Peter Vieira
2014/01/30

ECE 6110 – CAD For Communication Networks
Lab 1 – TCP Throughput Measurements

I. Introduction

A simple TCP connection was set up with a client (source node), two routers and a server (sink node). Figure 1 shows the network topology. The client sends infinite amount of data to the server over three links. The two outside links have 10 ms propagation delay and a bandwidth of 5 Mbps. The link connecting the two routers is a bottleneck with a 20 ms propagation delay and a 1 Mbps bandwidth.

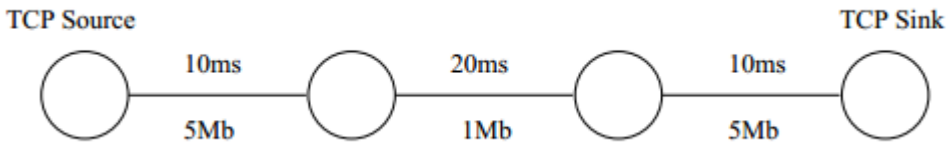

Fig. 1: TCP Topology

II. TCP Single-Flow Throughput

Using Network Simulator 3 this topology was constructed and simulated. During the simulation the throughput was measured as a function of the receiver window size, the bottleneck queue size and the segment size. Also, TCP-Tahoe and TCP-Reno protocols were compared for each set of parameters. Based on a general understanding of TCP communication the expected behavior of the network is as follows:

- As receiver window size increases, goodput should increase.
- As source segment size increases, goodput should increase given a large enough queue size.
- As queue size increases, goodput should increase only if the segment size is close to the queue size.
- The ideal throughput would be approximately the bandwidth of the bottleneck, which is 1 Mbps * 1 byte/8 bits = 125000 bytes/sec.
- Tcp-Reno should perform better due to its fast-retransmit after a triple duplicate ACK, during which the congestion window gets halved instead of dropped down to one packet size.

Figure 2 shows how the goodput varied as a function of window size, segment size and queue size, with each different segment size being represented by a different color. It is evident from the figure that network goodput using TCP-Tahoe protocol generally increased as queue size increased, except for when the window size was 2000 bytes, in which the rate was constant across all four queue sizes; and when the window size was 64000 bytes, in which the goodput increased up until a queue size of 64000 bytes with a segment size of 128 bytes. Based on the NetAnim flow animation, this seems to be caused by an acknowledgement timeout because the packets are so small, causing lots of packets to go on the transmission line at once, which creates a longer round-trip time.

With smaller receiver window sizes, goodput was inversely proportional to segment size, but with

larger window sizes, as segment size increased so did goodput. This is most likely due to the fact that smaller segment sizes enable the sender to get closer to the queue and window size limits. But when the window size is larger enough, smaller segment sizes result in more packets sent and thus create longer round-trip times than larger segment sizes would, and thus are more prone to timeouts.

The results from Tcp-Tahoe compared to Tcp-Reno were similar, and not as different as expected. On the whole Tcp-Reno performed better, but in some cases it performed worse. Specifically, Tcp-Reno performed worse in two setups: 1) segment_size = 128 bytes, window_size = 8000 bytes and queue_size = 2000 bytes, and 2) segment_size = 128 bytes, window_size = 64000 bytes and queue_size = 64000 bytes. This could possibly be due to a packet loss during the Tcp-Reno simulation and not during the Tcp-Tahoe simulation. Timeouts also could have occurred during the Tcp-Reno simulation, causing the congestion window to go back to one maximum segment size (MSS). Overall, a single TCP flow resulted in some unexpected but understandable outcomes.

