Competing Tahoe Senders

Adam Stambler
Brian Do
Group 7
12/7/2010

Computer and Communication Networks : 14:332:423

Break Down of Contributions

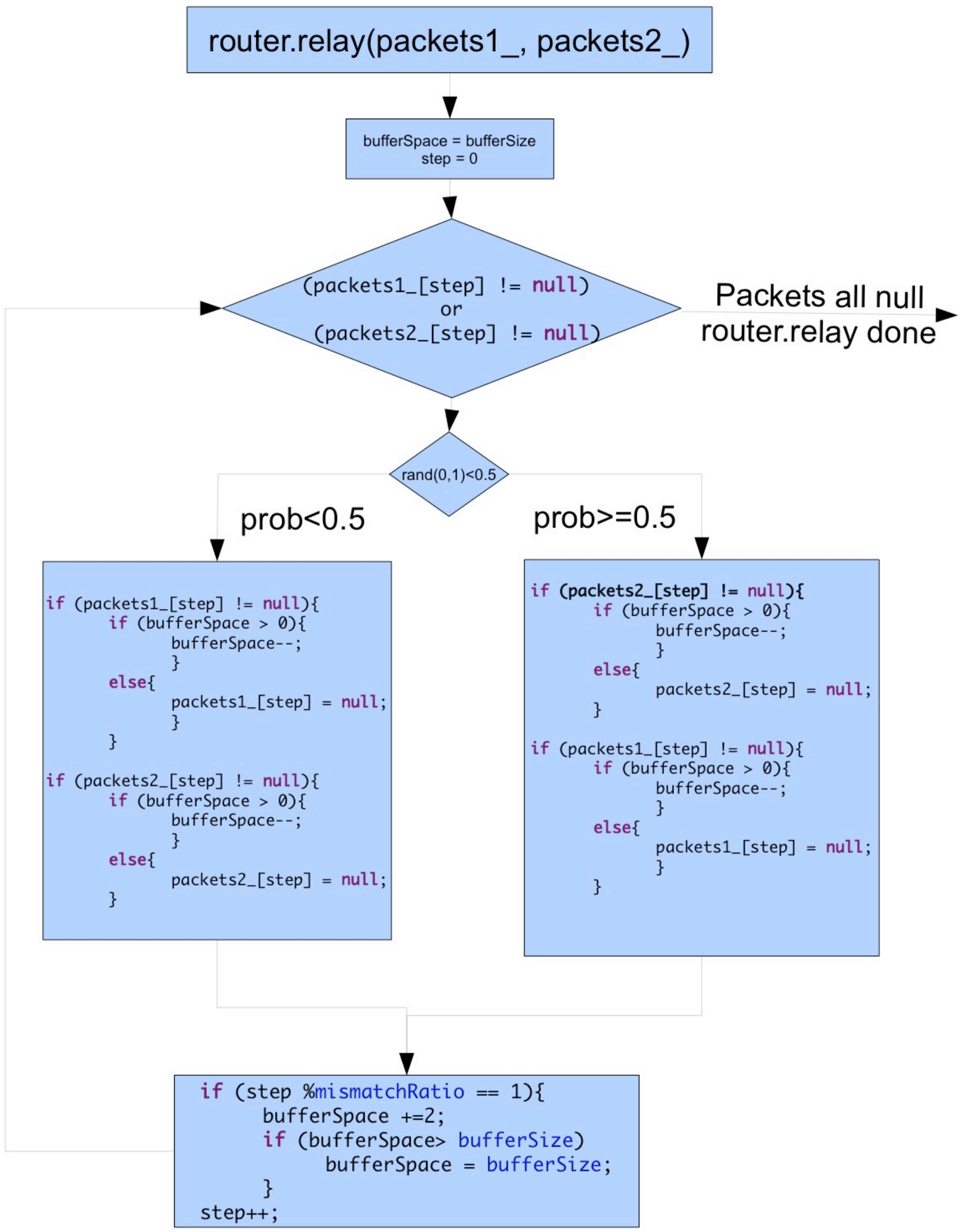
	Adam Stambler	Brian Do
Algorithm Design	50	50
Coding	70	30
Debugging	40	60
Analysis of Results	40	60
report preparation	40	60

Introduction:

For our project, we explored competition for scarce router resources in a TCP tahoe network. In our network, we had two TCP Tahoe senders transmitting to two different TCP receivers. Both of these connections had to go through a single bottleneck router which had a limited buffer size and a slower transmission speed. This bottleneck causes losses for the TCP senders. In our project we watched how competition effected the the transmissions. Additionally, we studied the effects of TCP senders trying to keep alive their connection by sending junk data versus simply stopping communication when they are done.

