Jakub Szpunar
CS4480
WS Lab4

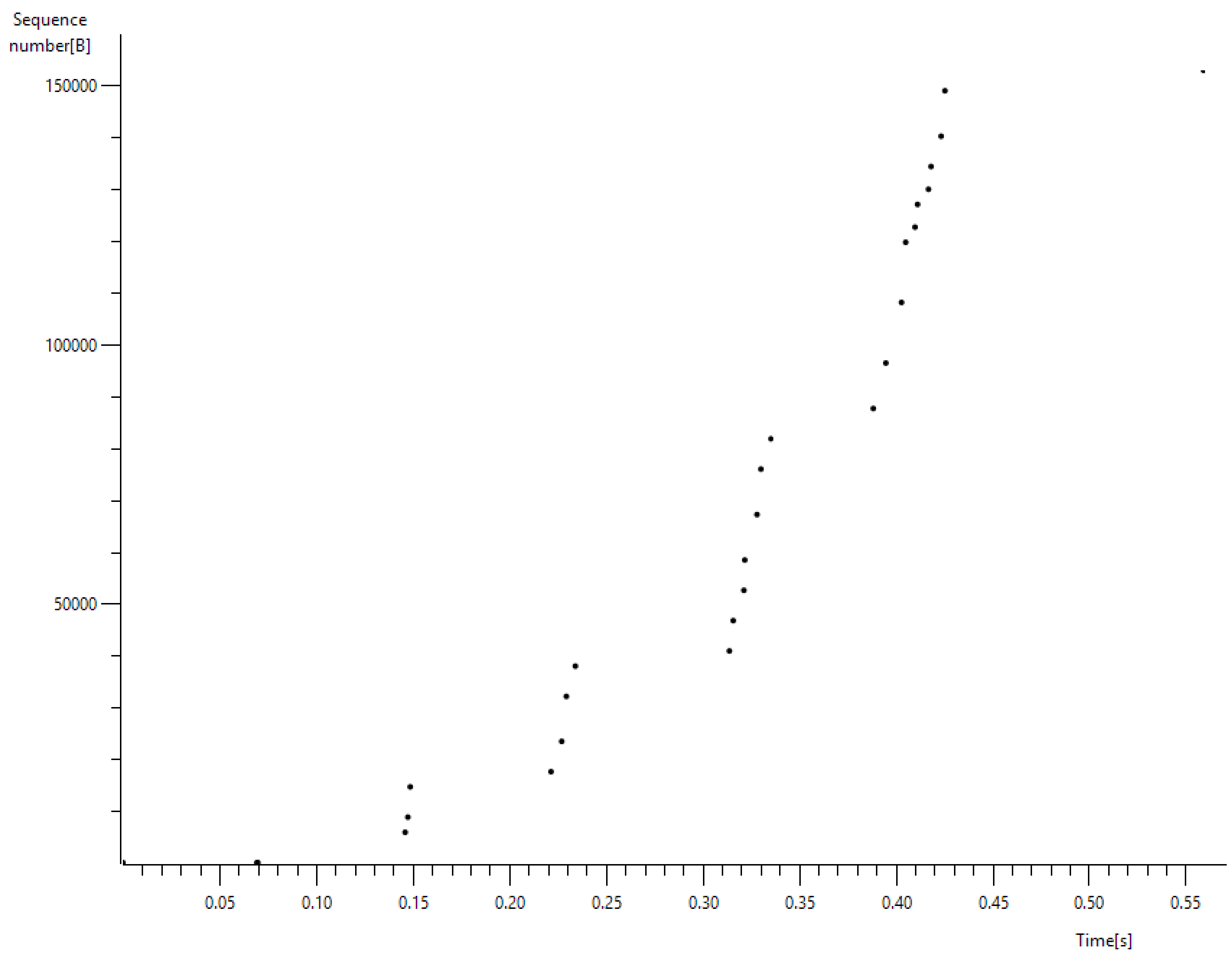
1) The source was 192.168.1.100, the port used was 1161. This information as gathered from: “Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 128.119.245.12 (128.119.245.12)” and “Transmission Control Protocol, Src Port: health-polling (1161), Dst Port: http (80), Seq: 164041, Ack: 1, Len: 50”
2) The IP address of gaia.cs.umass.edu is 128.119.245.12. This was gathered from the information given in #1.
3) My computer is 192.168.11.41 and uploaded the file via port 11301. “Transmission Control Protocol, Src Port: 11301 (11301), Dst Port: http (80), Seq: 151841, Ack: 1, Len: 1024” and “Internet Protocol Version 4, Src: 192.168.11.41 (192.168.11.41), Dst: 128.119.245.12 (128.119.245.12)” are the relevant parts of the packet to obtain this information.
4) The sequence number starts out as 0. The packet is a syn packet as the flag for syn is set. The data is gathered from: “Sequence number: 0   (relative sequence number)” and “Flags: 0x002 (SYN)” in the first packet.
5) The sequence number is 0. The acknowledgement number is 1 as expected. The value of 1 was determined by adding 1 to the sequence number of the packet it received. The flags field has both syn and ack set. Data: “Sequence number: 0   (relative sequence number)” and “Acknowledgment number: 1   (relative ack number)” and “Flags: 0x012 (SYN, ACK)”.
6) The sequence number is 1. Data: “Sequence number: 1   (relative sequence number)” from packet 4. “POST” was found in the data field for this packet.
7) Data here taken from wireshark. RTT and Est. RTT computed using excel and the formulas given in the text in 3.5.3. Sequence numbers were viewed as shown in #5. Time was determined by looking at the time-stamps in wireshark.

