416, win 502, options [nop,nop,TS val 196730
5007 ecr 382969405], length 0
        0x0000:  4500 0034 c93e 4000 3f06 5aaa 0a0a 0164  E..4.>@.?.Z....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3edd 3145 3a00  ...d.@...~>.1E:.
        0x0020:  8011 01f6 1802 0000 0101 080a 7542 b12f  ............uB./
        0x0030:  16d3 a63d                                ...=
```

server_f1_ack.png Download

```
17:03:46.129169 IP (tos 0x0, ttl 64, id 63030, offset 0, flags [DF], proto TCP (
6), length 52)
    10.10.2.100.4000 > 10.10.1.100.32832: Flags [.], cksum 0x1802 (incorrect ->
0x9c72), ack 2256420574, win 509, options [nop,nop,TS val 383173287 ecr 19673050
07], length 0
        0x0000:  4500 0034 f636 4000 4006 2cb2 0a0a 0264  E..4.6@.@.,....d
        0x0010:  0a0a 0164 0fa0 8040 3145 3a00 867e 3ede  ...d...@1E:..~>.
        0x0020:  8010 01fd 1802 0000 0101 080a 16d6 c2a7  ................
        0x0030:  7542 b12f                                uB./
```

server_f2.png Download

```
17:05:07.163193 IP (tos 0x0, ttl 64, id 63031, offset 0, flags [DF], proto TCP (
6), length 52)
    10.10.2.100.4000 > 10.10.1.100.32832: Flags [F.], cksum 0x1802 (incorrect ->
 0x5fe4), seq 826620416, ack 2256420574, win 509, options [nop,nop,TS val 383254
323 ecr 1967305007], length 0
        0x0000:  4500 0034 f637 4000 4006 2cb1 0a0a 0264  E..4.7@.@.,....d
        0x0010:  0a0a 0164 0fa0 8040 3145 3a00 867e 3ede  ...d...@1E:..~>.
        0x0020:  8011 01fd 1802 0000 0101 080a 16d7 ff33  ...............3
        0x0030:  7542 b12f                                uB./
```

server_f2_ack.png Download

```
17:05:07.164443 IP (tos 0x0, ttl 63, id 51519, offset 0, flags [DF], proto TCP (
6), length 52)
    10.10.1.100.32832 > 10.10.2.100.4000: Flags [.], cksum 0x1802 (incorrect ->
0x2332), ack 826620417, win 502, options [nop,nop,TS val 1967386087 ecr 38325432
3], length 0
        0x0000:  4500 0034 c93f 4000 3f06 5aa9 0a0a 0164  E..4.?@.?.Z....d
        0x0010:  0a0a 0264 8040 0fa0 867e 3ede 3145 3a01  ...d.@...~>.1E:.
        0x0020:  8010 01f6 1802 0000 0101 080a 7543 ede7  ............uC..
        0x0030:  16d7 ff33                                ...3
```

Q2 TCP bulk file transfer - CWND limited

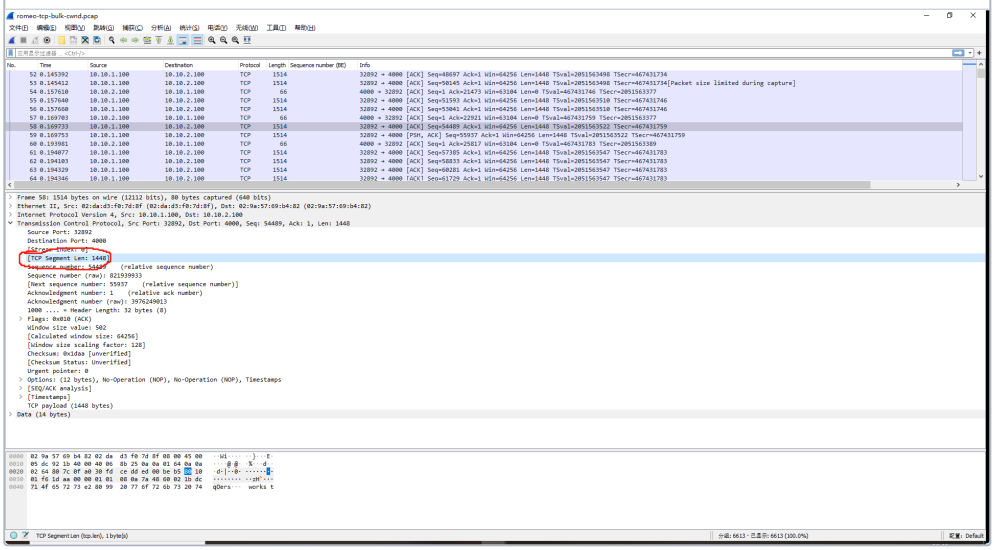
3 Points

Q2.1 TCP as a stream

0.5 Points

The file sending application passes blocks of data with size 4096 bytes as an argument to the `send` socket API call. Look at the TCP segment length in the TCP header of the data segments from romeo to juliet - show a Wireshark screenshot and circle the TCP segment length of a typical data segment.

segment_length.png Download



Is the data transferred in 4096 B blocks? Explain.

No, since 4096 B is much bigger than the MTU

Will TCP segments be fragmented by the IP layer, as oversized UDP datagrams were?

Yes

Can data from different 4096 B blocks (different `send` calls) end up being transmitted in the same TCP segment? Explain. Compare and contrast your answer to this question with the equivalent UDP behavior.

Yes. Using different send calls to transmit data using TCP, they can all be combined together as one TCP segment. But sending different data block with different send calls using UDP, it will never be combined together.

Q2.2 ACKing multiple segments in one ACK

0.5 Points

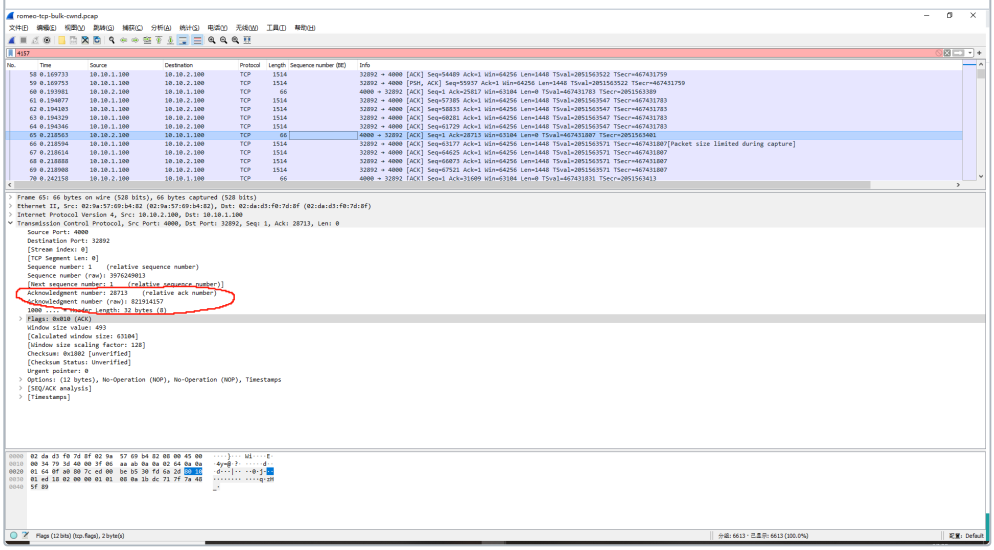
Does the number of data segments differ from that of their acknowledgements?

