Z5117408

EXERCISE 1

Q1:

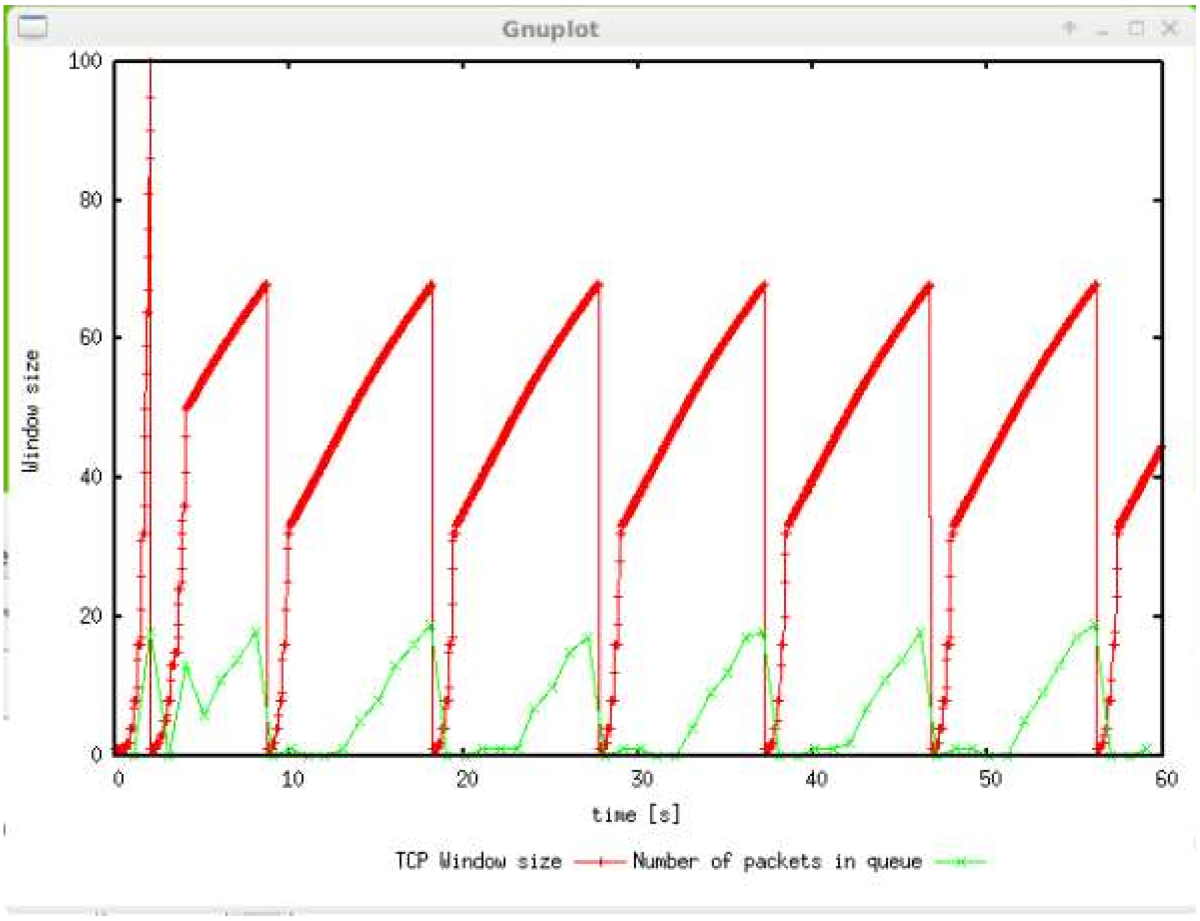
What is the maximum size of the congestion window that the TCP flow reaches in this case? 100 at time 2.0800000000000014

What does the TCP flow do when the congestion window reaches this value? Resets to a congestion window of size 1 and ssthreshold to 50 (half of congestion window at time out i.e. 100/2)

Why? Because the slow start has overshot the available network bandwidth which causes the packet buffer to fill up and then lots of packets to start dropping. Thus, it needs to drastically back off to allow the network to recover from this congestion. In TCP-Tahoe this means going back to 1 MSS on timeout. The mon file backs this up;

1.1000000000000001 ***0*** *0.0 7.0* ***0*** *11.666666666666664*
2.1000000000000001 ***35*** *0.17156862745098039 144.0* ***18*** *94.375*

What happens next? With the ssthreshold set to half the window size (at the time of the timeout i.e. 100/2 = 50), slow start beings again. Once slow start hits this updated ssthreshold, congestion avoidance mode (+1 per RTT instead of +1 per new ACK) is entered. Timeout then occurs again at around ~67 MSS congestion window, and the above repeats repeatedly after this point to give the TCP jigsaw pattern (ssthreshold ~34, congestion window around ~68 (27.719999999999498 67.9833 for example)).