Packet #	Sequence #	Sent (time)	Received(time)	RTT	Est. RTT
1 (4 in ws)	1	0.026477	0.053937	0.02746	0.02746
2 (5)	566	0.041737	0.053937	0.0122	0.025553
3 (7)	2026	0.054026	0.077294	0.023268	0.025267
4 (8)	3486	0.054690	0.124085	0.069395	0.030783
5 (10)	4946	0.077405	0.169118	0.091713	0.038399
6 (11)	6406	0.078157	0.217299	0.139142	0.050992

8) The length of each of each segment is the sequence number of the following packet, minus the sequence number of the packet. It is also found in wireshark under data.

Packet #	Length
1 (4 in ws)	565
2 (5)	1460
3 (7)	1460
4 (8)	1460
5 (10)	1460
6 (11)	1460

9) 5840 was the smallest window size value I was in wireshark. The window size never got to be zero, or less than the maximum segment size. Since buffer space never ran out, throttling did not occur due to a lack of buffer space.
10) No segments had to be retransmitted. If a segment was retransmitted, there would have been two segments with duplicate sequence numbers. I didn't see any duplicate sequence numbers from the uploading client.
11) Generally 1460 bytes are ACKed. I figured this out by subtracting the ack number of the nth packet from the n+1th packet. I did find a double ACK for a packet. One packet read : Acknowledgment number: 37969   (relative ack number)", the other read "Acknowledgment number: 40889   (relative ack number)" Subtracting these numbers leads 2920 bytes of data ACKed in one acknowledgement, the equivalent of two packets worth of data.
12) The first post was sent at time 0.026477 with a sequence number of 1. The final data was sent at time 5.651141 with a sequence number of 164091. We see that a total of 164090 bytes were sent over 5.524664s. Bytes/seconds = 29700 Bytes/second.
13) The slowstart phase begins with the first packet at t=0, the phase ends when t=0.124185. This phase is characterized by an S shape in the graph. Congestion avoidance takes over with the new packets at t=0.305040. After this point, the transmission is approximately linear with bursting sends of data. The data seems to be different from the idealized behavior by being bursty with clumps of data sent at the same time instead of being more regular.
14) Below is the Time-Sequence-Graph(Stevens) for my own upload. The speed of the transfer seems to constantly increase as time goes on until it finishes. It seems TCP is additively increasing the transmission rate over the entire course of the transfer. Congestion control seems to be limited, if non-existent since the top speed is never reached. A larger download would likely have caused congestion control to occur. The sending does not look ideal because this congestion control never kicks in. However, the packets seem to be sent in a less artificially bursty was as in the previous problem.


Sequence number[B]
150000
100000
50000
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
Time[s]