Algorithm Design:

Our main algorithmic challenge was altering the TCP simulator router to relay both TCP sender messages. The flow diagram is shown in Figure 1. The router is responsible for buffering and transmitting as many packets as possible as it receives them. However, since the buffer space is a bottleneck, it must discard some packets. We implemented this bottleneck by iterating through the sender transmission periods. During each period, the router randomly selects which sender's packet started first and therefore which sender is given priority in that transmission period. For each sender, it checks to see if the buffer has space. If it can support a packet, the packet is added. Otherwise, the packet is dropped. Additionally, to model the routers transmission of packets in the router's buffer, we free space in the buffer as the router has enough time to transmit the packets.



```
if (packets1_[step] != null){
        if (bufferSpace > 0){
                bufferSpace--;
                }
        else{
                packets1_[step] = null;
                }
        }

if (packets2_[step] != null){
        if (bufferSpace > 0){
                bufferSpace--;
                }
        else{
                packets2_[step] = null;
        }
```

```
if (packets2_[step] != null){
        if (bufferSpace > 0){
                bufferSpace--;
                }
        else{
                packets2_[step] = null;
        }

if (packets1_[step] != null){
        if (bufferSpace > 0){
                bufferSpace--;
                }
        else{
                packets1_[step] = null;
                }
        }
```

```
if (step %mismatchRatio == 1){
        bufferSpace +=2;
        if (bufferSpace> bufferSize)
                bufferSpace = bufferSize;
        }
step++;
```

Figure 1: TCP Router Relay Flow Chart

Implementation:

For the project, we used the Java simulator provided as the basis for our simulator and used a python program to produce our plots. Our simulator implementation consists of a modified router, sender, and simulator source file. Each new file and class is designated with an additional P5 at the end of its name: the router is routerP5.java, the sender is TCPSenderTahoeP5.java, and the simulator is TCPSimulatorP5.java. The router has been modified to have a new two sender relay function (described in figure 1). The sender has been modified so that saves a log file which we use later for graphing. Finally, the TCP simulator has been modified to stitch our competing senders together. It now contains 2 TCPTahoeSendersP5 and a RouterP5. It accepts command line arguments for number of simulator iterations, the mismatch ratio between the router and receiver, and the buffer size of the router. To run our sender simulator, simply run TCPSimulatorP5 and give the number of iterations as its command line argument.

To simulate the cases involving the dropped connection and the connection maintenance, the code for the normal contention scenario was modified. The dropped connection was implemented simply by preventing all segment transmission and processing for the second sender between the specified period. Also during this period, connection values of zero for this sender were written to the log. When the connection resumed, the the SSThresh was reset to its initial value and the sender was placed into slow start.

To have the sender maintain its connection while it elected not to send segments, the connection values and sender state were saved. This was done by having the sender transmit 1-byte segments to maintain these values; the values to be saved were written to log each time a 1-byte segment was transmitted. Also as these 1-byte segments were transmitted, all requests for new packets and retransmissions were ignored by the sender. When the sender resumed sending, the saved values allowed it to resume transmission as though it had never stopped.

Our plotting utility, called tcpgraphy.py, plots the log files generated by our TCP senders. It must be run in the same directory and the Sender1 and Sender2 log files. The program uses the opensource library matplotlib to generate the plots.

Results and Discussion:

