Exercises for Hands-on 6: TCP

Make sure to fill in questions in the space provided underneath the question text.

Question 1: What are the IP addresses of maple and willow on this network? (Hint: Check the man page of tcpdump to discover how you can obtain the IP addresses)

Question 2: A TCP connection runs not just between two machines, but between two specific *ports* on those machines. What ports are used in the connection between willow and maple?

---

1) Using `tcpdump -r tcpdump.dat -n` we get: willow.csail.mit.edu ⇒ 128.30.4.222 ; maple.csail.mit.edu ⇒ 128.30.4.223

2) willow ⇒ 39675 , maple ⇒ 5001

Question 3: How many kilobytes were transferred during this TCP session, and how long did it last? Based on these numbers, what is the throughput (in KiloBytes/sec) of this TCP flow between willow and maple?

Question 4: What is the round-trip time (RTT) in seconds, between willow and maple, based on packet 1473:2921 and its acknowledgment? Look at outfile.txt and find the round-trip time of packet 13057:14505. Why are the two values different?

---

3) (assuming that 'TCP session' means the entire .dat file, rather than the point after the handshake)
The first packet is sent at: **00:34:41.473036**
Last packet sent at : **00:34:44.339015** ⇒ 2.865979 sec
The last sequence was 1572017:1572889 ⇒ 1,572,889 bytes
⇒ 548814 bytes/sec **⇒ 548.8 kb/s**

4) a) sent: 00:34:41.474225 ack: 00:34:41.482047 = .007822s ⇒ 7.8 ms
b) sent: 00:34:41.474992 ack: 00:34:41.499373 = .024381s ⇒ 24.4 ms

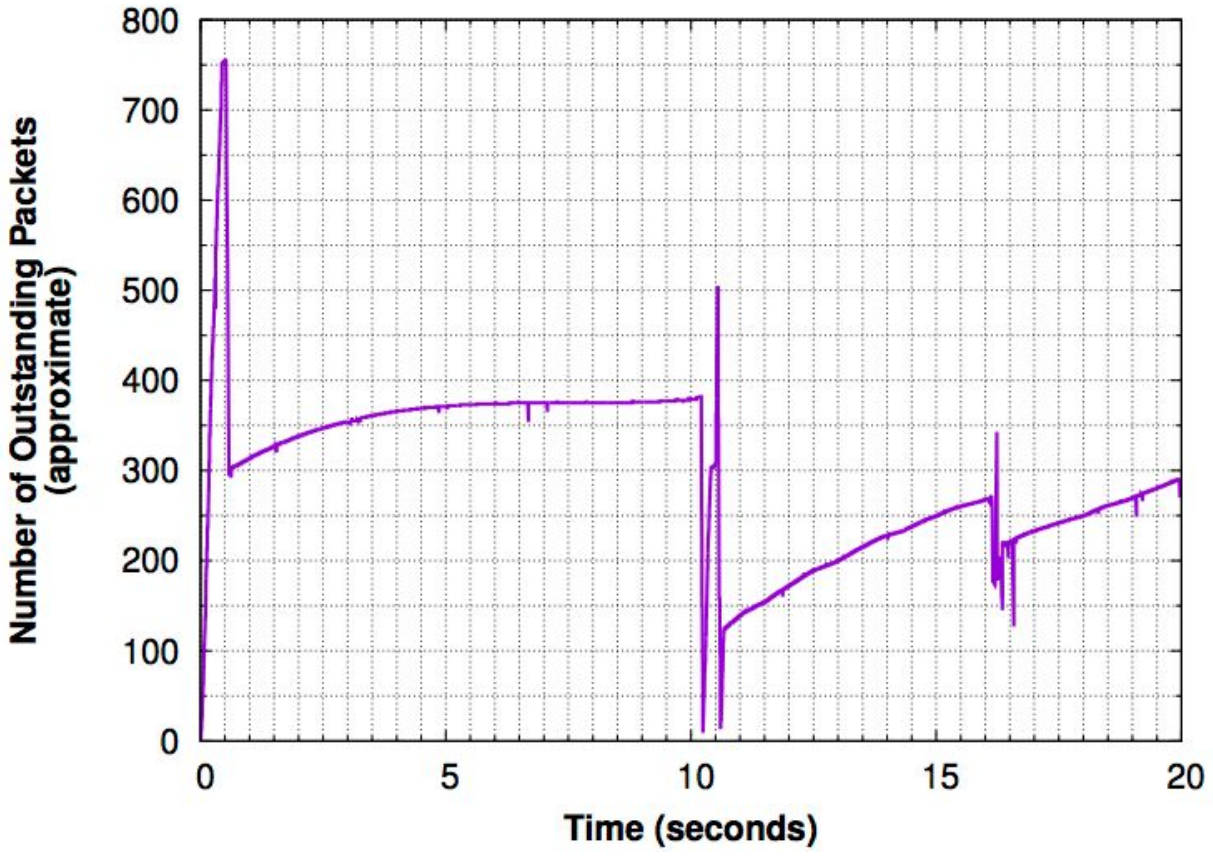
It's hard to say definitively. Networks change constantly, there could be competing traffic, more queuing, the packet could have taken a slightly different route going either direction, etc.
Also, receiving an ack is non-deterministic. The sender could receive duplicate acks for a previous packet, and re-transmission would have to occur before an ack for that packet could be sent.