Z5117408

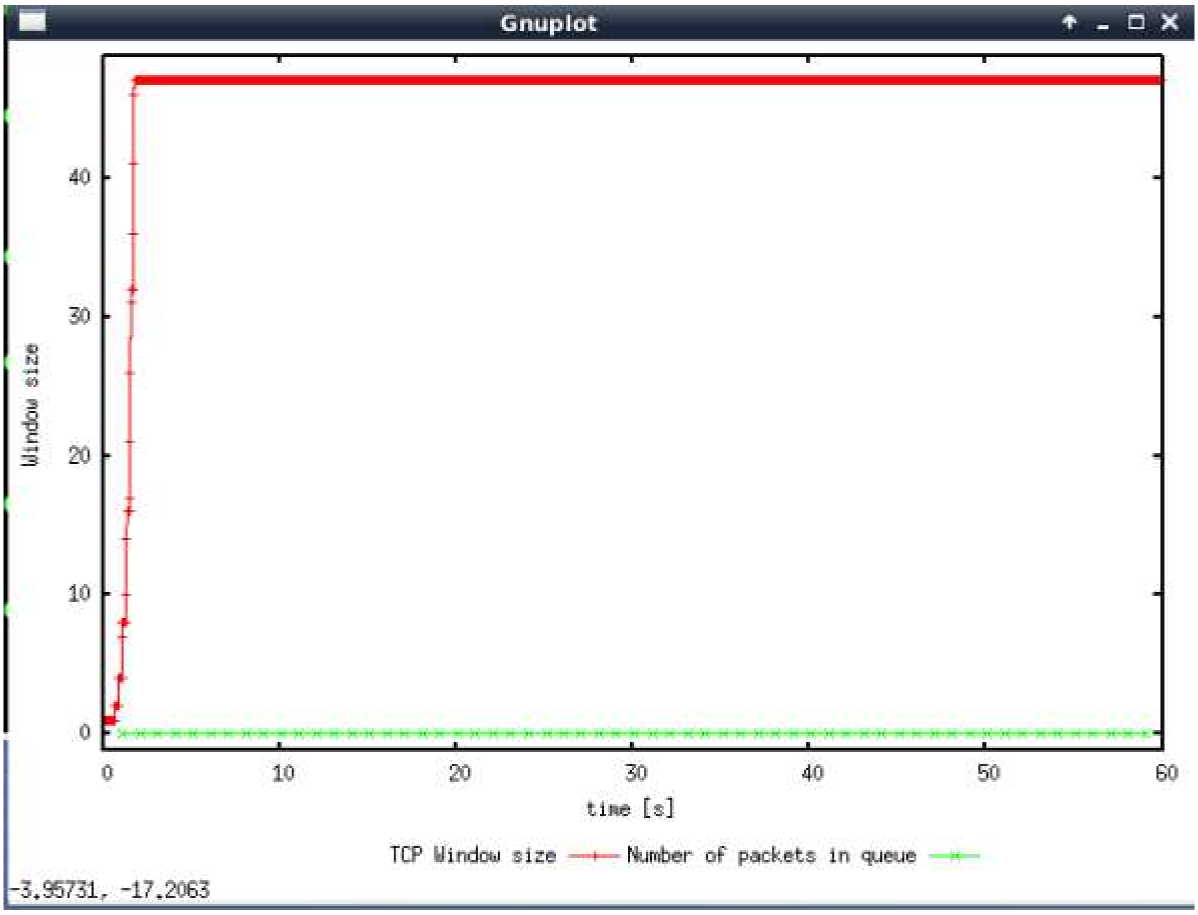
Q2:

What is the average throughput of TCP in this case? (both in number of packets per second and bps)

