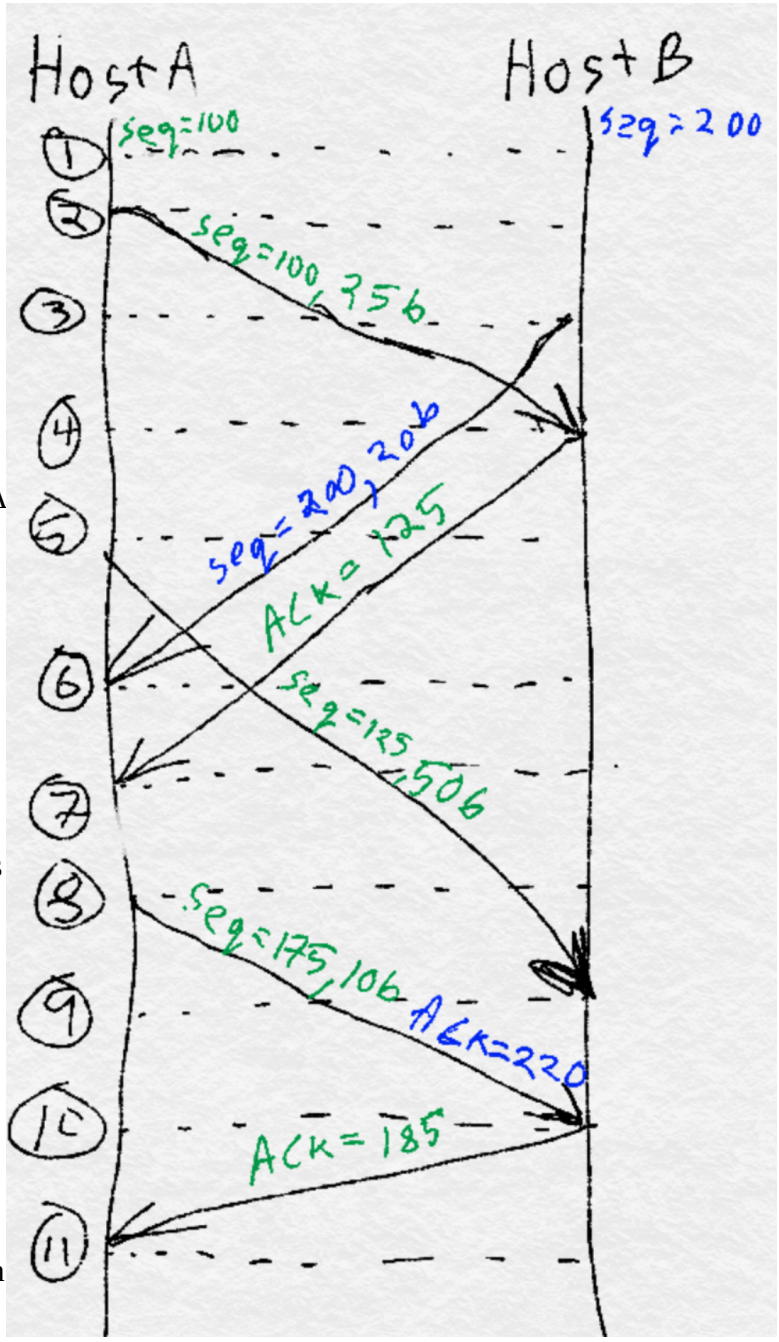
Barret Jackson
66122573
October 23, 2020

COSC 328 Lab 4

1)

Packet/Ack	**Seq#**	**Ack#**	**Size(bytes)**
Packet 1	100	200	25
Packet 2	200	100	20
ACK 1	125	125	0
Packet 3	125	200	50
Packet 4, ACK 2	175	220	10
ACK 3,4	185	185	0



For clarity, the image to the right describes the sequence of transmissions between hosts A and B. Text in green is related to transmissions from Host A to Host B (and ACKs of those transmissions from Host B to A), while text in green indicates transmissions from B to A (and ACKs from A to B).