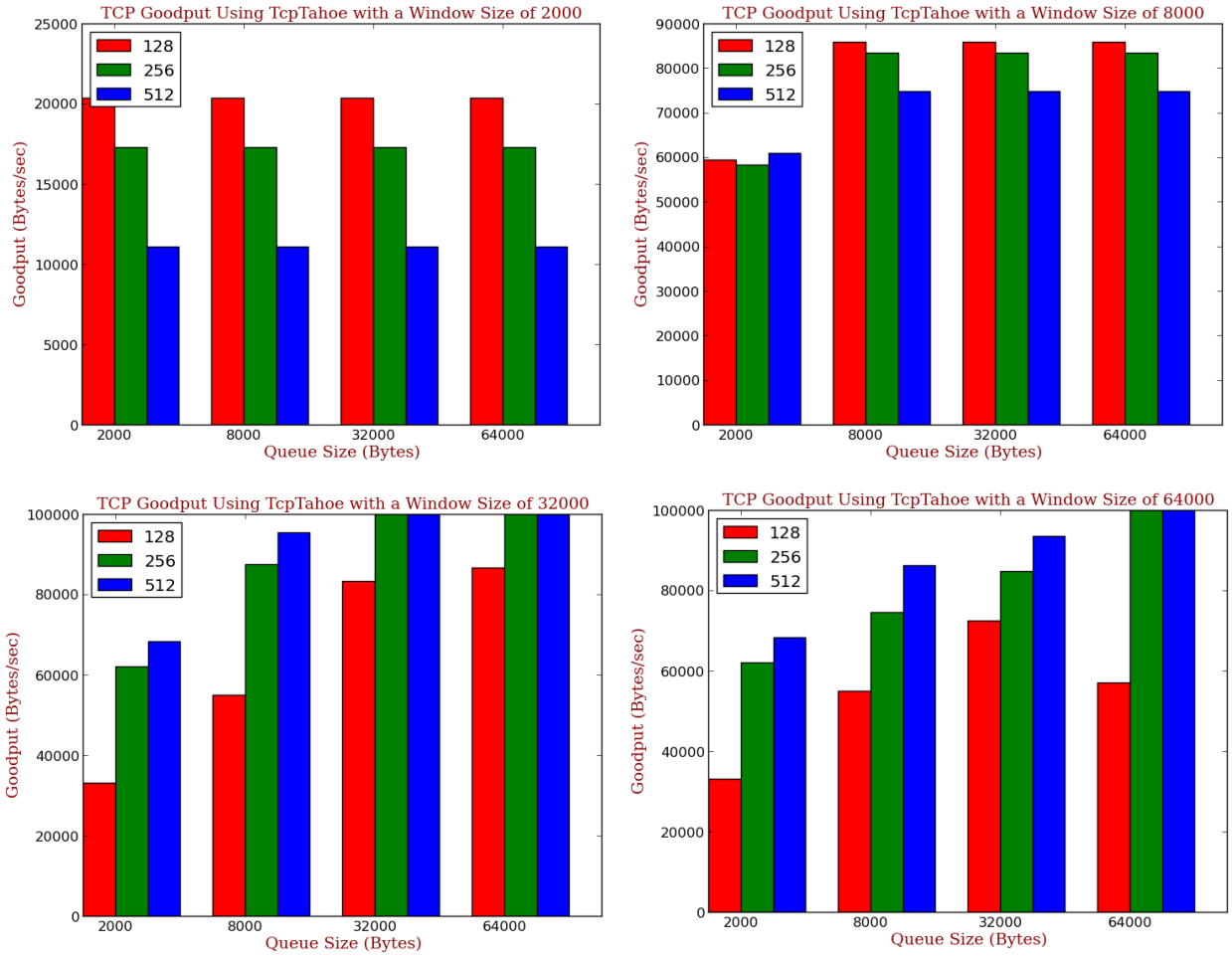

Fig. 2: TCP-Tahoe Goodputs with Increasing Receiver Window Sizes

Each colored bar represents a different packet segment size in bytes.