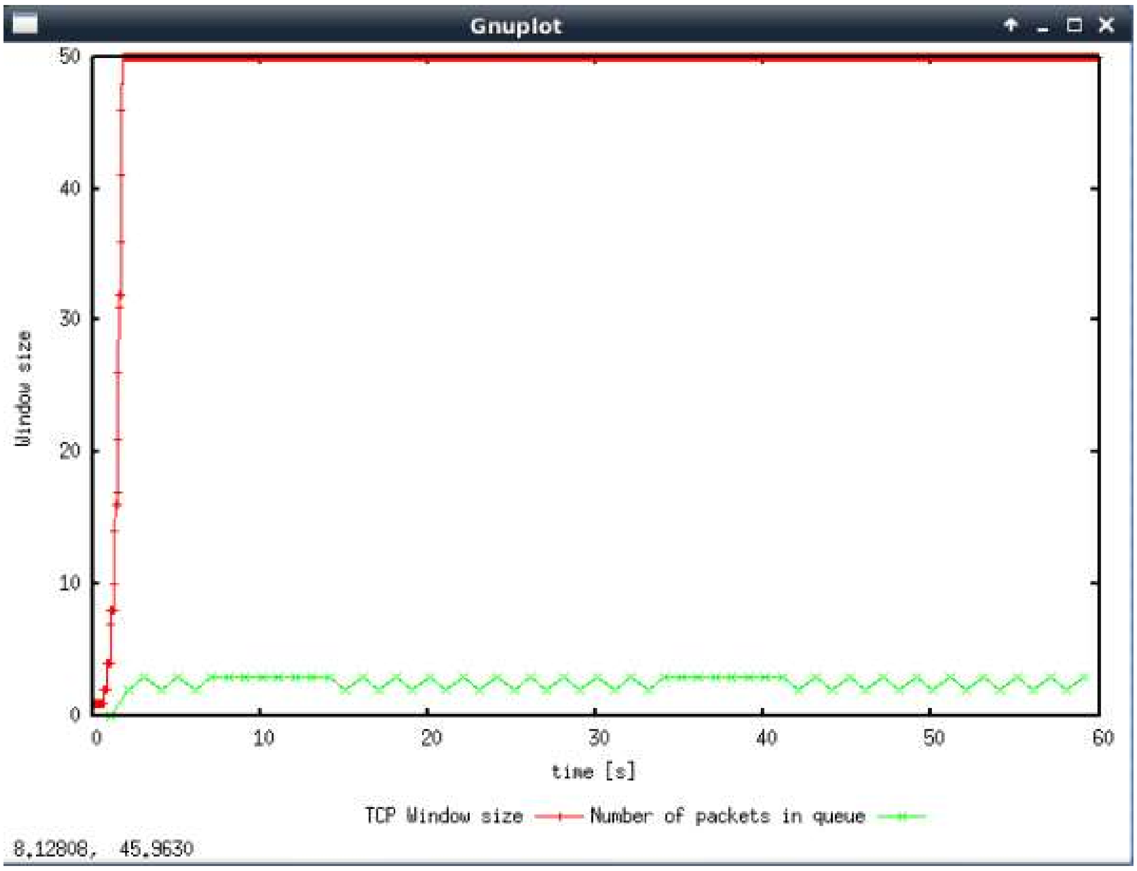
59.100000000000001 41 0.0036883771140698092 185.0 1 **188.97610921501706 pps**

and hence ~189 * 540 * 8 = **816,480 bps throughput** (w/ header vs 756,000 w/o header)

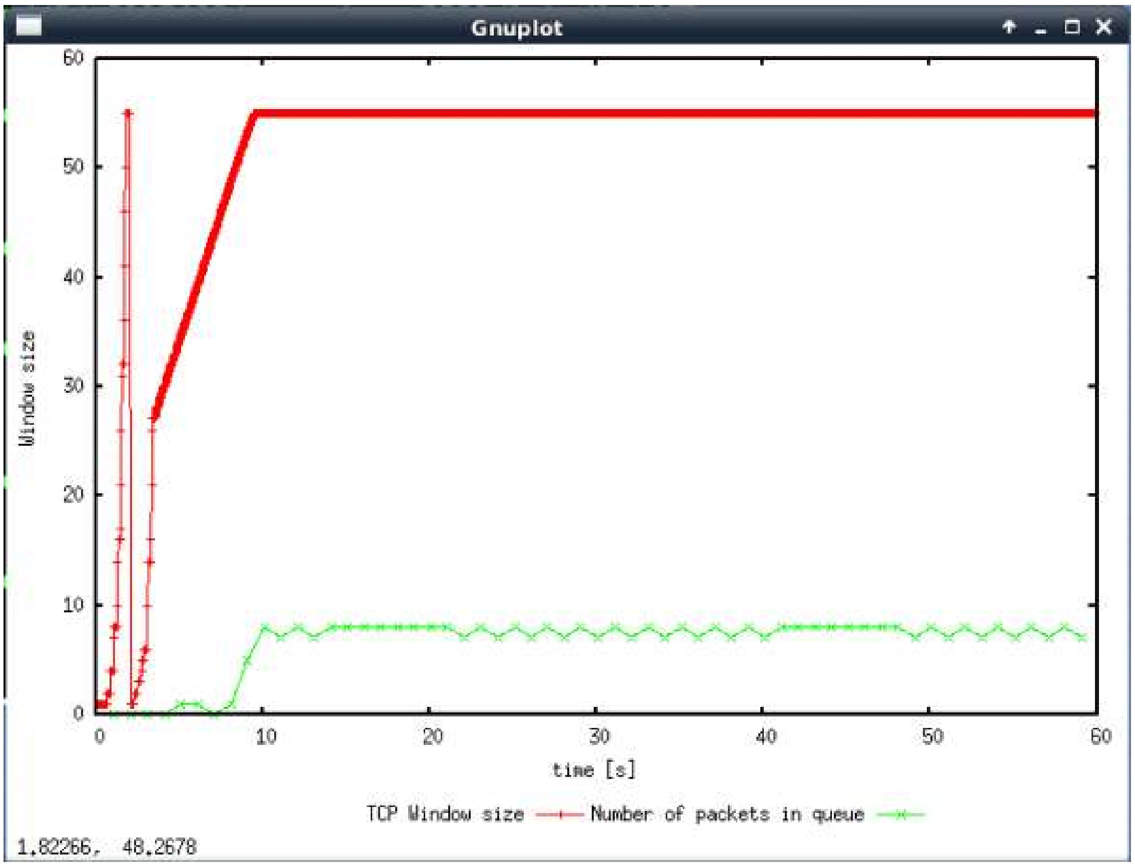
Q3:



