Lab 05 Report

`Lines in this font represent terminal commands and terminal output.`
Lines beginning with `$` are the terminal commands that were run.
Lines in this font are the answers to the questions.

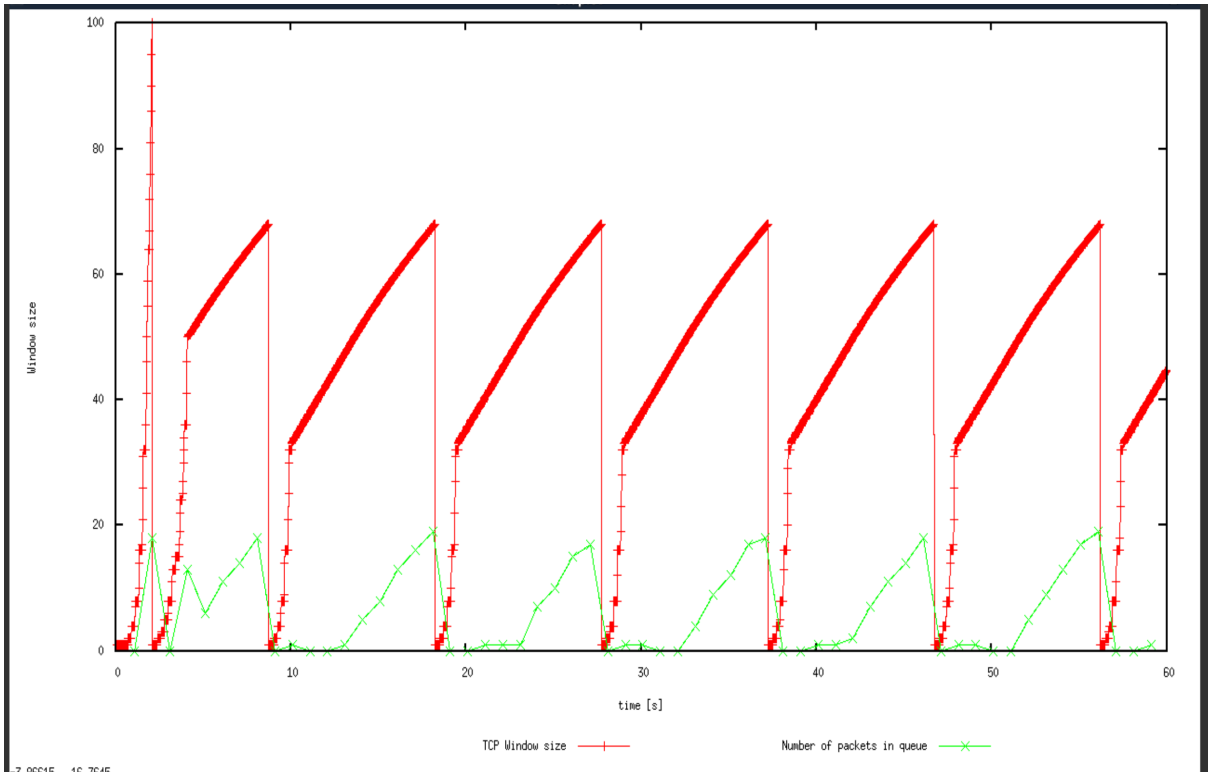
EXERCISE 1

1. (a) Maximum segment size is 67.