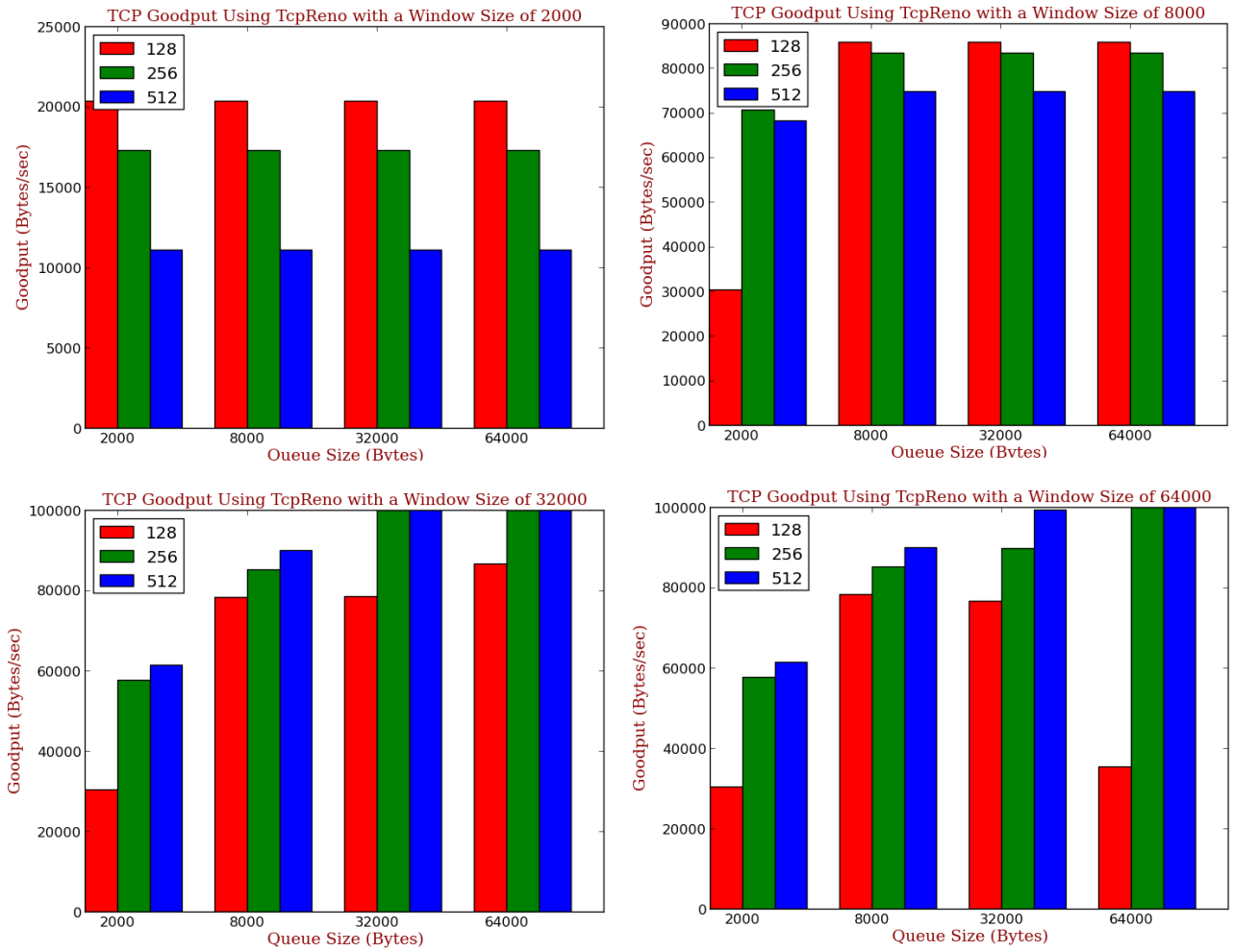

Fig. 3: TCP-Reno Goodputs with Increasing Receiver Window Sizes

Each colored bar represents a different packet segment size in bytes.

III. TCP Multi-Flow Throughput

The second setup consisted of ten simultaneous flows all sharing the same three links. To measure the variation between the ten flows, ten bulk send applications each transmitted data to a different port on the receiver node. In other words, all ten flows are sharing the bandwidth of the links. Each bulk send application was started at a random time between 0 – 0.1 seconds, with times generated using a pseudo-random number generator with a uniform distribution.

One would expect the flows to share the bandwidth relatively evenly, and for Tcp-Reno to perform better than Tcp-Reno due the fact that Tcp-Reno uses a fast-retransmit algorithm, which prevents the congestion window from dropping back to one packet size on a triple duplicate ACK.