Find an ACK that acknowledges more than one data segment. Also show the ACK that is received immediately before this one.

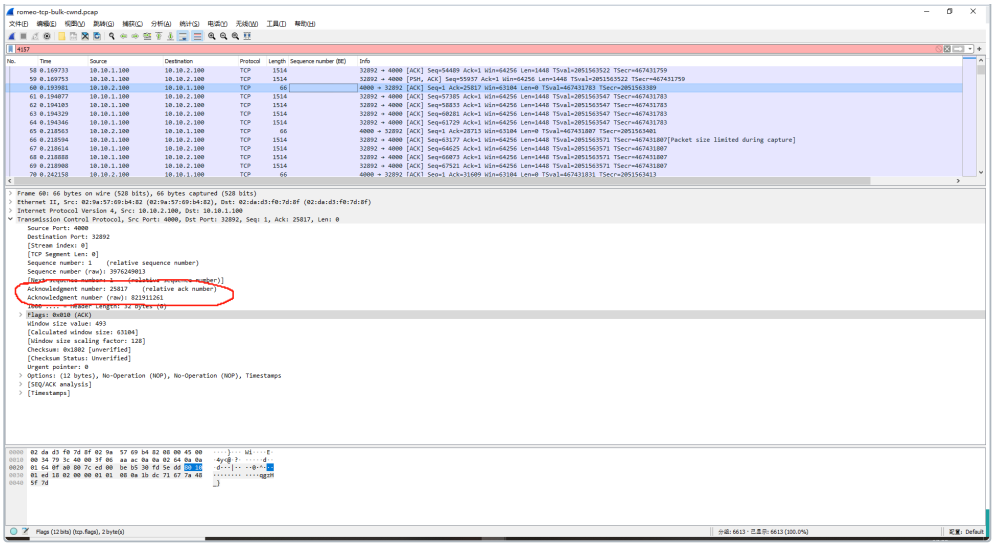
How can you tell that the ACK you have selected acknowledges more than one data segment? Annotate your screenshots by circling the TCP header fields that show evidence of this. (Or, if you are pasting `tcpdump` output, explain the evidence.)

We can tell it from the difference between the two ACK packets. T he difference is 2896 which is 2 * 1448(the size of the data of one segment)

▾ ack_after.png Download



▾ ack_before.png Download



How many data *segments* are acknowledged by that ACK, and how many bytes of data?

2 data segments are acknowledged by that ACK and 2896 bytes o f data.

Q2.3 SACK

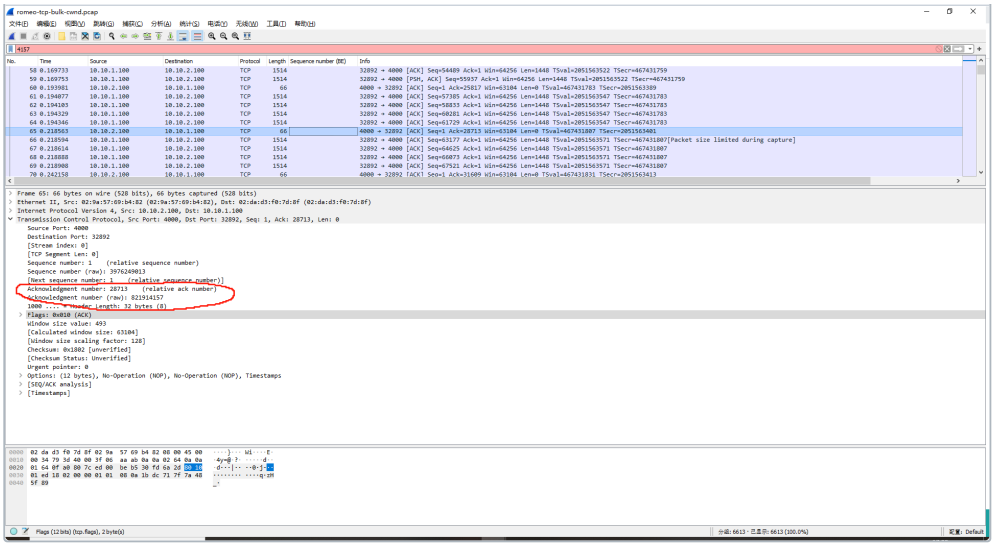
1 Point

When segments are dropped in a TCP connection, the sender retransmits those segments.

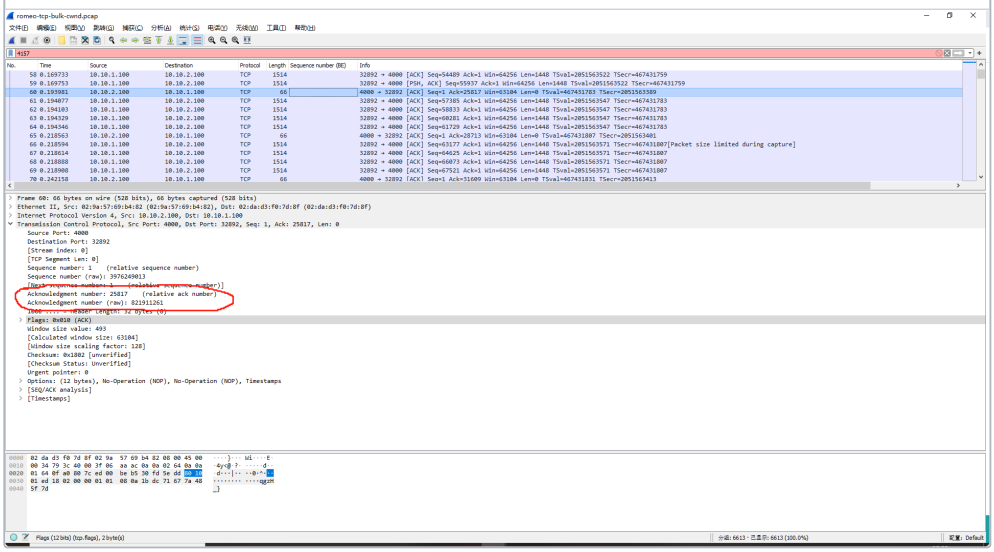
- Find and show a retransmission in your packet capture. Also find and show the initial transmission of this segment. Note the time and the TCP sequence number of each packet. How long did it take for the TCP sender to realize that the segment was lost, and to retransmit it?