Max window size = 47, no oscillation, no queuing

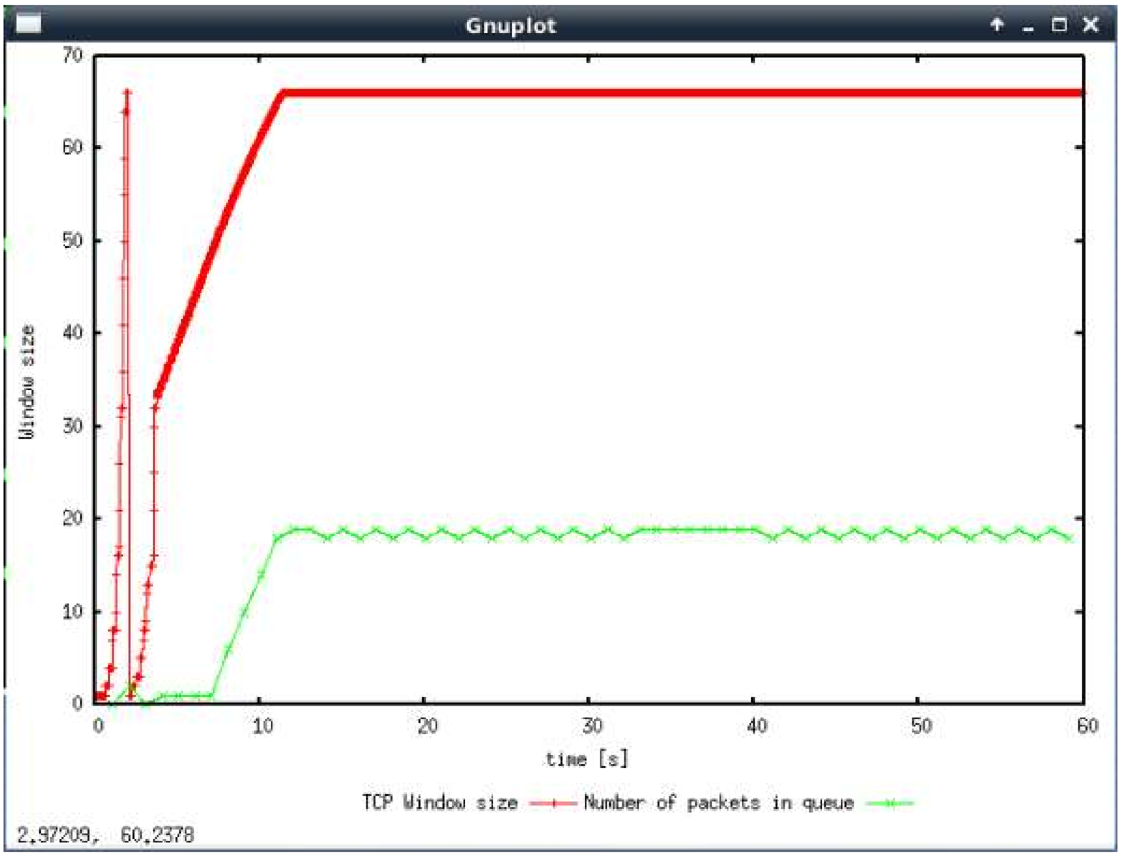


50, no oscillation, mild queuing

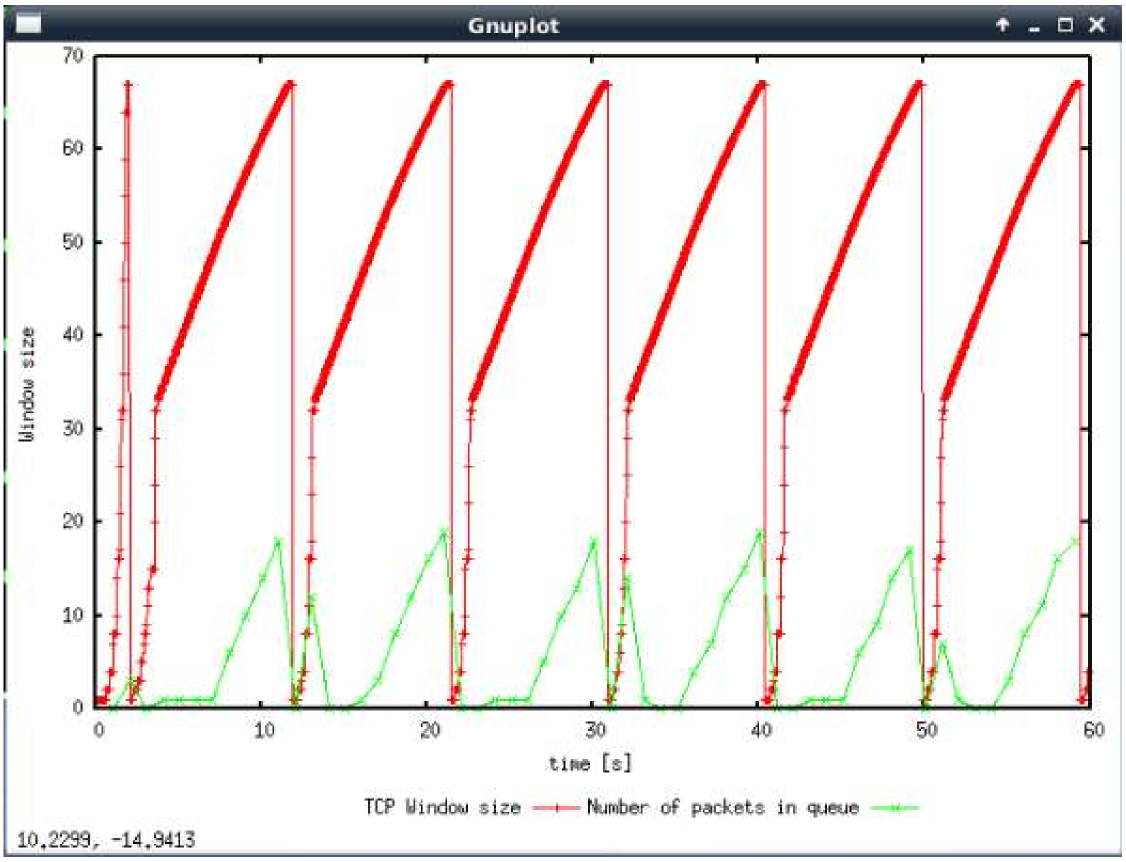


55, only a single oscillation from the slow start bandwidth estimation process, medium queuing

Z5117408



66, again only slow start oscillation, but maximum queuing



67, never stops oscillating

Rerun the above script, each time with different values for the max congestion window size but the same RTT (i.e. 100ms). How does TCP respond to the variation of this parameter? As seen from the above graphs, TCP seems to respond to increased window size by giving lots more throughput for little delay initially, but then this eventually wanes to a point where it reverses. Throughput increases greatly with little increase in congestion at the start (coming up to the knee e.g. up to ~50), but then it starts to reverse (more delay for less throughput, i.e. 55 seems to be between the knee and cliff), until eventually the throughput suffers dramatically and delays/timeouts become very frequent (66 is pretty much on the edge of the cliff, 67 has just fell off it).