2) a) Slow start operates from time intervals [1,6] and [23,26].
b) Congestion avoidance occurs over time intervals [6,16] and [16,22].
c) After transmission round 16, segment loss is detected by a triple duplicate ACK since the cwnd is set to the new ssthresh.
d) After transmission round 22, segment loss is detected by a timeout since the cwnd is set to 1.
e) The initial value of the threshold at the first transmission round is 32 since the cwnd doubles until reaching size 32 then increases linearly.
f) The value of the threshold at the 18th transmission round is 21 since the cwnd increased to 42 before experiencing triple duplicate ACKs, which sets ssthresh = cwnd/2 = 42/2 = 21.
g) The value of the threshold at the 24th transmission round is 13, as in f), ssthresh = cwnd/2 = 26/2 = 13.

h)

Transmission round	Segments sent
1	1
2	2-3
3	4-7
4	8-15
5	16-31
6	32-63
7	64-96

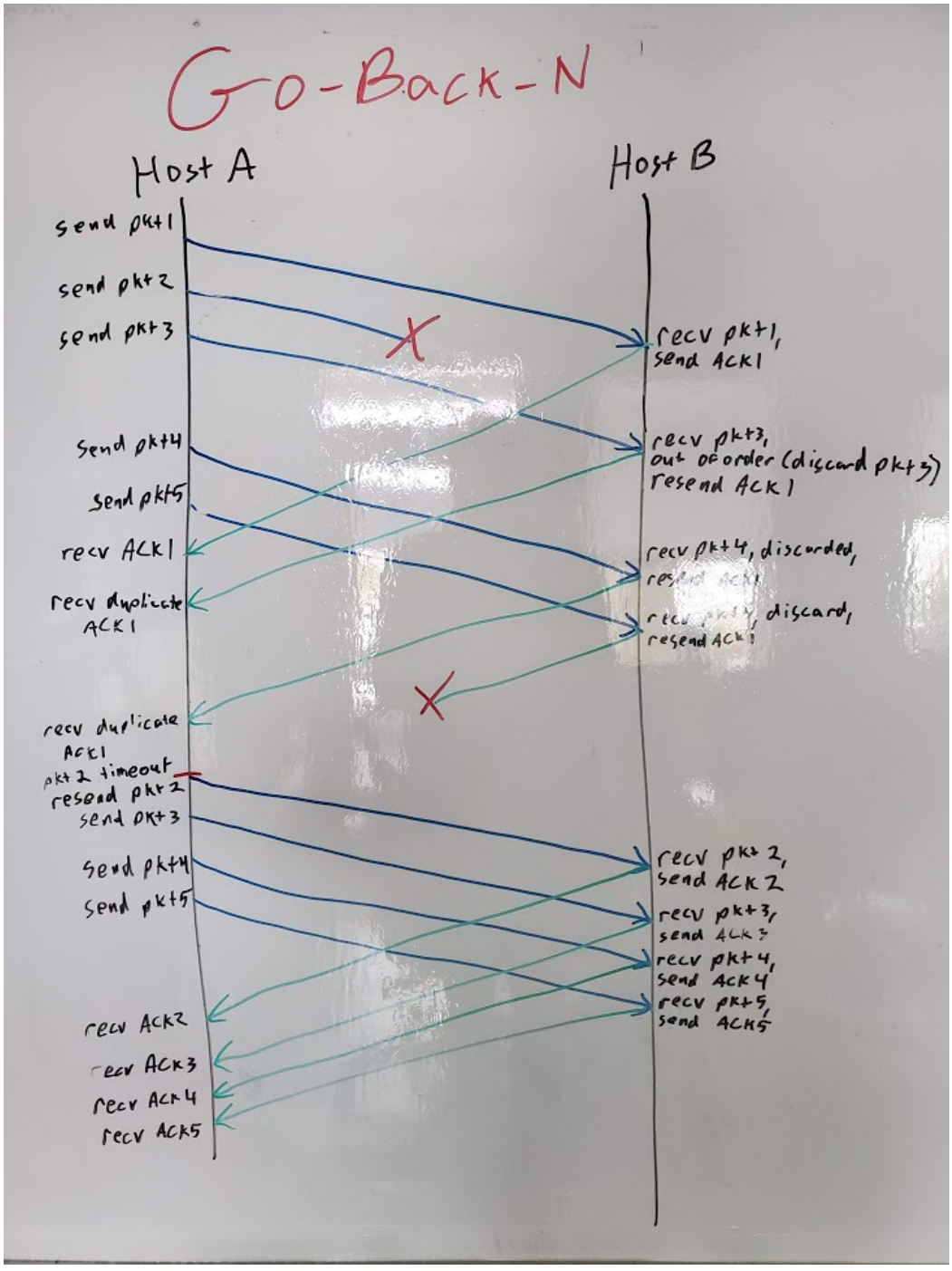
From the above table, the 70^{th} segment is sent in the 7^{th} transmission round.