from the timestamp, we can see it took 6.2-5.35 = 0.85 sec to reali ze it is lost and do the retransmission.

ack_after.png Download



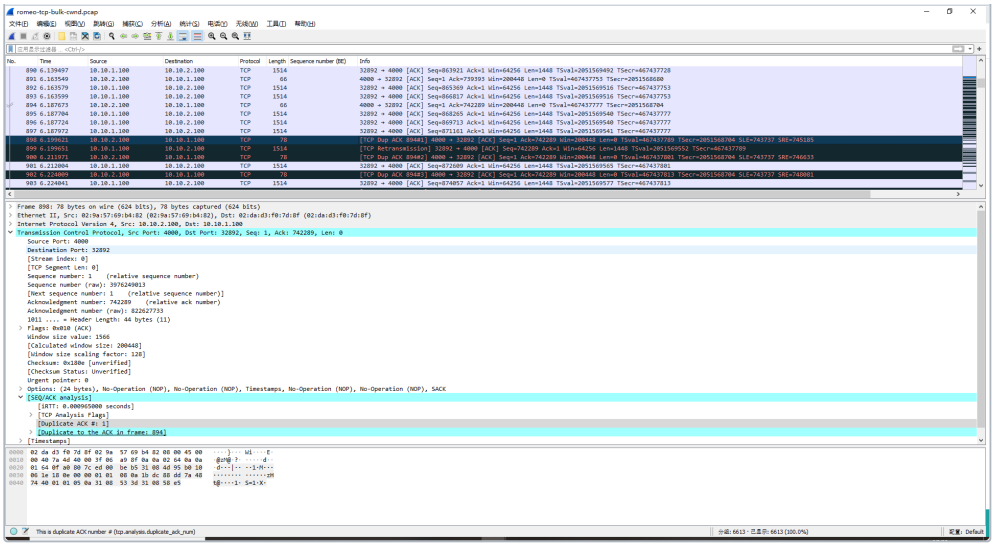
▾ ack_before.png Download



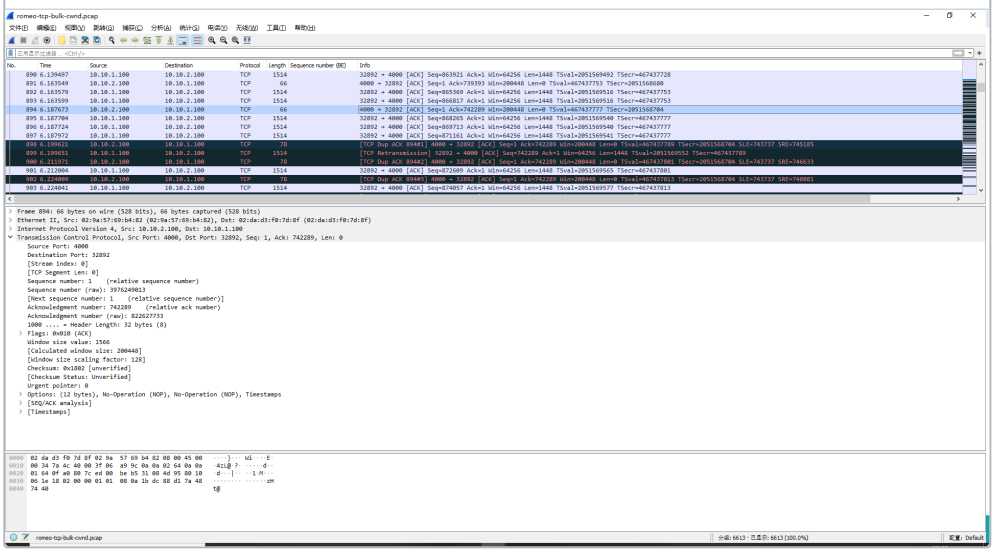
- Did the sender learn that the segment was dropped because it received a duplicate ACK? If so, show the duplicate ACK *and* the first ACK, with the same ACK number. What transmitted data segment did the receiver send the first ACK in response to, and what transmitted data segment did the receiver send the duplicate ACK in response to? Explain. (If the retransmission you selected was not sent in response to a duplicate ACK, you can find any duplicate ACK.)

The first ACK response to the segment with seq number: 868265 and the duplicate ACK response to the segment with seq number: 742289

▾ dup_acj'.png Download



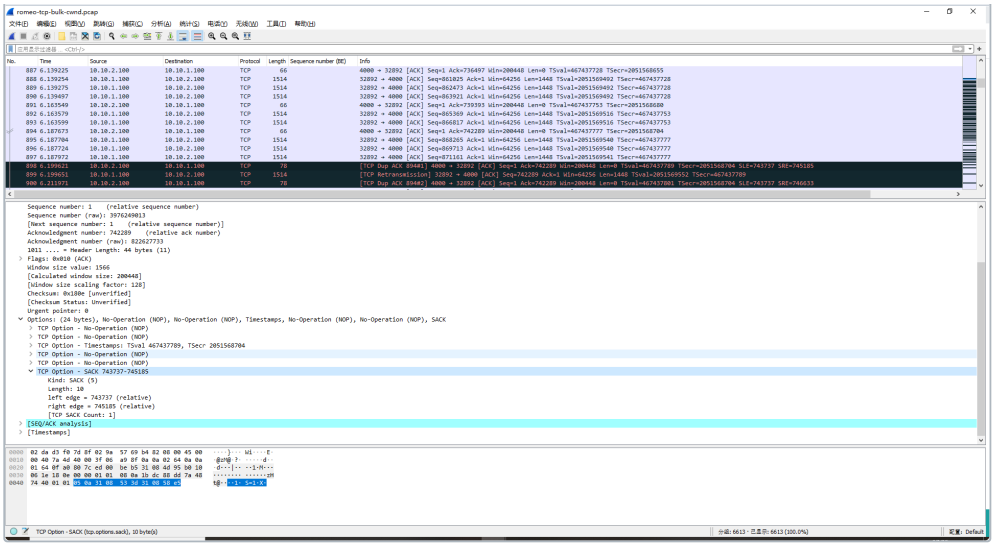
▾ first_ack.png Download



- When sending a duplicate ACK, the receiver can use *selective acknowledgement* to inform the sender about subsequent segments (after the missing segment) that have arrived successfully, so the sender need retransmit only the segment that was lost and not the subsequent segments. The hosts in our connection should be configured to use SACK by default. Show the SACK TCP option in the duplicate ACK. Which segments were received successfully after the missing segment? Explain.