Case 1: Competing TCP Tahoe Senders

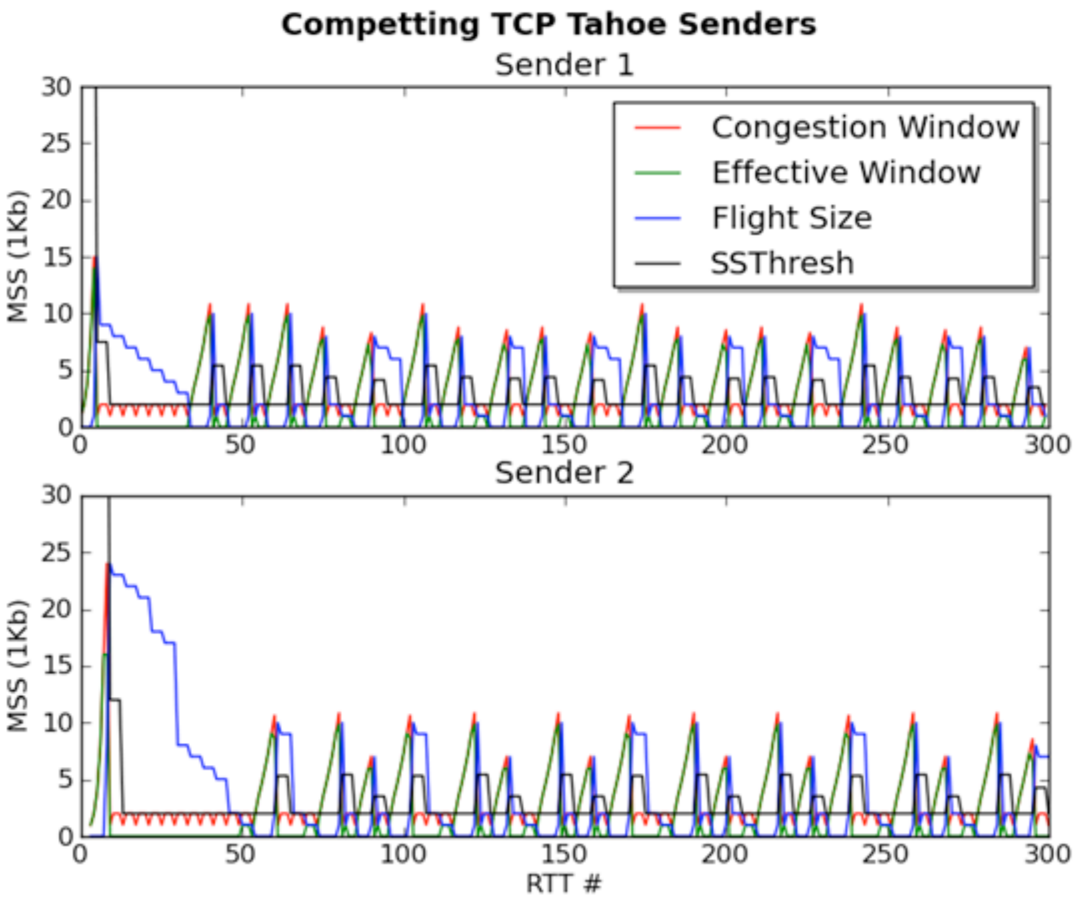


Figure 2: Plots for opposing TCP Tahoe senders over a shared router for 300 RTTs where the second sender's tranmission is delayed 3 RTTs. The mismatch ratio is 10:1 and the buffer cache can store 6+1MSS. The sender utilizations were measured as follows: Sender1 utilization: 27 %, Sender2 utilization: 26 %

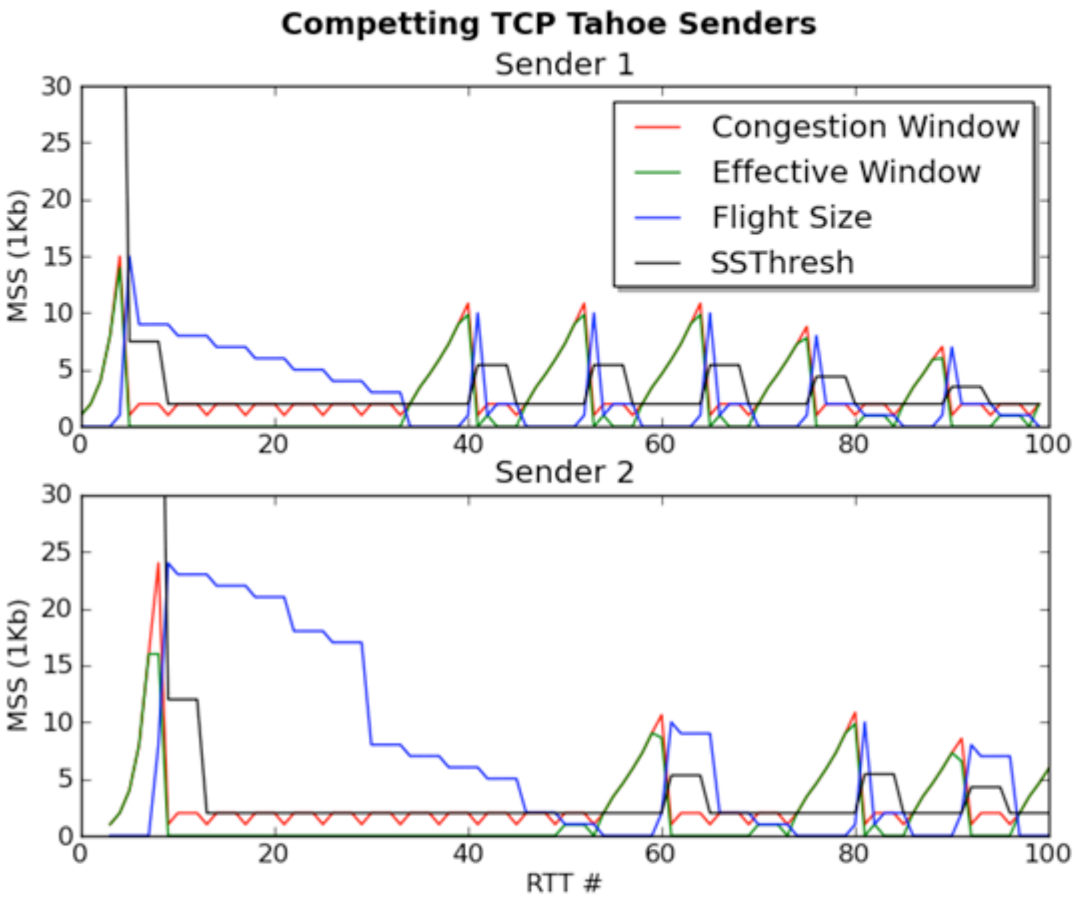


Figure 3: A Magnified View of Figure 2.