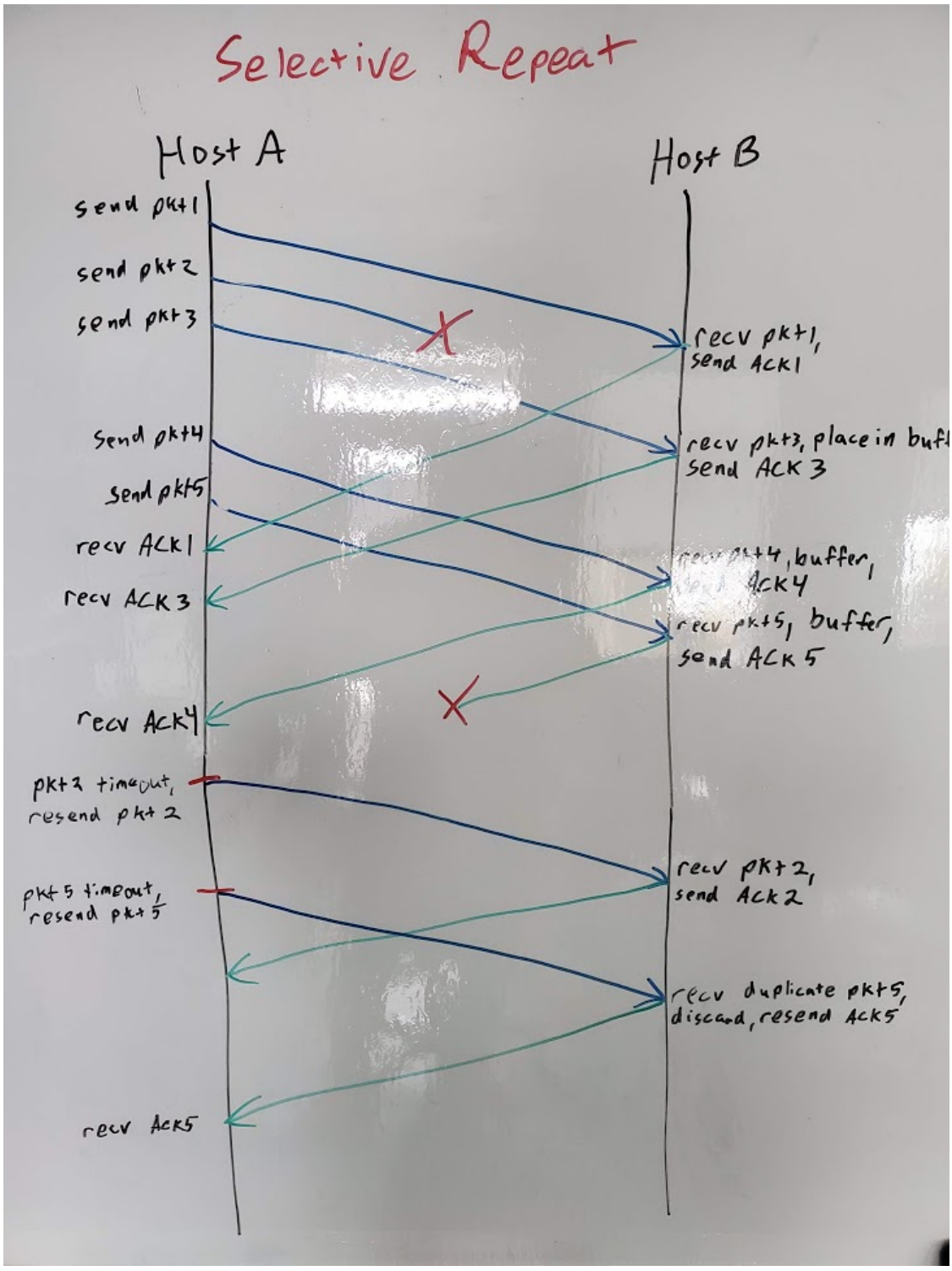
i) Since a triple duplicate ACK is detected, TCP Reno is implemented, which means the threshold is set to ssthresh = cwnd/2 = 8/2 = 4, and the cwnd is set to the new threshold, so cwnd = ssthresh = 4.