From the TCP SACK option, The segments with sequence number s from 743737 to 745185 were received.

▾ tcp_sack.png Download



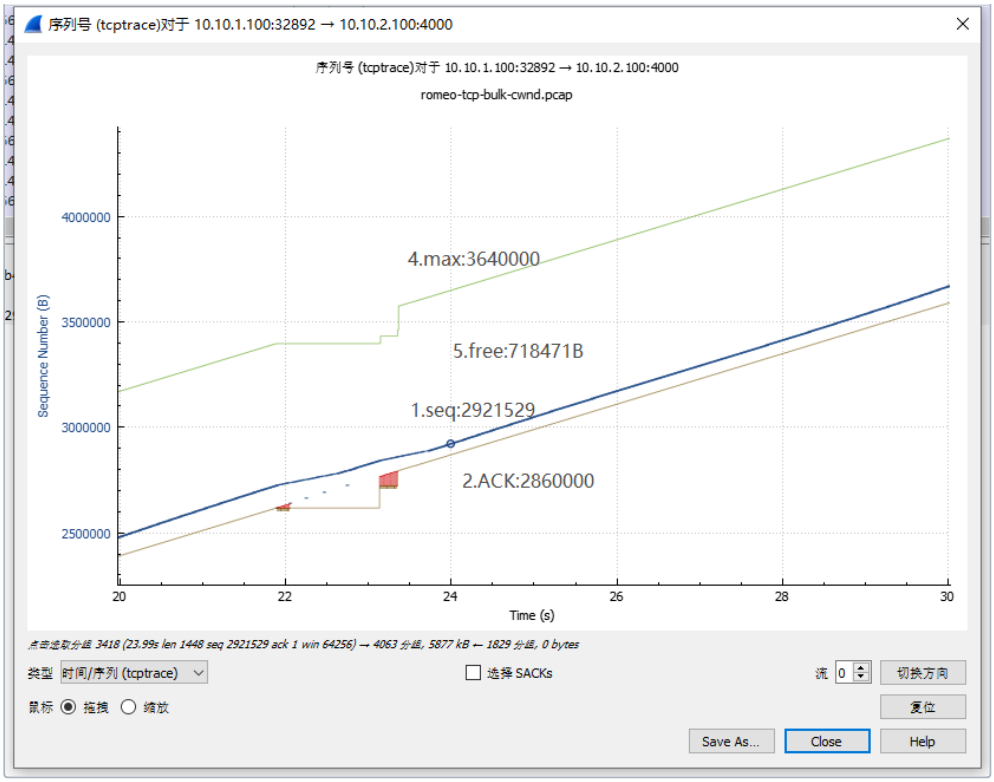
Q2.4 tcptrace plot of CWND-limited transfer

0.5 Points

On the `tcptrace` plot (from Wireshark) for this bulk data transfer, zoom in on a 10-second interval in the middle of the file transfer. For some instance of time in this interval, annotate your image to indicate

1. the (approximate) sequence number of the current segment,
2. the (approximate) ACK nummber of the most recently received ACK,
3. the (approximate) number of bytes "in flight",
4. the (approximate) maximum segment number this host can send without exceeding the other host's advertised receive window size, and
5. the (approximate) free space in the other host's receive buffer, in bytes.

▾ tcptrace.png Download



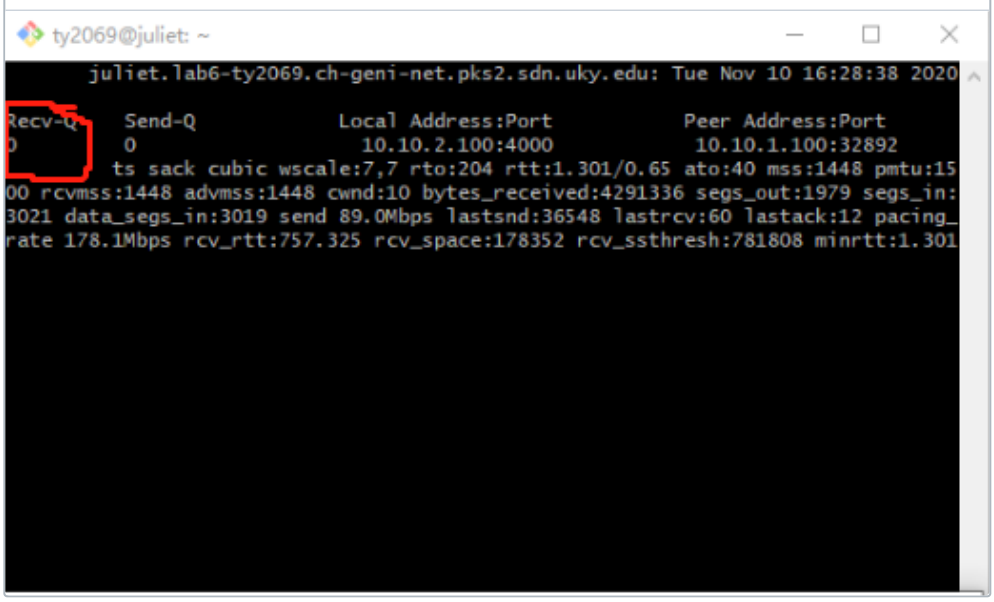
Q2.5 Queues in the CWND-limited transfer

0.5 Points

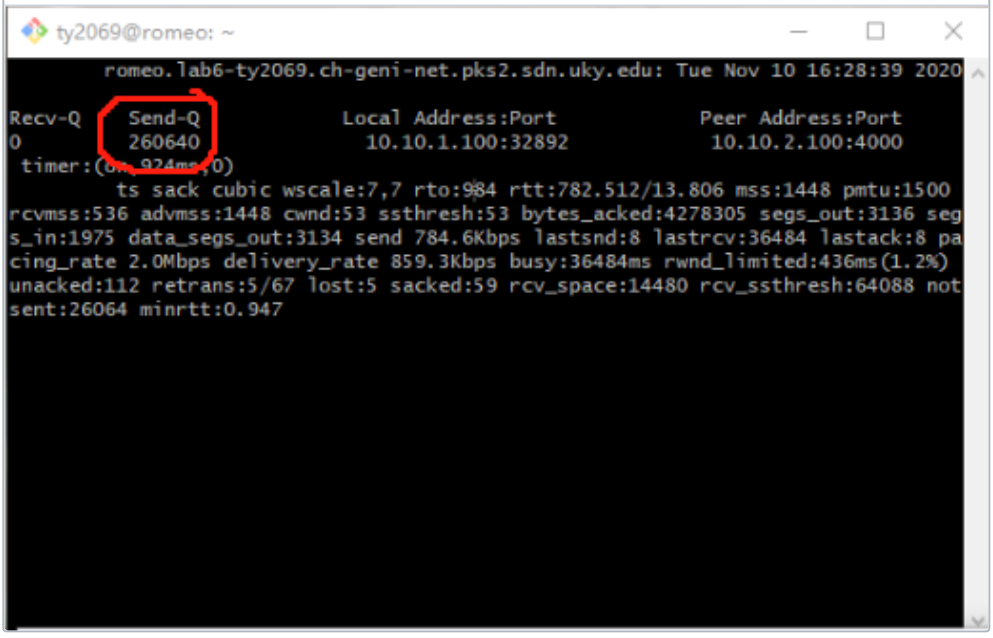
Show a screenshot of the socket queue status on both hosts in middle of the bulk file transfer. Annotate the screenshot; circle the

1. socket send buffer on the TCP sender,
2. receive queue on the TCP receiver.