Above are the plots for the two opposing senders for the initial 300 RTTs. There is also a set of plots with only the first 100 RRTs for a more detailed view. In the beginning, sender 1 then sender 2 are initialized separately (sender 2 3RTT later) with a large SSThresh (65535 bytes) and begin to send in slow start until three duplicate acknowledgements are detected. Slow start is an exponential process and this is observed in the rapidly increasing slopes of the effective windows for both senders during their early stages.

The presence of two competing senders accelerates the rate at which the router's buffer fills, which causes segments to be dropped. The first segment is dropped from sender 1's transmission in the third RTT but the third duplicate acknowledgement is not received until the fourth RTT. This subsequently causes sender 1 to retransmit in the fifth RTT, two RTTs after the dropped segment.

Between 5 and 33 RTT, sender 1 oscillates between two states, retransmission and sending 1-byte segments. The 1-byte segments are used to keep the connection alive while the sender waits for the congestion to lessen. Both the frequency and necessity to retransmit and to send 1-byte segments are lengthened by the presence of two competing senders as opposed to one. This is inefficient because being in either one of these states entails that no new packets get transmitted. However, while one sender is in one of these states, the other sender has less competition for buffer space.

At 34 RTT sender 1 enters congestion avoidance, where it remains until three duplicate acknowledgements are received at 39 RTT. Sender 2 does not enter congestion avoidance until 52 RTT. From this point on, the two senders alternate between retransmission and congestion avoidance. Take notice that the spikes, excluding that in the initial slow start phase, for sender 1's congestion window, effective window, and flightsize do not grow greater than the spikes between 38 and 60 RTT. During this period, sender 2 was in an extensive oscillation between retransmission and 1-byte transmissions and only utilizes a fraction of the router buffer so there is little competition for buffer space.

Below is a situation in which only one sender transmits over a router of the same buffer size. The spikes are uniform and their peaks are determined only by the size of the buffer and the mismatch ratio of the connections to and from the router.

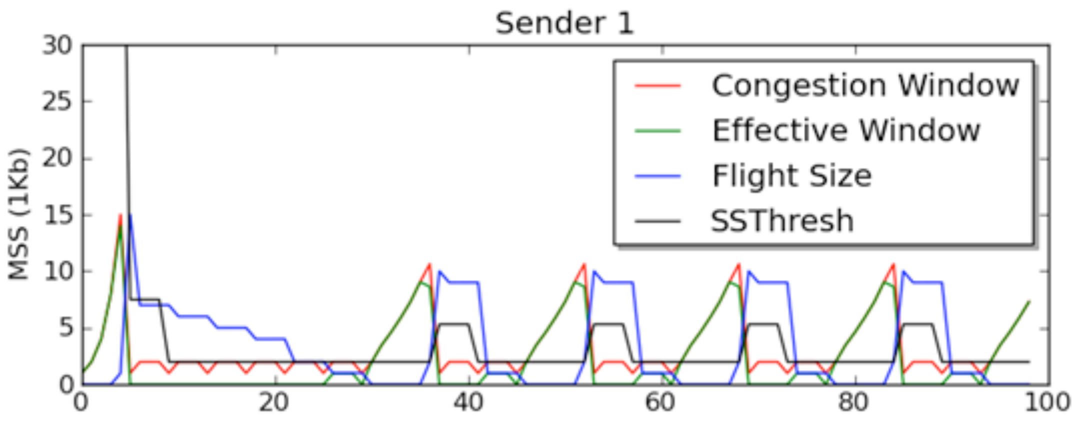


Figure 4: Plot for Sender without Competition

Comparing this plot against the plots of the opposing senders, it is apparent that peaks of the spikes in the competing graphs are not uniform whereas the spikes in figure 4 are. If chance may allow, a competing sender will only achieve a spike of magnitude similar to the single sender when its opposing sender is transmitting a 1-byte segment. Otherwise, buffer overflow occurs prematurely and senders are forced to exit congestion avoidance before the potential congestion window, effective window, etc. are reached.

Case 2: Competing TCP Tahoe Senders (Sender 2 drops connection between 100-150 RTT)