j) In the event of a timeout, TCP Tahoe is implemented, setting the threshold to the same as i), ssthresh = 4, but the cwnd is reset to cwnd = 1.

3)

With the go-back-N protocol, the sender window advances by 1 each time a new ACK is received. For example, at the beginning of this transfer, the sender window starts before pkt1. When HOST A receives ACK1 from Host B, Host A's sender window advances so pkt2 is the first packet in the sender window. Host A's sender window doesn't advance until ACK2 is received from Host B, then again with ACK3, etc.

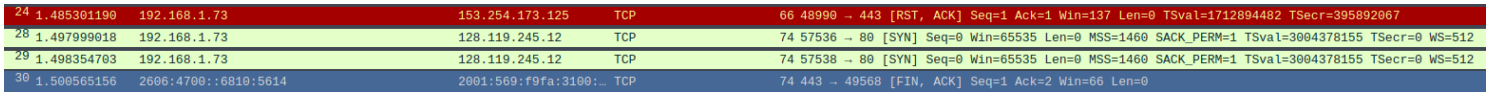




With the selective repeat protocol, the sender window once again does not advance until consecutive ACKs are received, i.e. it advances when ACK1 is received, then waits after receiving ACK 3 and 4 until ACK2 is received, then advances the window so that pkt5 is the first packet in the window and so on.

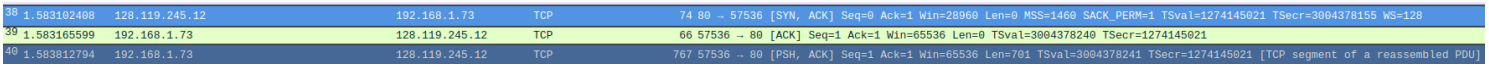
4) Wireshark!

4. The sequence number of the TCP SYN segment used to initiate the TCP connection is seq=0. Firstly, to determine which handshake corresponded to gaia.cs.umass.edu, I performed a reverse nslookup on the IP address, using *nslookup -type=PTR 128.119.245.12* and received the name gaia.cs.umass.edu as the result, confirming I had the right IP address. Next, the sequence number itself should be a hint since the first operation in a TCP connection is sending a SYN message to the server. Also, in the Wireshark info for that packet, a [SYN] flag is present, with the source being my computer and the destination being gaia.cs.umass.edu.



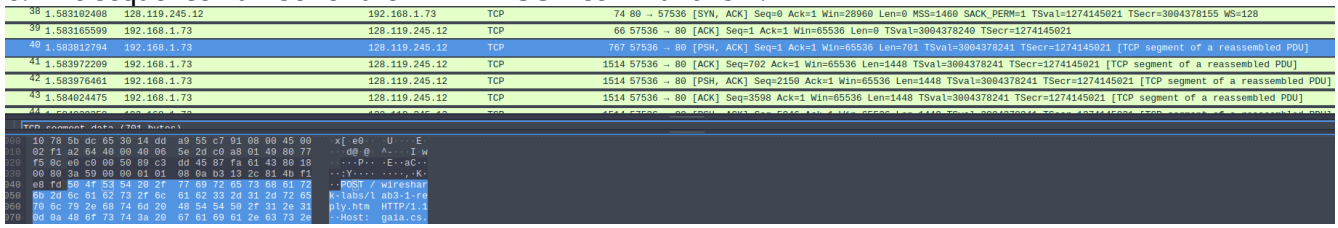
Rows 28 and 29 both indicate the SYN message from my client computer

5. With the source now being umass and the destination my computer's IP address, a [SYN, ACK] packet is sent which also has sequence number 0 in reply to my computer's SYN request. With this packet, an ACK=1 flag is sent as well. Once again, in the info section, the [SYN, ACK] flags are indicated for this row. If x is the sequence number of the client's SYN request, then the ACK message sent back from the server is x + 1, in this case, ACK = 0 + 1 = 1 as expected.