receive_queue.png Download



send_buffer.png Download



How can you tell from the queue status that the connection is *not* application-limited?

the application has a large volume of data to place in the send buff er, so the connection will not be application-limited.

How can you tell from the queue status that the connection is *not* RWND-limited?

The receiver_queue is 0 all the time. So it is not RWND-limited

Q3 TCP bulk file transfer - RWND limited

1.5 Points

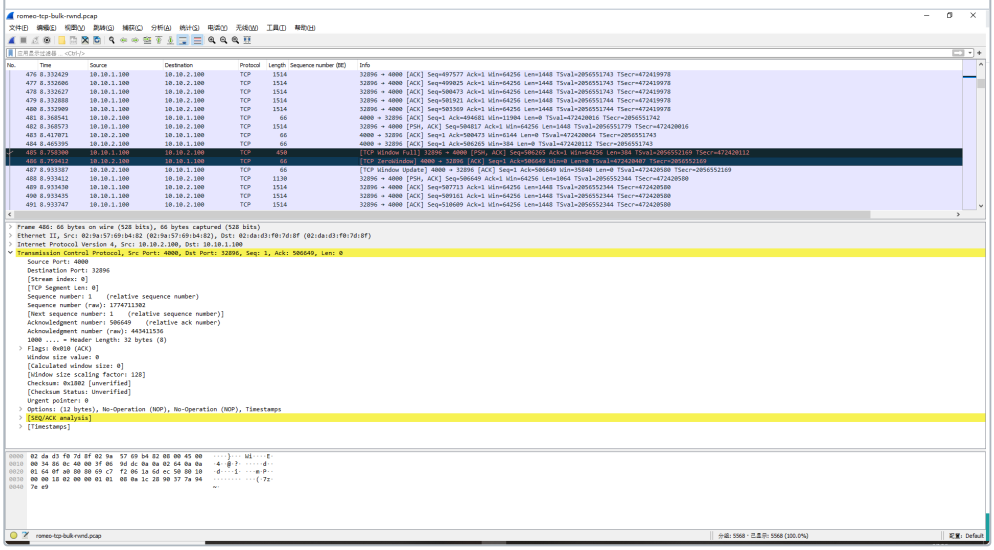
Q3.1 TCP window update

0.5 Points

When a connection is RWND-limited, the receiver may sometimes send an ACK indicating that the receive buffer has no space available (i.e. Zero Window).

Find one of these in your packet capture, and show it here.

▼ Zero_window.png Download

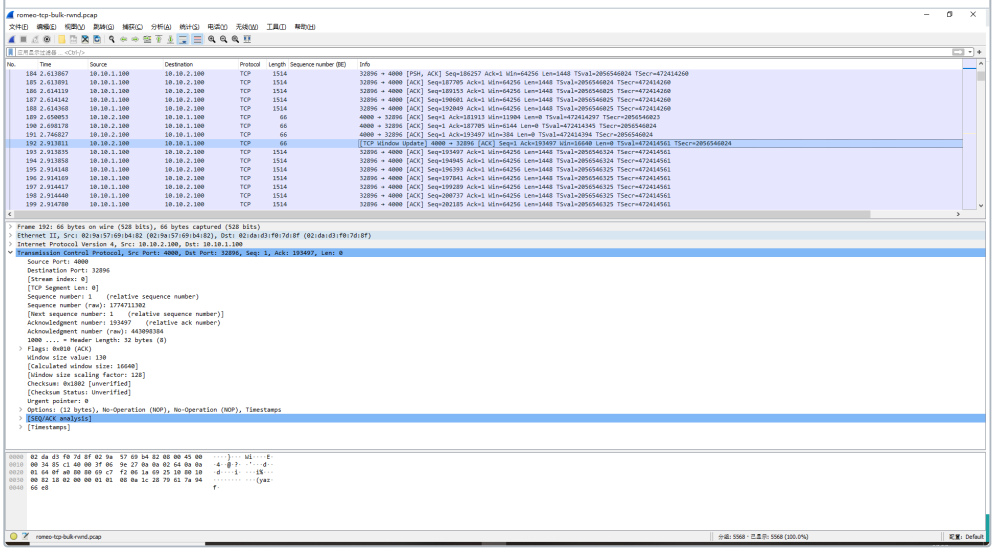


You are also likely to see some ACKs in your packet capture that are TCP Window Update ACKs.

Find one of these in your packet capture, and show it here.

Also show the ACK that is sent by the receiver just *before* the TCP Window Update.

▼ TCP_window_update.png Download



Compare the ACK number and the advertised window size in these two ACKs - the TCP Window Update and the ACK immediately preceding it. What is the purpose of the TCP Window Update? Does it acknowledge previously unacknowledged segments?