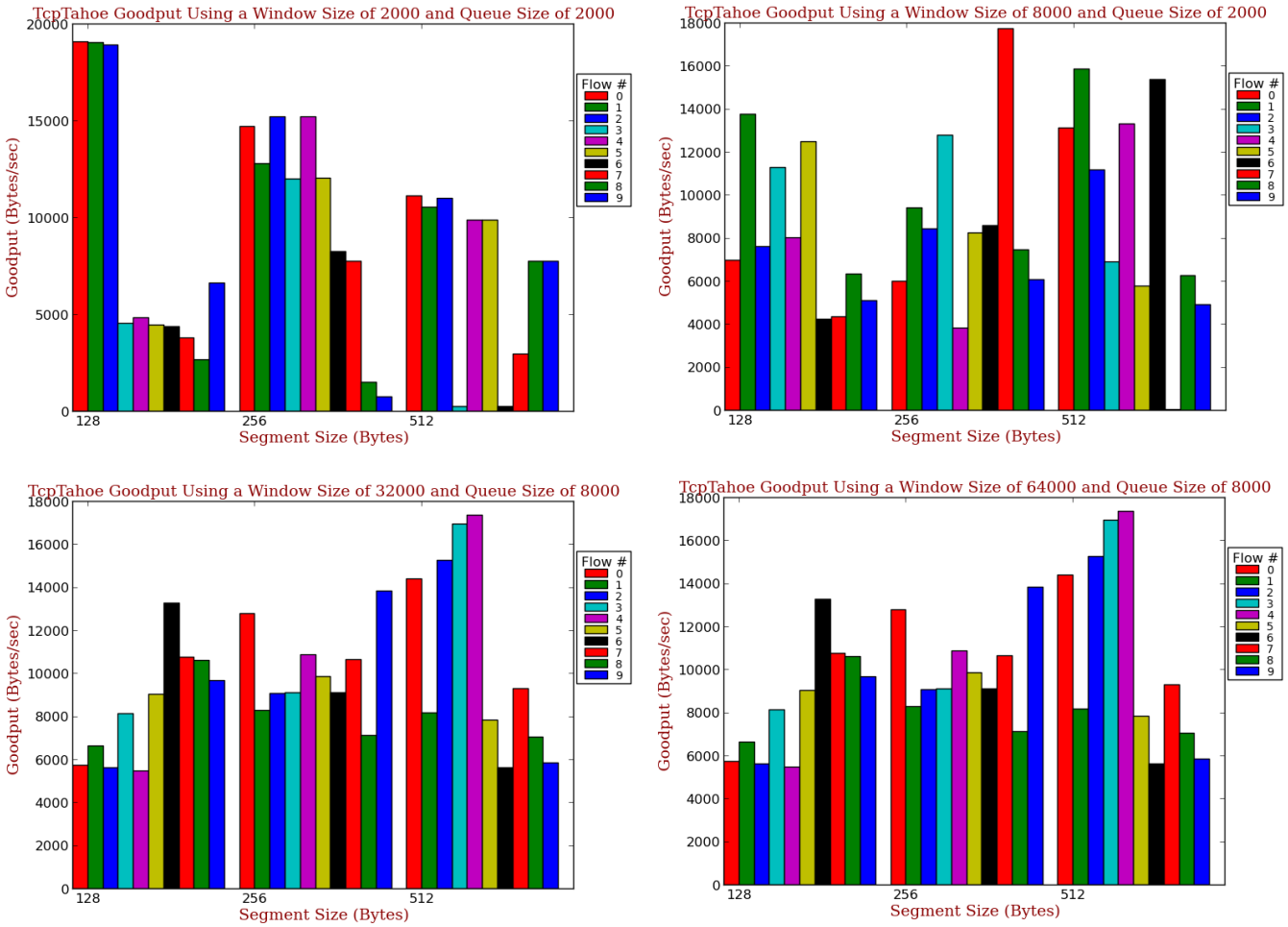

Fig. 4.Tcp-Tahoe Goodputs of 10 Simultaneous Flows with Varying Window and Queue Sizes

Each colored bar represents a different flow.

Figure 4 and 5 show how the goodput of the 10 simultaneous flows compared for different segment sizes, window sizes and queue sizes, for Tcp-Tahoe and Tcp-Reno, respectively. Only four representative simulation setup results are shown for brevity and clarity. Each colored bar represents a different transmission flow.

Tcp-Tahoe and Tcp-Reno performed similarly, especially due to the fact that the same seed for the random number generator was used for both simulations. The ten different transmission flows did not share the bandwidth evenly at all. Instead their goodputs varied tremendously. This is likely due to the event in which if one or more flows have a timeout or a triple duplicate ACK and slow their transmission rates significantly due to a drop in the congestion window and slow start threshold, it leaves more bandwidth for the other flows. Therefore, which flow results in the largest goodput has to due with luck, and is based primarily on the pseudo-random number generator.