Find the value of the maximum congestion window at which TCP stops oscillating (i.e., does not move up and down again) to reach a stable behaviour. 66, assuming the question does indeed mean that it "eventually stops oscillating" and not "never oscillates" (in which case the answer would be 50).

What is the average throughput (in packets and bps) at this point?

59.100000000000001 14 0.0010792476102374346 232.0 18 **220.81911262798636**

How does the actual average throughput compare to the link capacity (1Mbps)?

221 * 540 * 8 = 954,720 bps on a 1,000,000 bps link

We obviously lose out on some of the throughput due to the window being reset to 1 on timeout, and then the slow increase of the window size during the congestion avoidance phase.

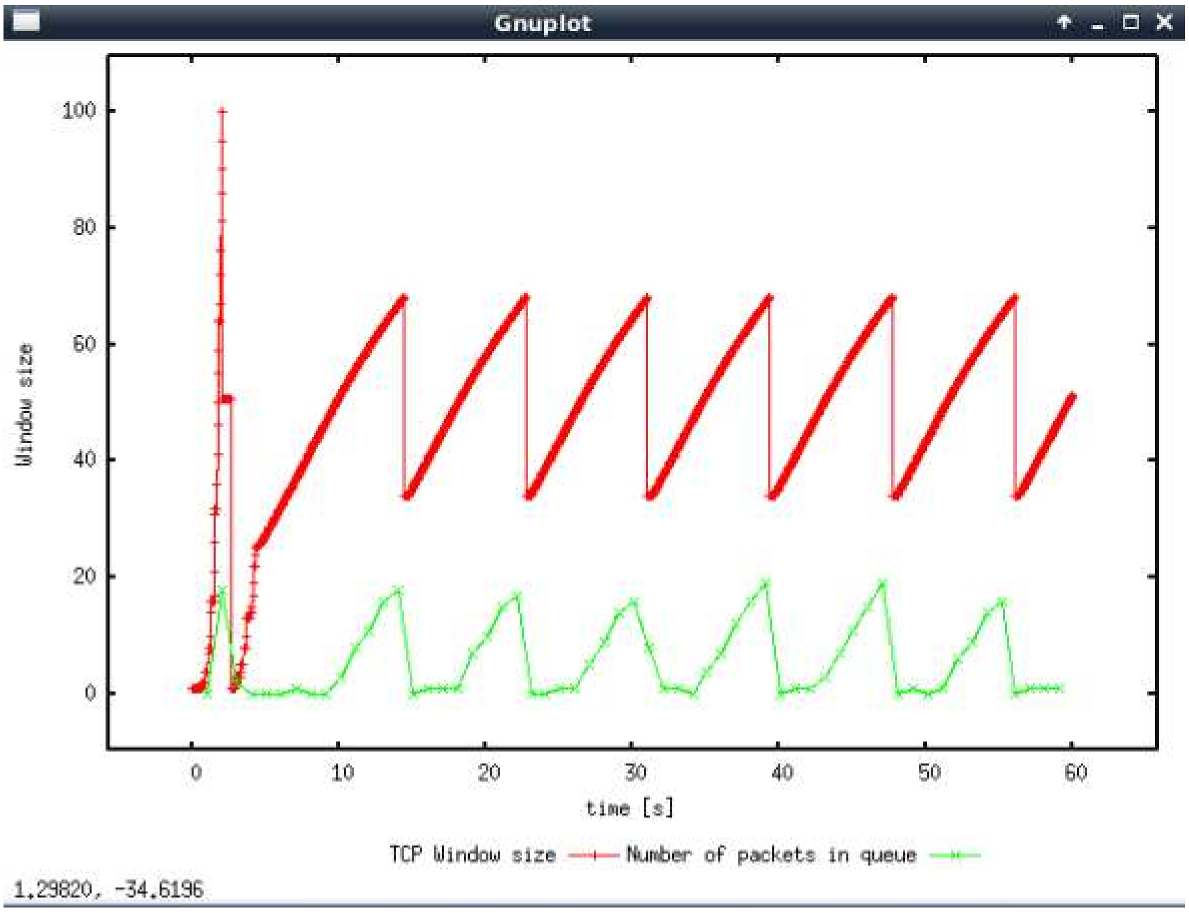
Note: we actually get higher throughput on the max window size = 50 example (this shows how the slow start timeout and congestion avoidance faze did indeed chop out some of the possible throughput). Here we still lose out on some of the throughput because the slow start still needs to build up to the window size.