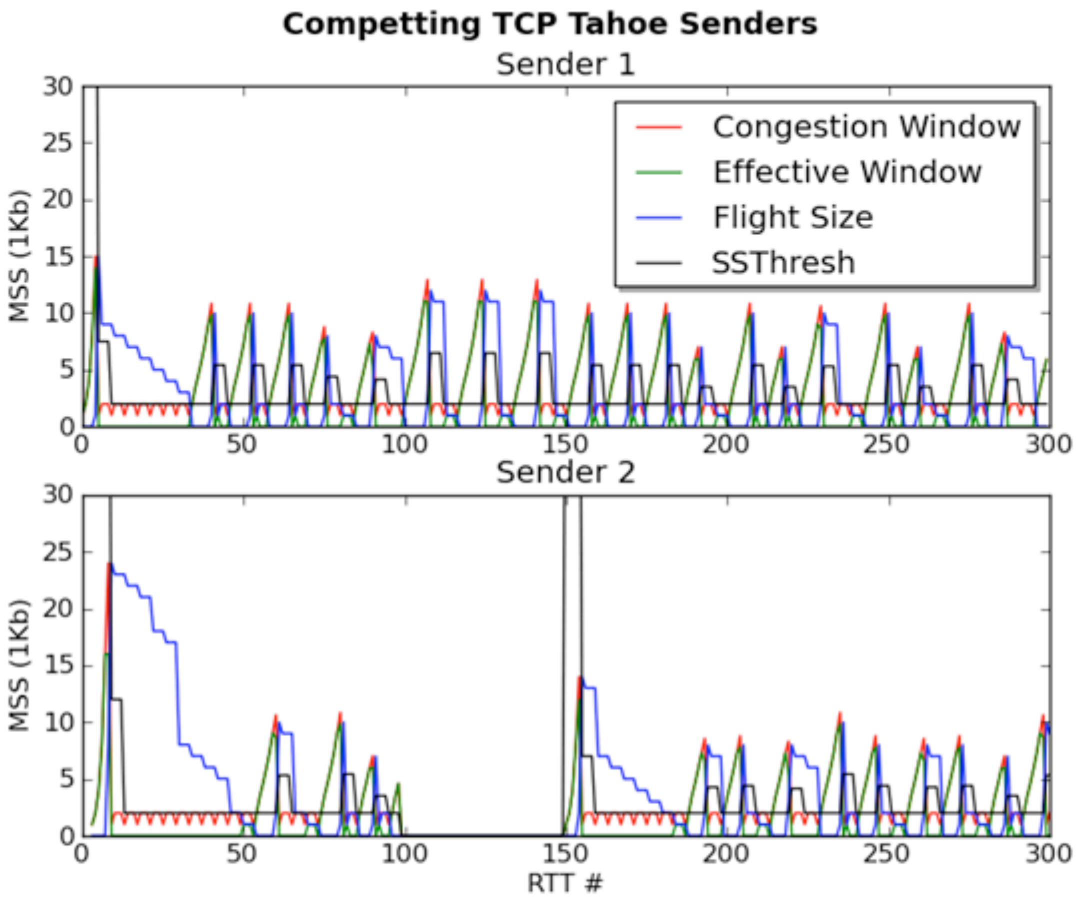
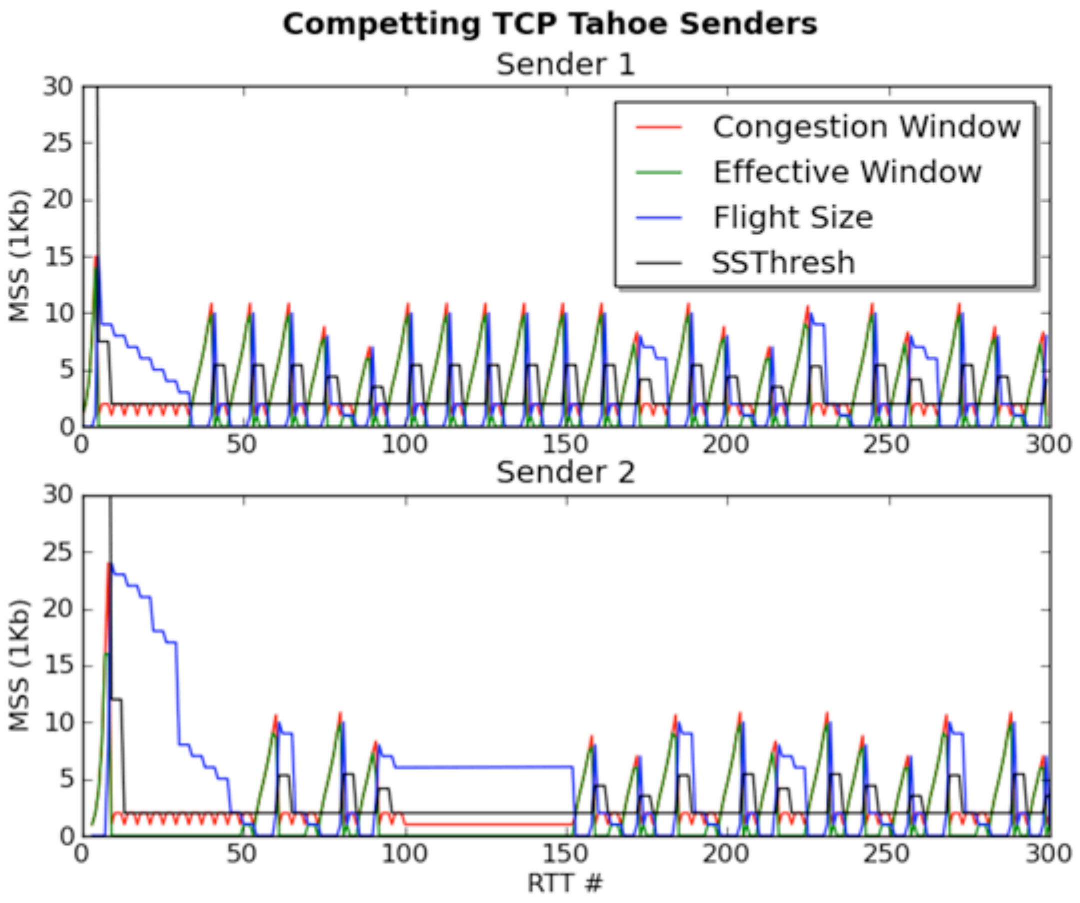