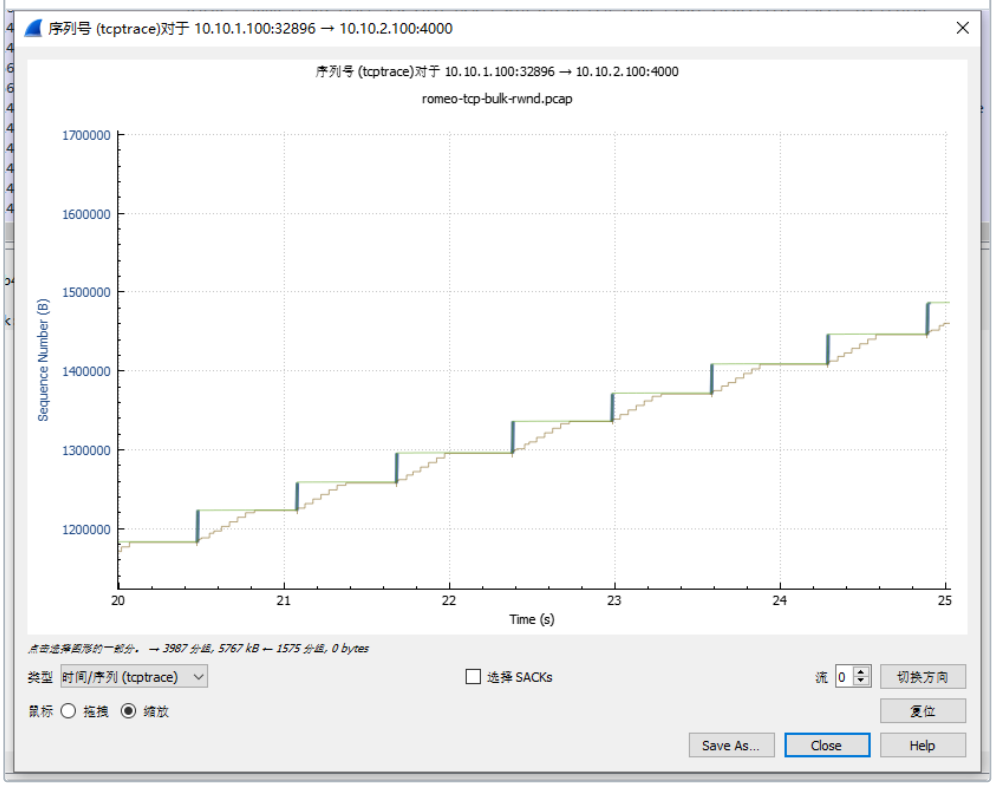
The TCP window update has the same ACK number as the previous ACK. It does not acknowledge previously unacknowledged segments. The purpose of the update is just notifying the sender that the receiver's buffer has free space.

Q3.2 tcptrace for RWND-limited transfer

0.5 Points

On the `tcptrace` plot (from Wireshark) for this bulk data transfer, zoom in on a 5-second interval in the middle of the file transfer. Upload this plot here.

rwnd_tcptrace.png Download



Explain how this image shows that the connection is RWND-limited.

In the image. The green line shows the receive buffer limit, and the blue line shows the segments it transmits. Every time the buffer has space. The segment will be put in it to reach the full size. And this process is repeated through the whole transmission. So it is RWND-limited.

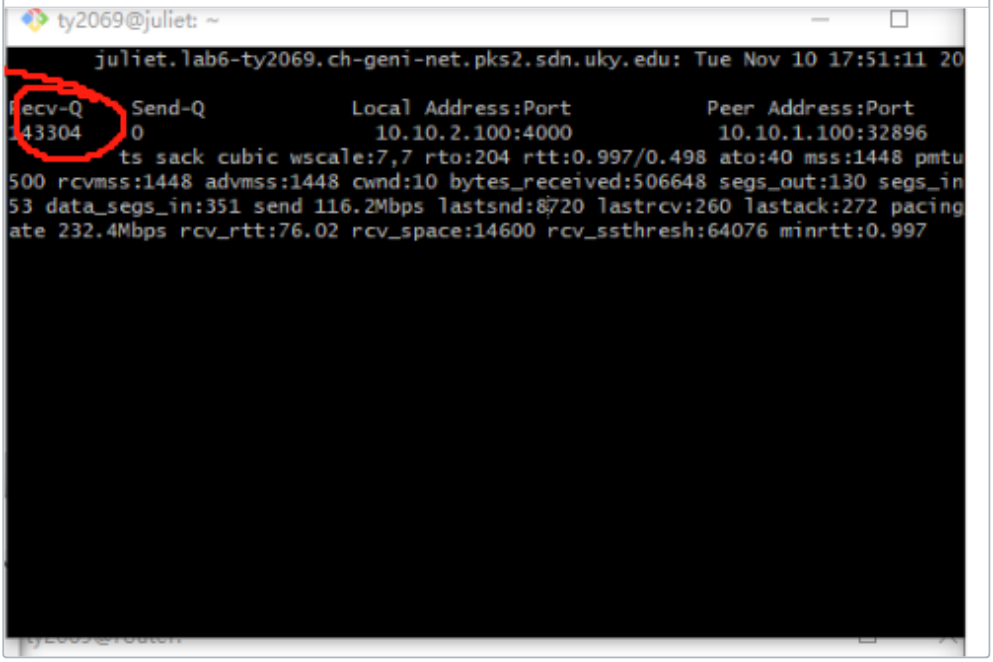
Q3.3 queues for RWND-limited transfer

0.5 Points

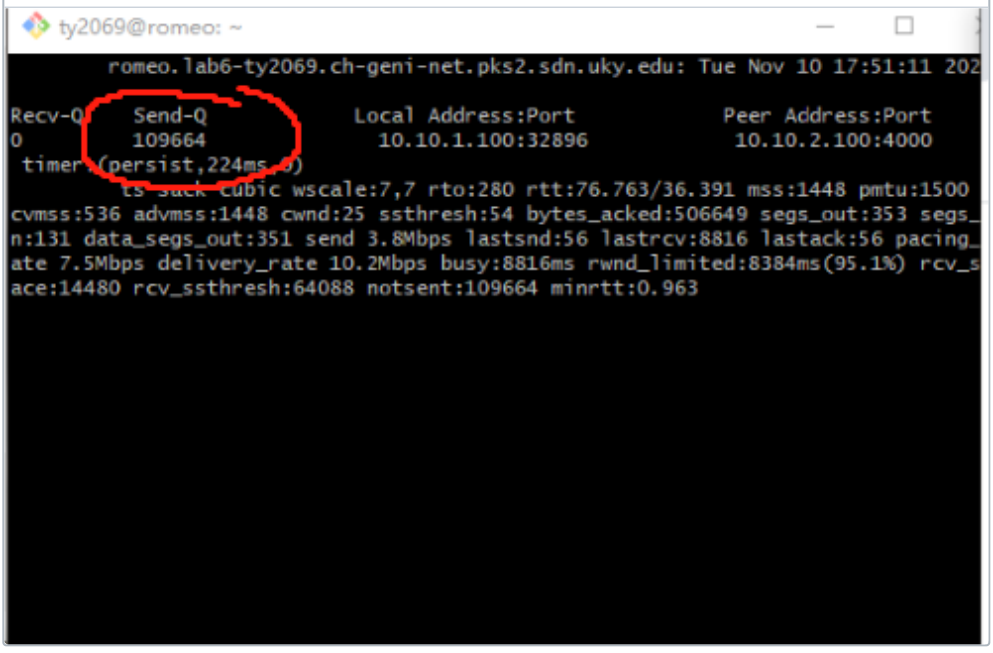
Show a screenshot of the socket queue status on both hosts in middle of the bulk file transfer. Annotate the screenshot; circle the

1. socket send buffer on the TCP sender,
2. receive queue on the TCP receiver.