59.100000000000001 0 0.0 231.0 3 **227.73037542662115 (~983, 664 bps)**

Z5117408

Q4:

Repeat the steps outlined in Question 1 and 2 (NOT Question 3) but for TCP Reno. Compare the graphs for the two implementations and explain the differences. (Hint: compare the number of times the congestion window goes back to zero in each case).



Max window is again 100 (2.0800000000000014 **100**)

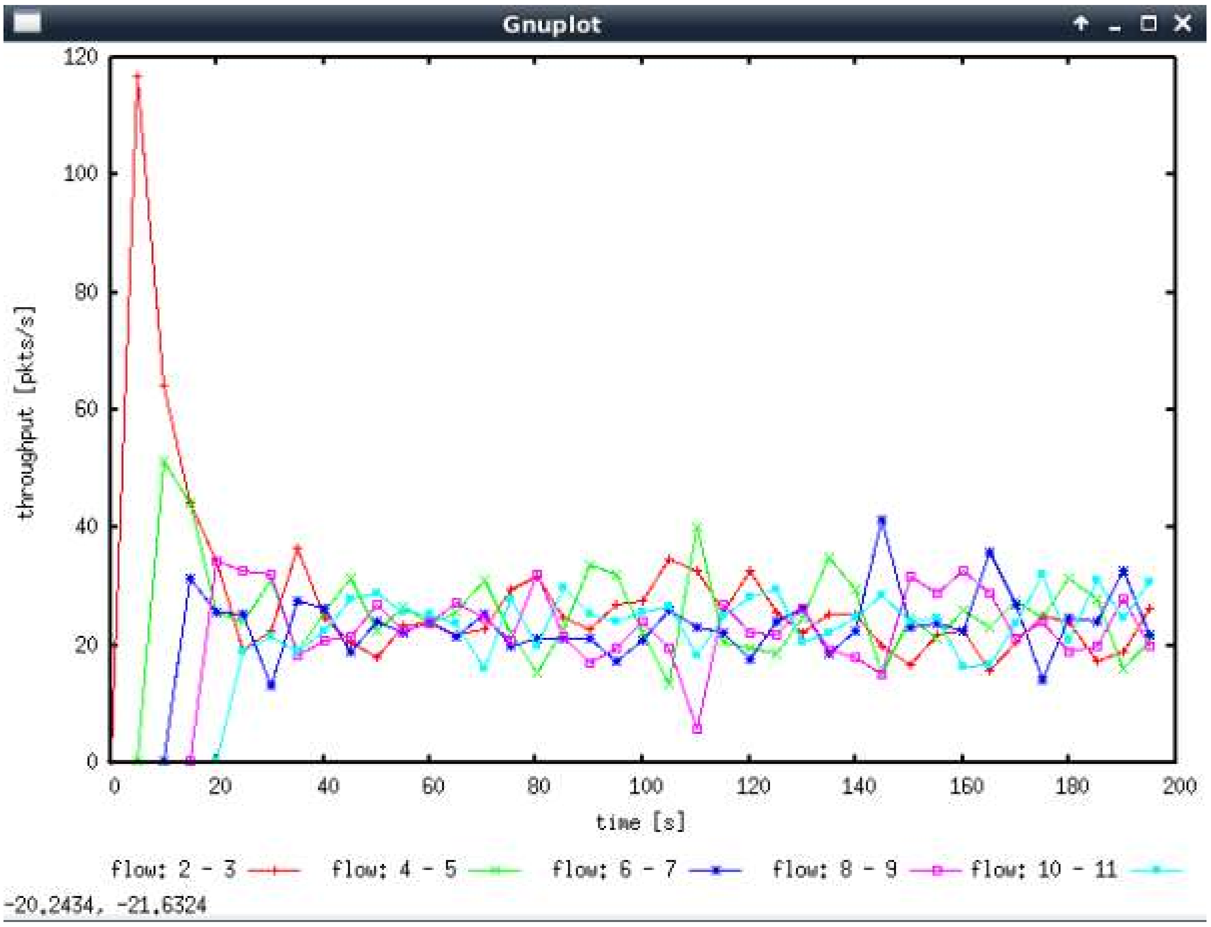
When it reaches 100, a triple dupe ACK occurred due to a dropped packet (but other packets still arriving to the receiver), and so the current window size is halved (50) and the ssthreshold set to this also (50). However, a timeout then occurs, and so the ssthreshold is halved yet again (set to 25) but the window size is set to 1 (timeout more severe than triple dupe ACK). Slow start is then initiated again until this new threshold is hit (25), upon which it goes into congestion avoidance mode. Window size then appears to increase to ~75, at which point is appears triple dupe ACK occurs again. There is no timeout and the network is able to recover and this pattern repeats to make the jigsaw pattern. TLDR: Reno only goes back to a window size of 1 once in this case (halves the size of the window instead via fast retransmit recovery option), whereas the former goes back to 1 on every timeout.