Figure 5: Plots for competing Tahoe senders where one sender drops and reestablishes connection. The sender utilizations were measured as follows: Sender1 utilization: 30 %, Sender2 utilization: 13 %

In the plots above, sender 2 drops its connection at 100 RTT and then reestablishes the connection at 150 RTT. During theisperiod where sender 2 ceases to send, the peak congestion window, effective window, flightsize, and SSThresh of sender 1 become uniform for all of its congestion avoidance cycles. The maximum values for these measurements are also greater than the maximum values of the spikes that exist outside of the span in which sender 2 ceases to transmit. This is expected because sender 1 has no competition for the router's cache, therefore its connection values are only dependent on the the size of the router buffer.

When sender 2 reestablishes its connection at 150 RTT, it must enter slow start with a large reinitialized SSThresh because its connection values were lost when the session was dropped 50 RTTs earlier. In slow start, sender 2 quickly transmits too many packets. Consequently, it must reduce its SSThresh and spend the next 20+ RTTs retransmitting and sending 1-byte segments. Anytime a sender has to go through slow start, its flightsize becomes excessively high and the sender and must spend the next span of RTTs retransmitting and sending 1-byte segments while it waits for the flightsize to fall; this is inefficient. Concurrently, the effects of the new competition with sender 1 are seen in the subdued spikes in sender 1's graph.

Case 3: Competing TCP Tahoe Senders (Sender 2 sends 1-byte sized segments to maintain its connection)



Plot 6: Plots for competing Tahoe senders where one sender stops sending meaningful data but maintains to connection. Sender1 utilization: 30 %, Sender2 utilization: 22 %

Similar to the previous example, sender 2 stops sending meaningful packets between 100 and 150 RTT. The difference is that in this case, sender 2 retains its session with its receiver by sending 1-byte segments. Keeping the connection alive even though it does not have meaningful data to transmit allows sender 2 to skip slow start on resumption because the values for congestion window, flightsize, etc. are kept alive. Therefore, on resumption, sender 2 can start congestion avoidance as if a lull in packet transmission never occurred.

Case 4: Competing TCP Tahoe Senders (Sender 2 has 150 RTT delay)