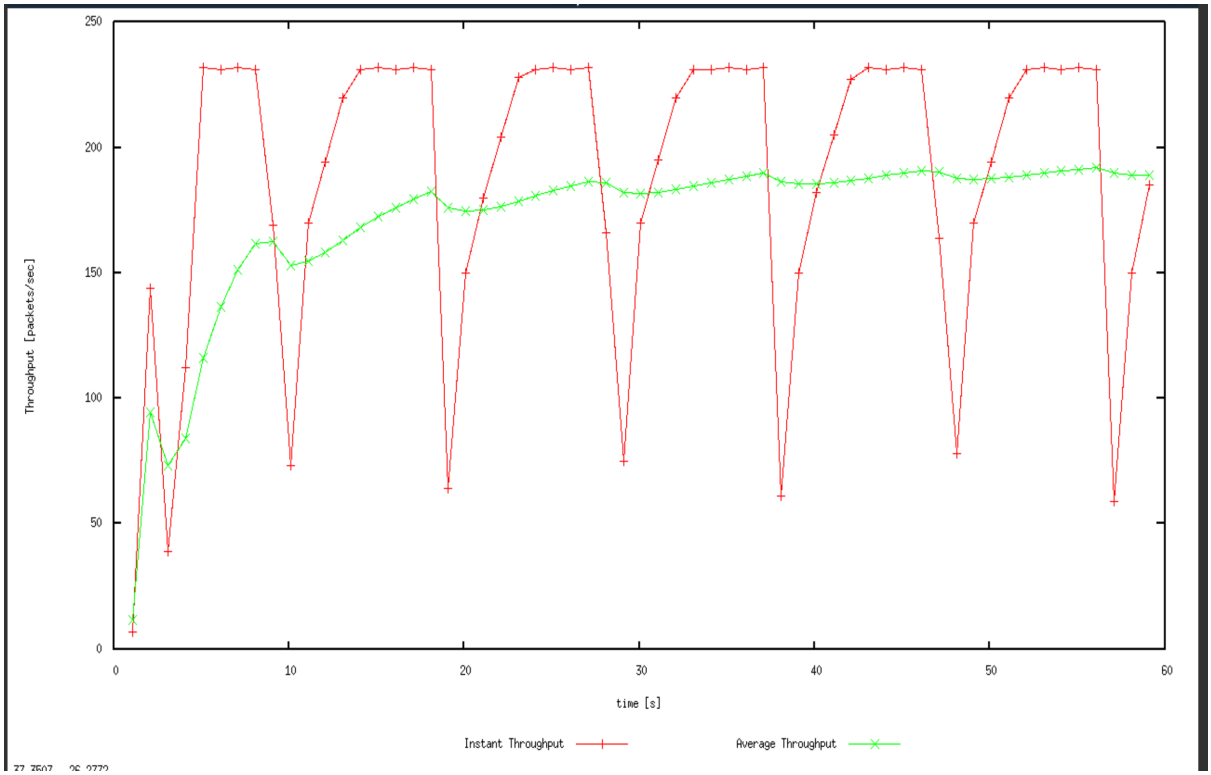
   (b) When the congestion window size reaches 67, the CWND is reset to 0.

   (c) TCP Tahoe resets the CWND to 0 upon any packet loss or timeout event.

   (d) After CWND is reset to 0, due to packet loss or timeout, TCP Tahoe returns to slow start, until the slow start threshold is reached. After this threshold has been reached, TCP Tahoe transitions to AIMD for increases. This accounts for the intial exponential increase, followed by linear increase until 68 is reached for CWND.



2. Average throughput was 188.976 packets per second. With 540 bytes of data per packet, due to the TCP and IP headers, and 8 bits in a byte, the average throughput becomes 816376.32 bits per second.



3. (a) Setting the maximum segment size to 66 gives no oscillation, that is, stable behaviour for the size of CWND. This is expected based on the answer to Q1.

   (b) Average throughput in this case is 220.819 packets per second. Converting this to bits per second, we get 953938.08 bits per second.