How does the average throughput differ in both implementations?

59.100000000000001 41 0.0034275204815248286 214.0 1 **203.41296928327645 (vs 188.97610921501706 pps)**

So, there was a marked increase in average throughput thanks to the added fast retransmit behaviour (namely that it prevents having to go back to a window size of 1, but instead just halves it).

EXERCISE 2



Throughput/time graph

Q1:

Does each flow get an equal share of the capacity of the common link (i.e., is TCP fair)? Yes, eventually.

Explain which observations lead you to this conclusion. The slow start for the first connection (and at that stage it is the only one) increases rapidly alongside its throughput (see graph) in its bandwidth discovery. This window size/throughput appears to be halved (tripe dupe ACK fast recovery). At around this stage, the second connection joins in but gets around half the throughput, with the original reducing throughput to half as well. Then as each subsequent connection joins in the existing flows become lower and lower, and the new ones increase until they match them. Eventually **they all converge and move up and down more or less over the same throughput range** as each other. Also, the average throughputs in the fairnessMon*.tr files are roughly the same.

Q2:

What happens to the throughput of the pre-existing TCP flows when a new flow is created? The pre-existing flows throughputs lower to accommodate for the new flow.

Explain the mechanisms of TCP which contribute to this behavior. TCP congestion control is separate in that it runs on each host machine in isolation. Hence the TCP flow relies on certain congestions events which indicate the network is beginning to be or is already congested (increasing delays, duplicate ACK's, timeouts, etc.) rather than explicitly notifying each other. In the graph, it appears that **increasing delays and the fast retransmit recovery feature are most likely accountable for indicating** to each TCP flow that the **network is becoming congested** once it is up and running because the **throughputs seem to fairly constant around a certain range** (jigsaw pattern likely indicates the window sizes being halved instead of being reset to 1 on most occasions)**.** It also seems likely there would have been some timeouts, (especially then each new flow joins in, particularly the first flow, when doing slow start for bandwidth discovery), evident by the more severe and sharper drops in throughput.

Argue about whether you consider this behavior to be fair or unfair. I think it is fair if the traffic is genuine. Each existing flow should indeed make throughput available for new flows as they come onto the network (all traffic is treated as equal priority and so the protocol of congestion avoidance itself is fair). I guess it could be leveraged in an unfair manner though if you set up multiple TCP connections over the same link to effectively get multiple shares of that shared throughput (e.g. use my laptop, computer and phone to download the 3 movies I want (1 connection per machine), rather than doing 1 at a time on 1 connection).