▾ receive_queue2.png Download



▾ send_buffer2.png Download



How can you tell from the queue status that the connection is *not* application-limited?

the application has a large volume of data to place in the send buff er, so the connection will not be application-limited.

How can you tell from the queue status that the connection is probably RWND-limited, not CWND-limited?

We can see there is a large amount of data in the Receive queue. So it is probably RWND-limited, not CWND-limited.

Q4 TCP bulk file transfer - interrupted

0.5 Points

Q4.1 Retransmission time

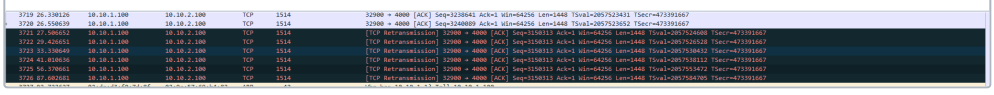
0.5 Points

In the `tcpdump` output or in Wireshark, find the part of the packet capture where the connection was broken. Examine the time difference between successive retransmissions of the same segment.

Describe how the time intervals between successive retransmissions changes during the period when the connection is broken, and show evidence from your packet capture.

from the packets time, we can see the first retransmission happen ed at time 27.5, and the second happened at 29.4 with a 2-sec diff erence. and the difference between the second and third is 4 seco nds and 8 seconds between the third and the fourth. So the time i ntervals between retransmissions is increased by 2^n where n is t he times of retransmission.

retransmit_interrupt.png Download



Q5 TCP congestion control

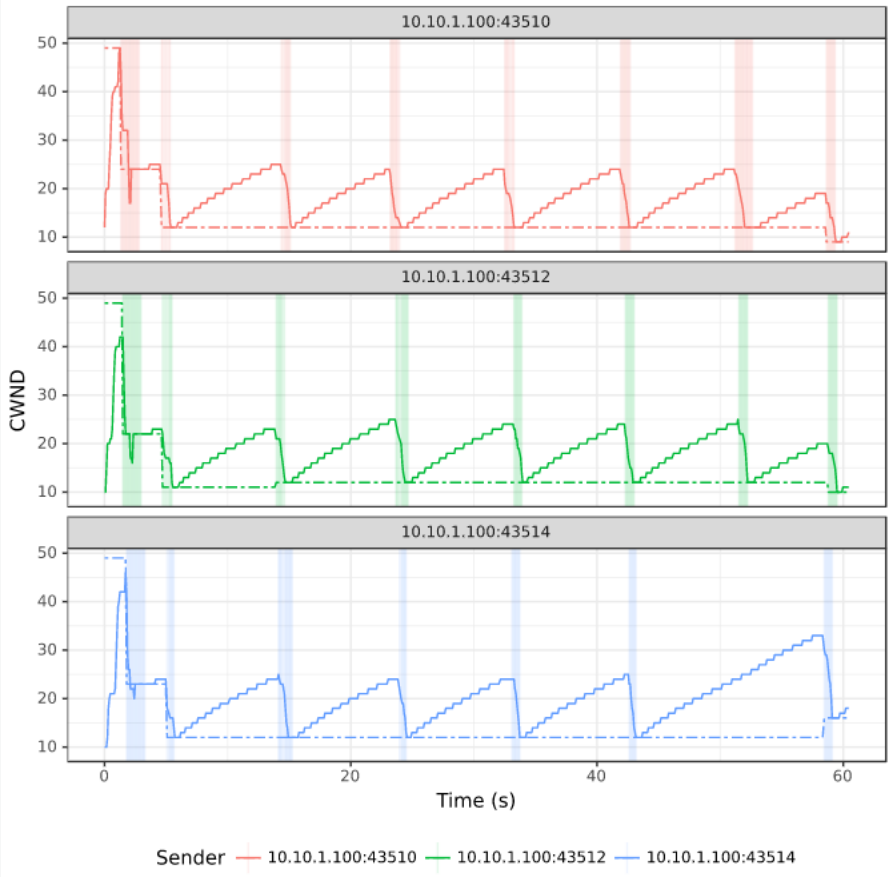
1.5 Points

Q5.1 Plot of CWND and slow start threshold for 3 TCP flows

0.5 Points

Create a plot of the congestion window size and slow start threshold for each TCP flow over the duration of the experiment.