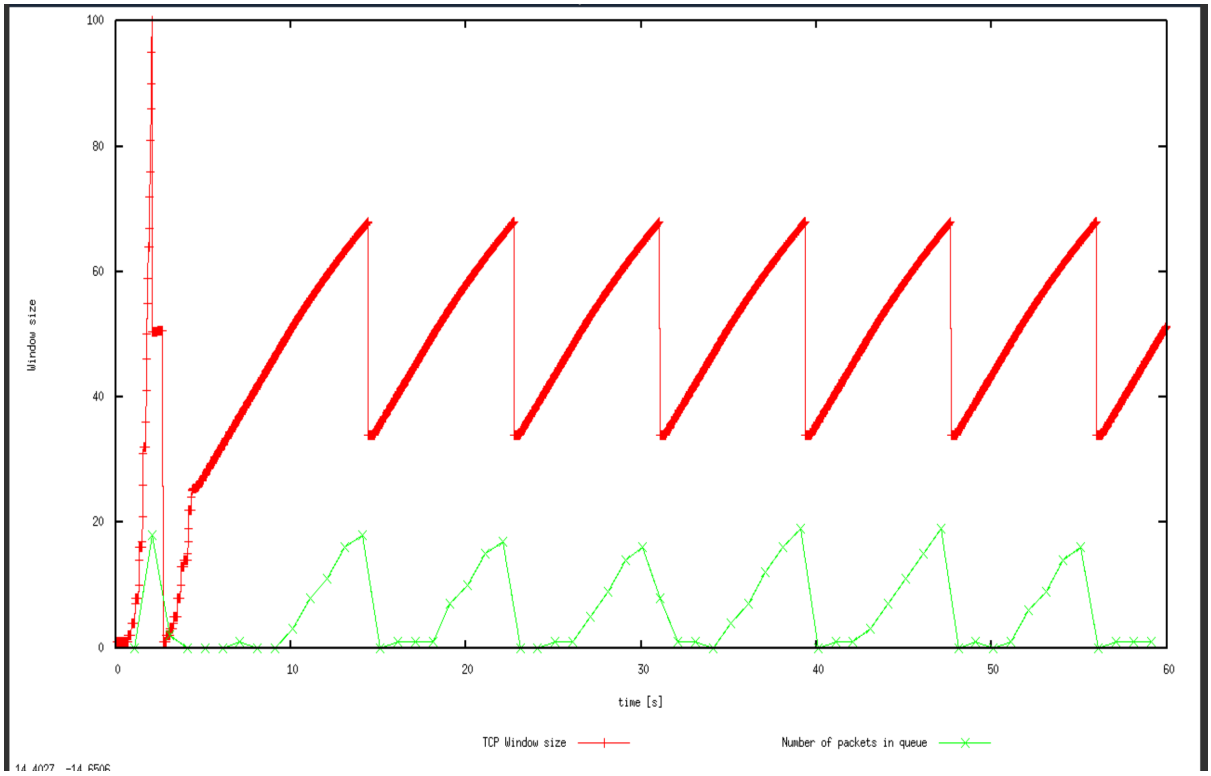
   (c) This throughput is approximately 95.4% of the link capacity.