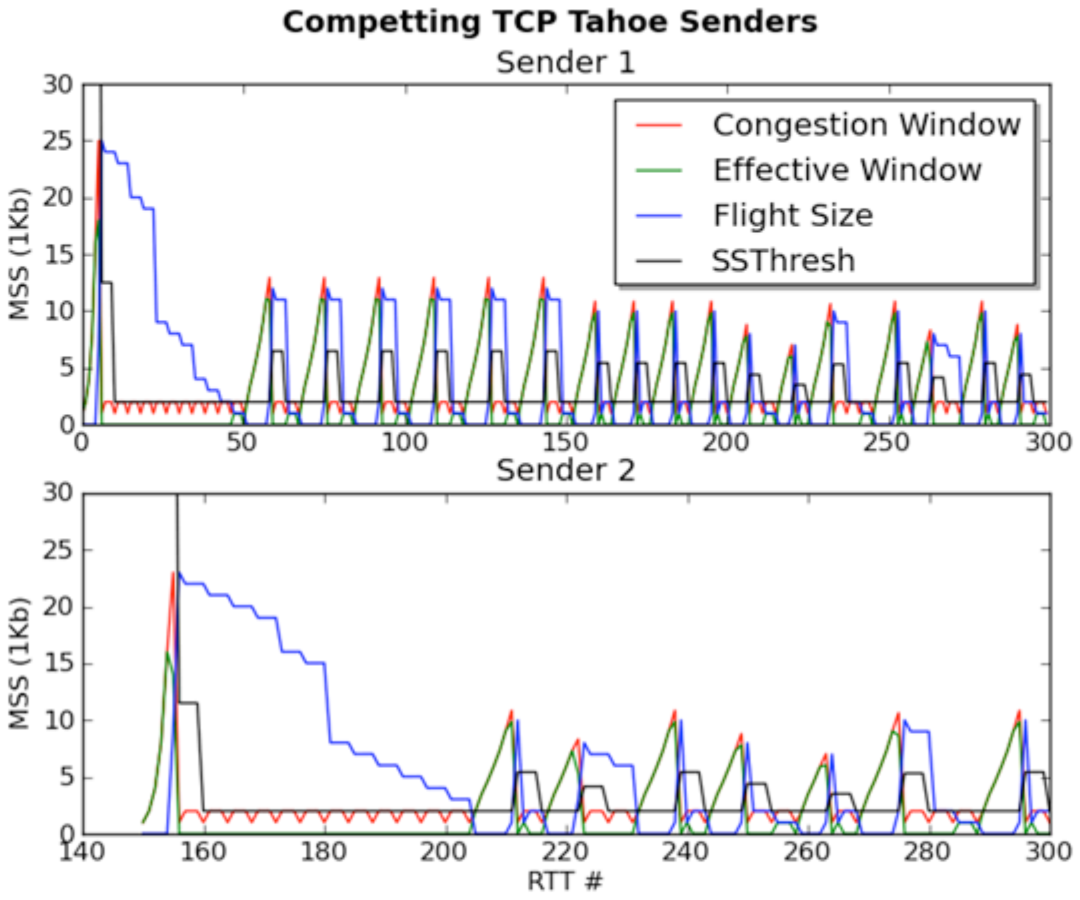


Figure 7: Plots for competing Tahoe senders where one sender's transmission is delayed 150 RTTs. Sender1 utilization: 31 %, Sender2 utilization: 21 %

Because sender 1 has no competition until 150 RTT, the congestion avoidance spikes until 150 RTT are very high and uniform. Once sender 2 begins to transmit, the effects can be seen immediately in sender 1's next cycle. Notice that sender 1's spike peaks after 150 RTT have lower magnitudes than those before sender 2 began transmission. This is due to the new competition for buffer space.

The four congestion avoidance cycles after 150 RTT for sender 1 have similar slope and maximum values. This is because sender 2 transmits a near constant amount of segments during this span of cycles. Once sender 2's flightsize falls from its large slow start value, sender 2 enters congestion avoidance. From this point on, both senders cyclically transition in and out of congestion avoidance and retransmission, and their spikes are no longer evenly spaced or uniform.

Case 5: Buffer sized increased from 7 segments to 9 segments

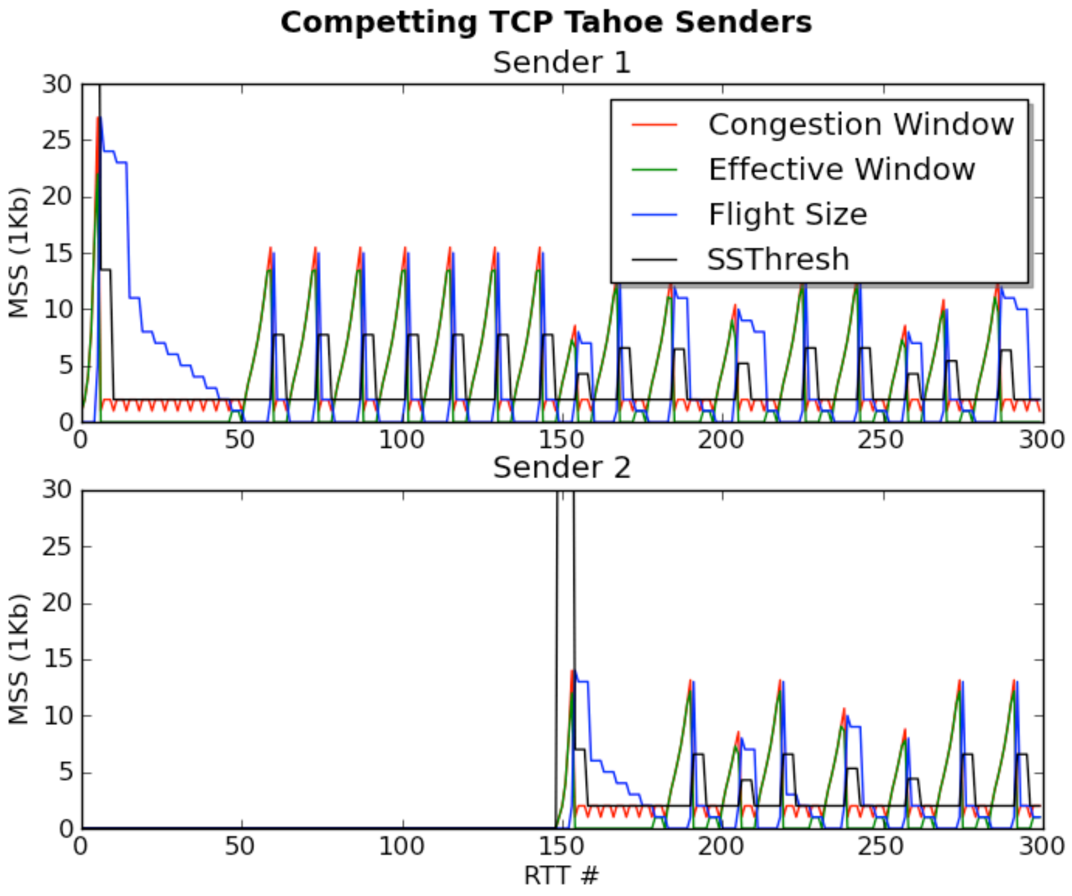


Figure 8: Plots for Contending Senders with 9MSS router cache. Sender1 utilization: 33 %, Sender2 utilization: 22 %

The increased router cache raises the maximum connection values for the congestion avoidance cycles.