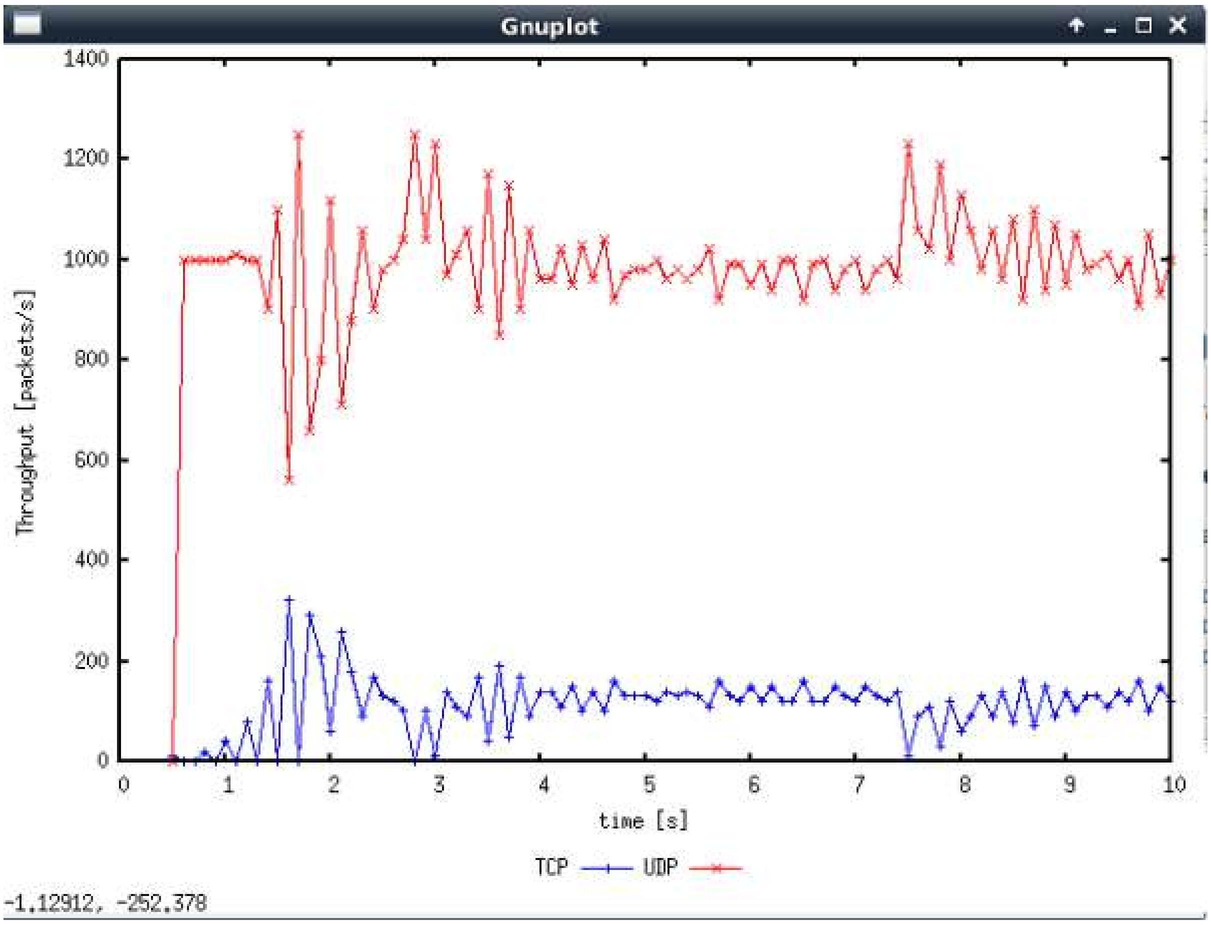
EXERCISE 3

Q1:

How do you expect the TCP flow and the UDP flow to behave if the capacity of the link is 5 Mbps?

Each source is generating data at 4Mbps. This means a total of 8Mbps trying to be pushed down a link of 5Mbps bandwidth. This is obviously a bottleneck and trying to send at this rate would cause severe packet loss and delays. The TCP source will obviously pick this up with the congestion events, and hence dial back how much it is sending. The UDP source on the other hand has no notion of congestion and will keep sending at the 4Mbps rate. It is therefore likely the TCP connection dials down around/under 1Mbps to not overwhelm the link.

Can you guess which colour represents the UDP flow and the TCP flow respectively? Going of my hypothesis I believe that the blue is TCP and the red is UDP.



Q2:

Why does one flow achieve higher throughput than the other? UDP does not take into account the congestion of the network and will just send at the rate it has been told to send at (i.e. 4Mbps on a 5Mbps link). TCP on the other hand will back off from its desired 4mbps because otherwise the network would become congested and TCP avoids this. Basically, **TCP is fair, UDP is unfair.**

Try to explain what mechanisms force the two flows to stabilise to the observed throughput.

Well on the UDP side there aren't really any "mechanisms" that force this other than it trying to force send its data at 4Mbps out onto the link (regardless of whether it is successful or not, it just does it).

On the TCP side however, it would use slow start to probe the network bandwidth, and because the UDP is already using a large portion of it, the TCP connection sets its window low. Then even if this is too high, the TCP connection will lower its window to accommodate for the UDP traffic (either halving or setting window size to 1 depending on the congestion even, fast retransmit or timeout, respectively).

Q3:

List the advantages and the disadvantages of using UDP instead of TCP for a file transfer, when our connection has to compete with other flows for the same link.

Advantages: Potentially faster if the other flows are TCP because they will back off to make sure the network is not congested (i.e. you can greedily use all the network bandwidth).

Disadvantages: Potentially much slower if all the other flows are UDP and are also trying to greedily use a large portion of the bandwidth (none of them back off, super large delays and packet loss). You would also have to implement file error detection and ordering because UDP does not offer these guarantees and they are important for files (need all bits in correct order).

What would happen if everybody started using UDP instead of TCP for that same reason?

See the first disadvantage above.