4. 1. (a) Maximum segment size is 67

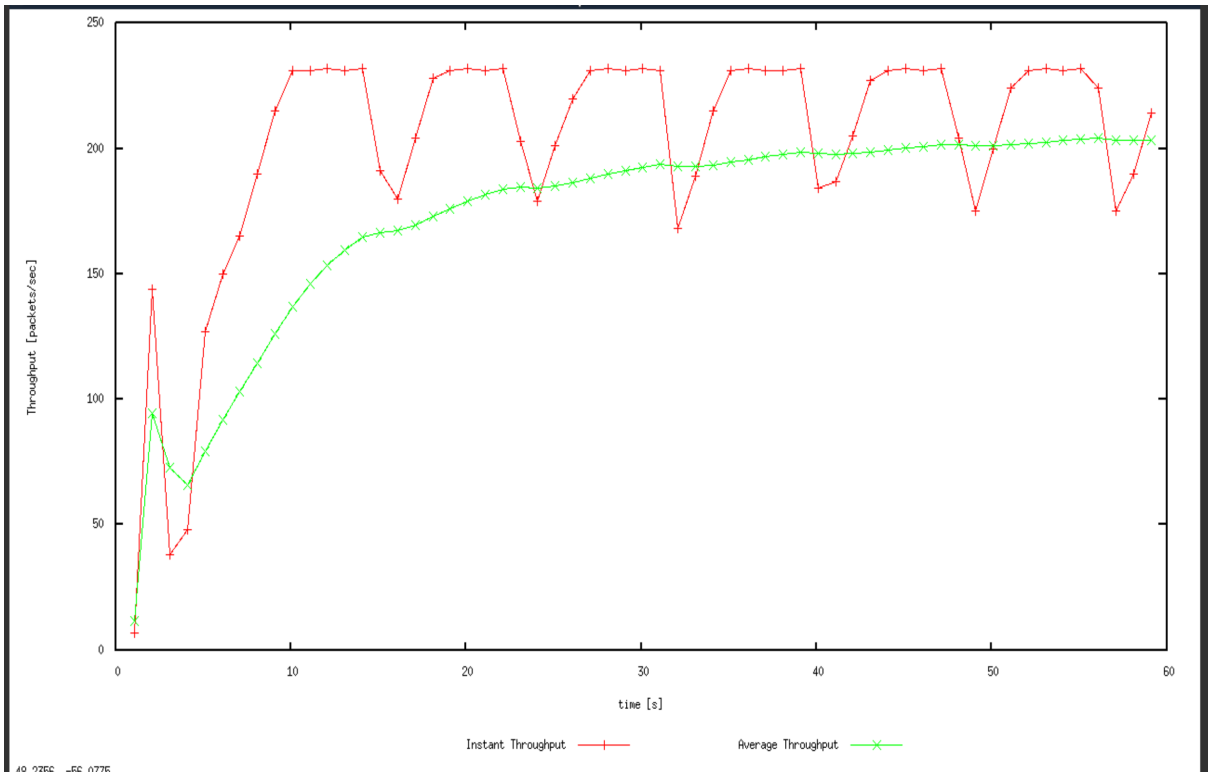
      (b) TCP Reno resets CWND to half the maximum value when CWND reaches 67.

      (c) This is because the loss event was a triple duplicate ACK, and not a timout. TCP Reno resets CWND to half the maximum value on a triple duplicate ACK, unlike TCP Tahoe, which resets CWND to 0 on any loss event.

      (d) TCP Reno does not transition back to slow start after a triple duplicate ACK loss event. This is becaues SSTHRESH is reset to half of the CWND size, which is also what CWND is reset to. Thus, in the next timestep, the CWND is not less than the SSTHRESH, they are actually the same. Therefore TCP Reno simply returns to AIMD.