plot.png Download



Q5.2 Annotated CWND plot

1 Point

Annotate the CWND plot for one TCP flow. Show:

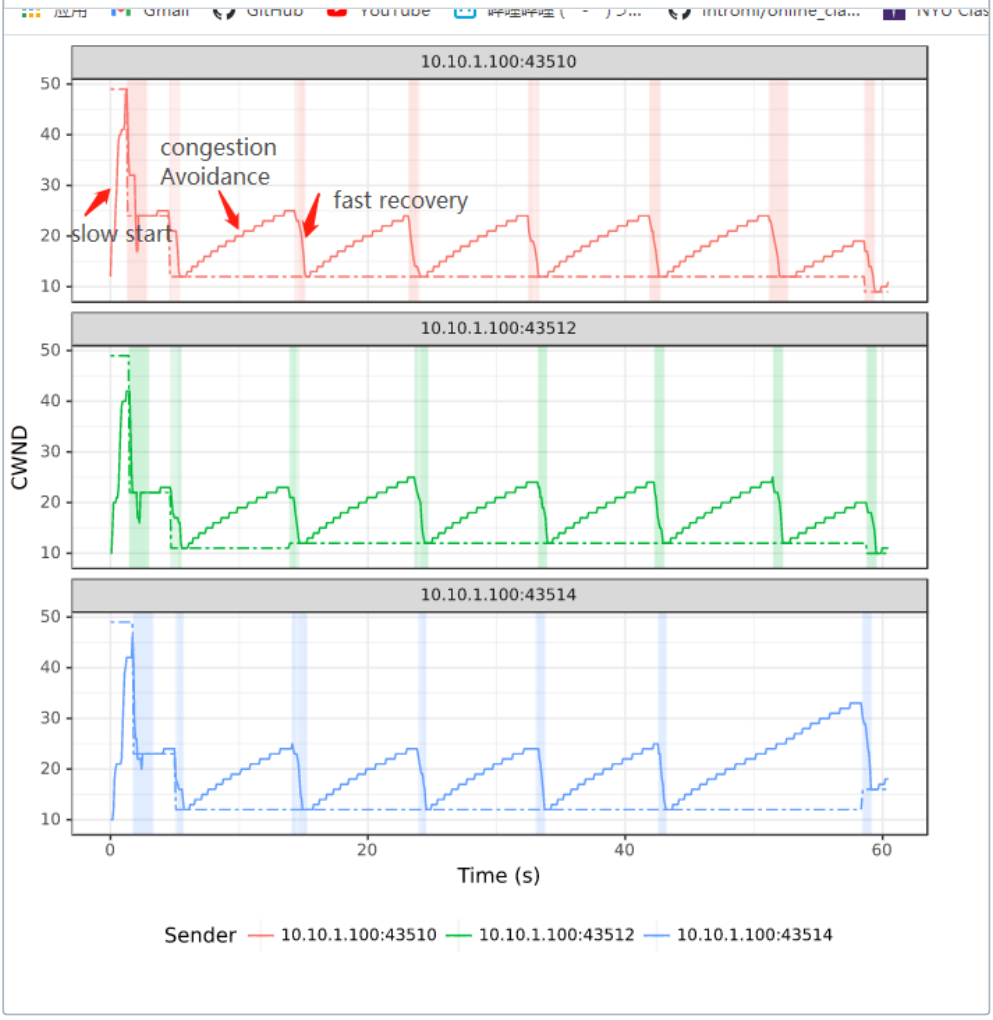
- Periods of "Slow Start"
- Periods of "Congestion Avoidance"

- Instances where multiple duplicate ACKs were received, triggering "fast recovery"
- Instances of timeout (if any)

Using your plot and/or experiment data, explain how the behavior of TCP is different in the "Slow Start" and "Congestion Avoidance" phases. Also, using your plot, explain what happens to both the congestion window and the slow start threshold when multiple duplicate ACKs are received.

In the "Slow start" mode, the congestion window increases expon entially but in the "Congestion Avoidance" mode, the congestion window increases linearly. When multiple duplicate ACKs are rece ived. the congestion window will be reduced to the slow start thre shold and the slow start threshold (dashed line) is set to half of the current CWND.

annotated.png Download



Q6 Delete your resources, please

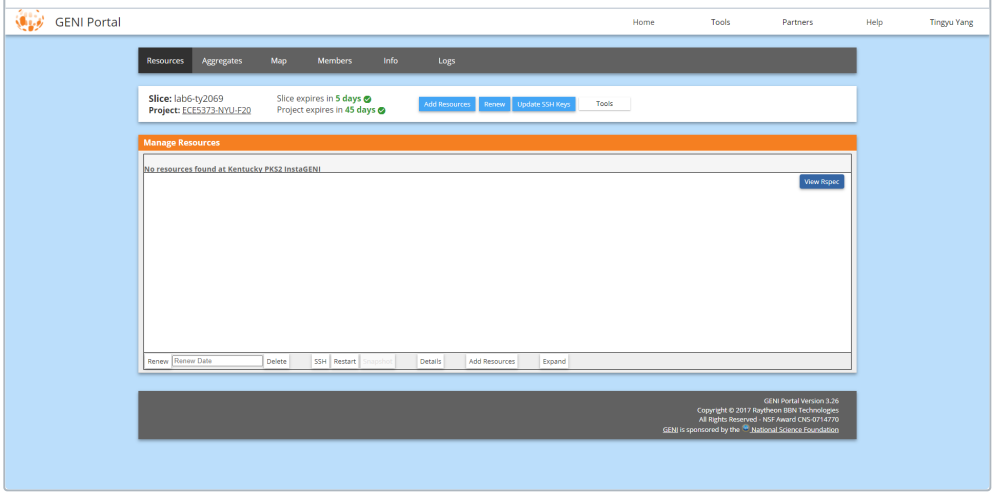
0 Points

Did you delete your resources in the GENI Portal? After you have finished submitting your answers to the questions above, delete your resources so that they will be available to other experimenters.

☑ Yes, I deleted my resources.

Upload a screenshot of the slice page for each of the slices that you used for lab 5. Your screenshots should show that there are no resources left in your slice.

delete.png Download



Lab 6: TCP

UNGRADED

19 HOURS, 3 MINUTES LATE

STUDENT

Tingyu Yang

TOTAL POINTS

- / 10 pts

QUESTION 1

TCP sockets and the TCP state diagram		3.5 pts
1.1	TCP connection refused	0.5 pts
1.2	TCP connection establishment	0.5 pts
1.3	TCP options	0.5 pts
1.4	TCP Socket API calls	0.5 pts
1.5	TCP socket connection establishment	0.25 pts
1.6	TCP socket ACKs	0.25 pts
1.7	TCP sequence numbers	0.5 pts
1.8	TCP connection termination	0.5 pts

QUESTION 2

TCP bulk file transfer - CWND limited		3 pts
2.1	TCP as a stream	0.5 pts
2.2	ACKing multiple segments in one ACK	0.5 pts
2.3	SACK	1 pt
2.4	tcptrace plot of CWND-limited transfer	0.5 pts
2.5	Queues in the CWND-limited transfer	0.5 pts

QUESTION 3

TCP bulk file transfer - RWND limited		1.5 pts
3.1	TCP window update	0.5 pts
3.2	tcptrace for RWND-limited transfer	0.5 pts
3.3	queues for RWND-limited transfer	0.5 pts

QUESTION 4

TCP bulk file transfer - interrupted		0.5 pts
4.1	Retransmission time	0.5 pts

QUESTION 5

TCP congestion control		1.5 pts
5.1	Plot of CWND and slow start threshold for 3 TCP flows	0.5 pts
5.2	Annotated CWND plot	1 pt

QUESTION 6

Delete your resources, please		0 pts