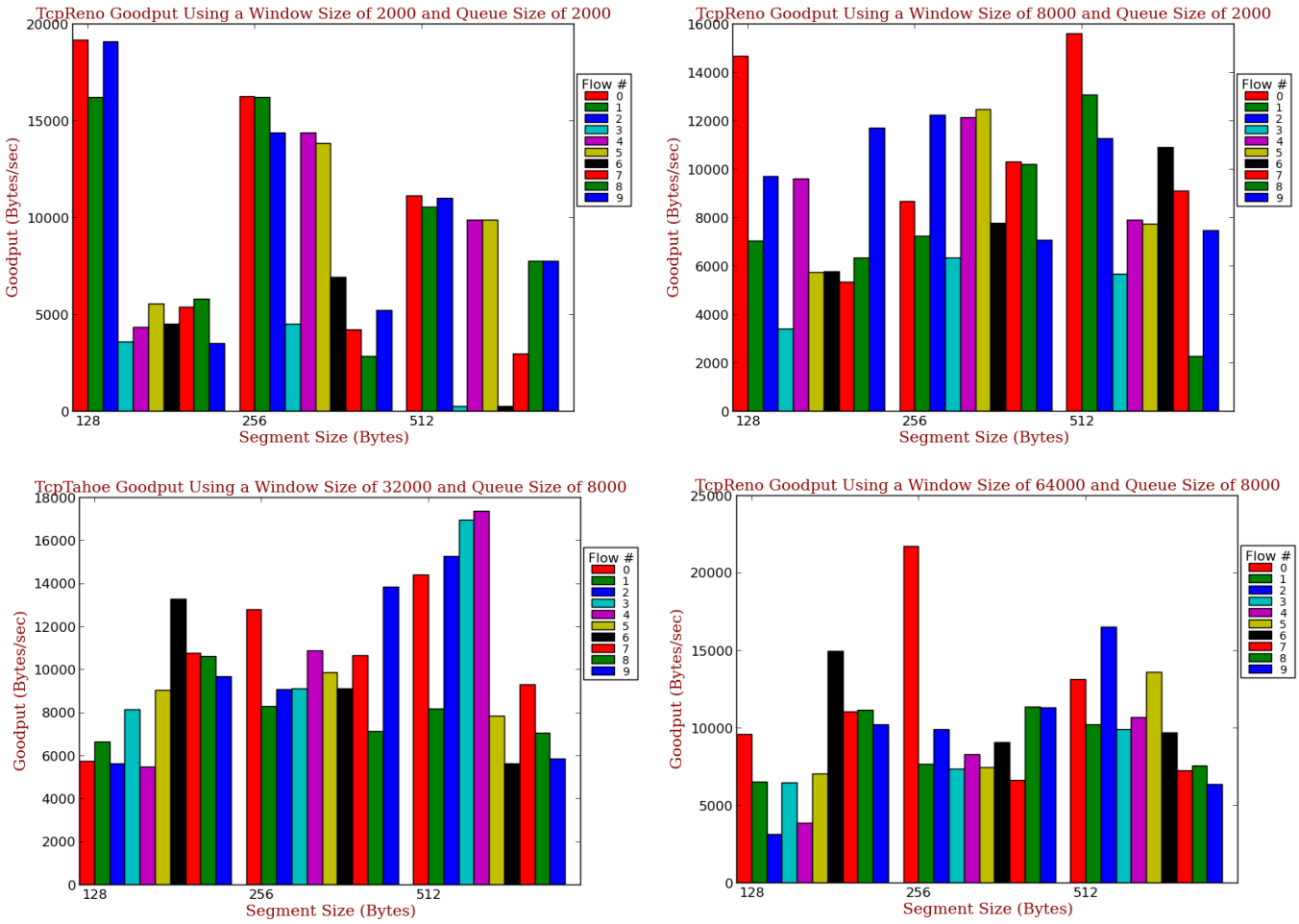

Fig. 5. Tcp-Tahoe Goodputs of 10 Simultaneous Flows with Varying Window and Queue Sizes

Each colored bar represents a different flow.

IV. Conclusion

Overall, the performance of the simulations were a bit surprising in some areas and as expected in others. Segment size interacted with queue size to perform better with smaller segment sizes due to the ability to get closer to the queue size before overflowing or splitting up packets. Tcp-Tahoe and Tcp-Reno performed similarly due in part to fact that timeouts seemed to be more prevalent than triple duplicate ACKs. In future simulations more information about packets losses and duplicate acknowledgements should be recorded for better insight.

When performing the ten simultaneous flows, a better approach would be to use different seeds for the pseudo-random number generator between runs and between the Tcp-Tahoe and Tcp-Reno setups. Also, testing different distributions for the pseudo-random number generator may provide more insight, and a more realistic average result.