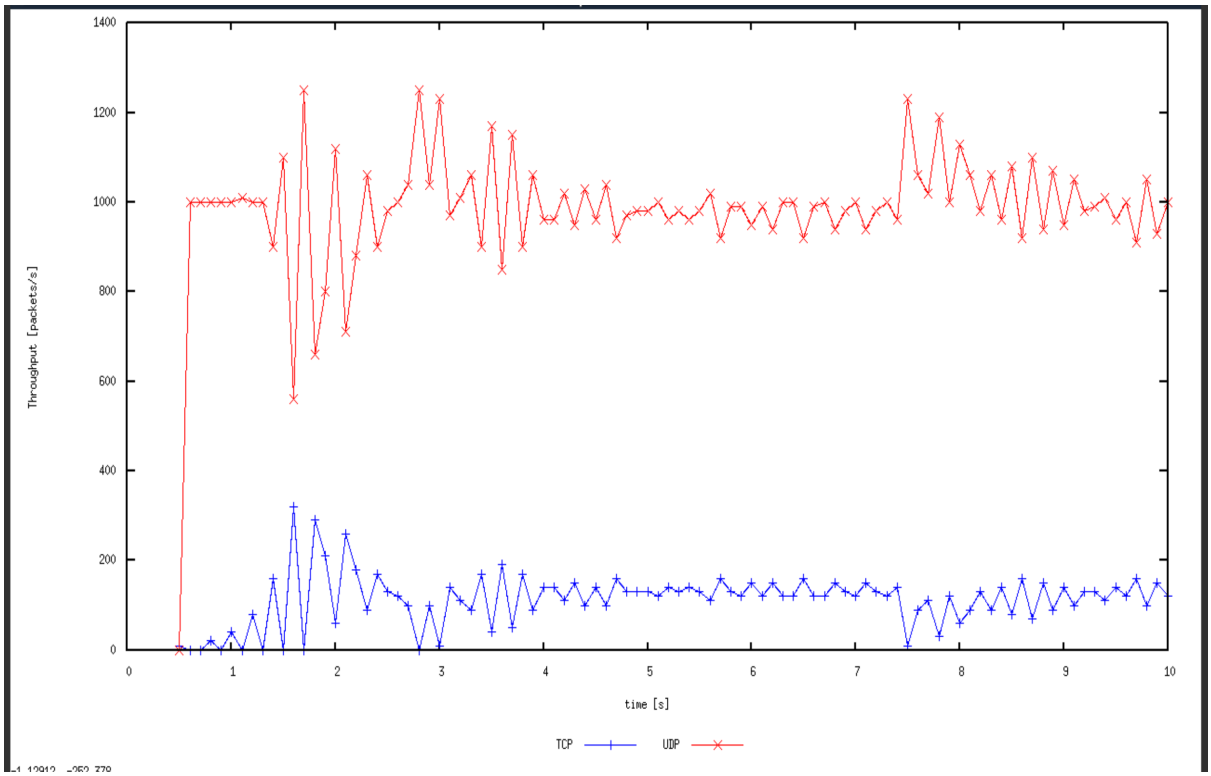


2. Average throughput was 203.413 packets per second, which is 878744.16 bits per second. Clearly, TCP Reno has a higher average throughput.



EXERCISE 3

1. Initially all of the traffic will be from UDP, 4 Mbps, as the TCP connection is undergoing slow start. However, soon TCP will increase the sending window size, exponentially through the slow start probing. At some point, the TCP traffic will exceed 1 Mbps, and overflow the link, causing packet loss in both the UDP and TCP connections. As a result, TCP will reset the value of CWND to half of the value that the loss occurred at, as well as resetting SSTHRESH. TCP then transitions into AIMD immediately, as CWND is no longer less than SSTHRESH. TCP will now linearly increase its traffic, until overflow occurs again, and CWND is reset. The cycle will continue to repeat. The UDP traffic will sit around 4 Mbps, with inherent fluctuations, and TCP will oscillate up towards 1 Mbps, as CWND resets. UDP is clearly the red stream, and TCP the blue stream.



2. UDP is transmitting at 4 Mbps upon launch of the program, as there is no regulation for UDP traffic. TCP intially has no throughput, as it is in the slow start phase, and is still beginning its growth. Eventually, the slow start increase will cause the first packet loss, and as such, CWND and SSTHRESH will both be reset. With TCP Reno, this puts an end to the slow start increase, as the loss events are all triple duplicate ACKs. Thus, TCP traffic incrementally increases under AIMD, so the TCP traffic will oscillate towards 1 Mbps, and begin to stabilise.

3. UDP will always have the same throughput if the link has the capacity to support the full capacity of the UDP traffic. UDP will always send its full capacity through the link. In this sense, UDP is advantageous over TCP, as the transfer will be far faster. However, if everyone switched to UDP, with no congestion control mechanisms, the full capacity traffic of each UDP user will constantly overflow the link, causing packet loss. Packet loss will corrupt the file in file transfers, and would then require a retransmission, further burdening the link and causing more loss.