Congestion Control

For the second part of the assignment, we ran a TCP connection between two different machines at MIT. These machines used a version of TCP known as TCP CUBIC. TCP CUBIC takes a similar approach to congestion control as the one you learned in lecture: senders increase their window until they see a loss, and then backoff. The only difference is in the increase function used to grow the window size. (You're welcome to dive into the details of TCP CUBIC at your own leisure, but to complete the hands-on, you do not need to know anything else about it.)

Below, we have plotted the (approximate) number of outstanding packets in this connection vs. time. All of the following questions refer to the portion of the connection shown in the plot.

You'll notice that the plot has some major trends, but exhibits small window-decreases in few places—e.g., between 6.5s and 7s and around 19s. Those small decreases are an artifact of how we measured the number of outstanding packets; you can ignore them. The questions below are concerned with the larger trends.



No questions on this page.

Question 5: Approximately how many outstanding packets could this connection support without causing queues to grow? Explain what aspect(s) of the graph support your answer.

Question 6: At what times during this connection was TCP in a slow-start phase? Explain what aspect(s) of the graph support your answer. For each phase, also explain *why* TCP was in slow-start (i.e., what event in the connection caused it to enter slow-start). (TCP CUBIC's use of slow-start is the same as what you learned in lecture.)

---

5) The graph appears to have a stable connection (with slope close to zero) at around ~375. There is a region close to this value where there is no packet loss for almost 10 seconds.

6) TCP uses slow-start when the connection is starting from a window below a certain low threshold (either at the very beginning, or after a long timeout which set the window size very low). One of the artifacts of slow-start is over-shooting a stable window size and having packet loss. First, we are in slow start from 0->1 sec, for the initialization of the connection (we can see that it largely overshoots the stable window, and has packet loss causing a large sudden decrease) and another from 10.5 -> 11 seconds (some network error caused the window size to reset to near-zero). The third ‘blip’ at around 18 sec is most likely not in slow start, as it is most likely above the threshold and is using the ‘fast recovery’ algorithm.

Question 7: At what times in the graph, if any, did packet loss likely occur? For which of these losses (again, if any) did TCP enter fast-retransmit/fast-recovery? Explain what aspect(s) of the graph support your answer.

Question 8: TCP CUBIC's increase function is slightly different than the additive increase you saw in class. If CUBIC used an additive increase function, would any portion(s) of this graph look different? If so, which portion(s), and why?

---

7) There seem to be 3 regions in which the connection experiences packet loss. First, at around 1 sec, the initial slow-start growth overshoots how much the connection can handle, and experiences packet loss. This causes the window size to drop largely and abruptly. Next, at around 10.2 sec, the connection experiences some sort of network error which causes timeouts/packet loss. Again, we can see an abrupt drop in window size, followed by slow start. I'm assuming that since the window drops to nearly zero, that the connection experienced a timeout rather than a duplicate ack, and no fast-retransmit occurred. Finally, at around 18 seconds we experience packet loss and the window size is cut quickly, the sender likely receives duplicate acks and does fast-retransmit on the segment which causes a quick spike in traffic. Rather than re-starting the window size from zero and using slow start, this time fast-recovery is used and the window size starts from a threshold above the initial window size.

8) TCP Cubic has a cubic function for increasing window size since last congestion event, rather than the linear additive increase. This causes a concave curve at first, ramping up the connection quickly, then an inflection point, and a convex curve where the connection is probed slowly before ramping up quickly. We can see in the region between 1-10 seconds, that the increase has a convex curve, then flattens out, then begins to form a concave curve before it is interrupted. In AIMD, this region would be much more linear. Also, we can see from the initial slow start that the graph jumps from 750 to 300, rather than 750/2, it's possible that this is also due to a slightly different configuration in CUBIC. Also, perhaps the 'slow start' phases would have curvier appearances as expected from standard TCP, or maybe it does and I just can't tell in this graph.