Row 38 here (highlighted in light blue) shows the SYNACK response from the server at umass

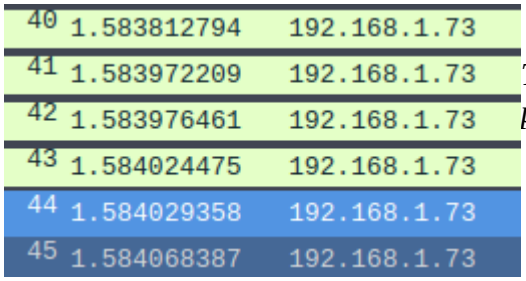
6. The sequence number of the HTTP POST command is 1.



Row 40 (highlighted) shows the POST command, as shown in the packet content at the bottom.

7. The sequence numbers of the first six packets sent from my client to the umass host with their timings are as follows: 1, 702, 2150, 3598, 5046, and 6494.

Packet	**Sequence number**	**Timing**
1	1	1.583812794
2	702	1.583972209
3	2150	1.583976461
4	3598	1.584024475
5	5046	1.584029358
6	6494	1.584068387

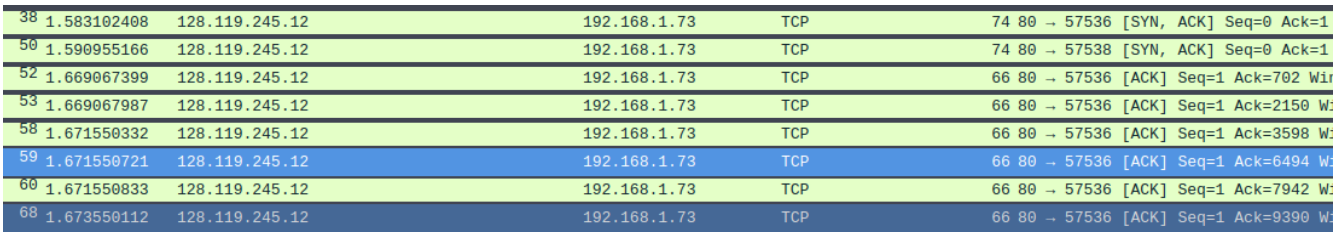


Timings for the first six data packets sent

The ACKs were received from umass at the following times:

Packet	**ACK number**	**Timing**
1	1	1.590955166
2	702	1.669067399
3	2150	1.669067987
4	3598	1.671550332
5	5046	N/A
6	6494	1.671550721

It appears as though a cumulative ACK was sent for packets 5 and 6 since there is no ACK=5046 message.



Packets 50-59 shown above are the ACKs for the first six packets (I used ip.src_host=128.119.245.12 as a filter so only packets from umass are shown)

The RTT values are given below:

Packet	**Calculation**	**RTT**
1	1.590955166-1.583812794	0.00714
2	1.669067399-1.583972209	0.08510
3	1.669067987-1.583976461	0.08509
4	1.671550332-1.584024475	0.08753
5	(1.671550721+1.671550332)/2-1.584029358	0.08752
6	1.671550721-1.584068387	0.08748

Note that since there was no ACK for packet 5, I used the average of the arrival times from packets 4 and 6 as an approximation.

The formula from the textbook for Estimated RTT is given as follows (Kurose and Ross 7th Ed):

$$EstimatedRTT=(1-\alpha)\cdot EstimatedRTT+\alpha\cdot SampleRTT$$

Using the recommended tuning parameter of α = 0.125 (from the textbook, page 281) , the estimated RTTs are as follows:

Packet	**Calculation**	**EstimatedRTT**
1	Initial value	0.00714
2	(1-0.125) * 0.00714 + 0.125 * 0.08510	0.01689
3	(1-0.125) * 0.01689 + 0.125 * 0.08509	0.02541
4	(1-0.125) * 0.02541 + 0.125 * 0.08753	0.03318
5	(1-0.125) * 0.03318 + 0.125 * 0.08752	0.03997
6	(1-0.125) * 0.03997 + 0.125 * 0.08748	0.04591