Case 6: Router Buffer Size's Effect on Sender Utilization

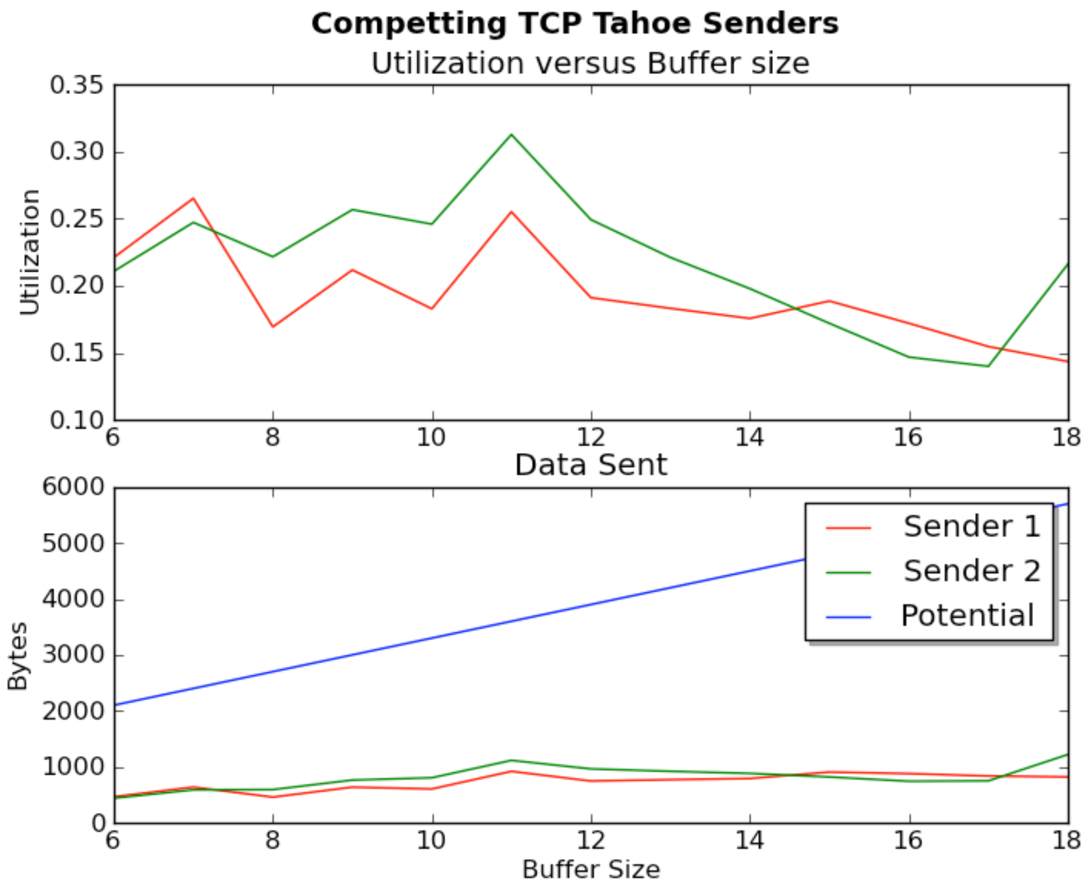


Figure 9: Buffer Size versus Sender Utilization

As a final test, we examined how the router's bottleneck, the buffer size, effected the sender utilization. The router to sender mismatch ratio was fixed twenty and the router's buffer size was varied from six to eighteen. This test yielded inconclusive and confusing data. While the increased buffer size should have produced a marked increase in sender utilization, the amount sent by each sender did not increase by much. As can be seen in Figure 9, the bytes sent stayed relatively flat relative to the potential transmission size. This causes the sender utilization to decrease as the buffer size increases. This same effect was seen in the provided, single TCPTahoe sender simulation. We believe this is a deeper error in the simulator but did not get a chance to investigate this effect further.

<u>Conclusion</u>

In conclusion, competing TCP senders cause both senders to transmit less data. In our tests, we have shown a drop in utilization from 30% for a single TCP sender to 26% over a thousand iterations. However, the competition increases the overall transmission rate of the router. The collective senders have a larger minimum transmission rate. Instead of the multiplicative decrease reducing the total transmission to 1 packet, it reduces it to 2 packets.

Then, as the fast retransmit occurs, the senders increase their total flow faster than a single sender could. This puts a higher lower bound on total data transmitted and increases the frequency of the multiplicative decrease/ slow start frequency.   Additionally, we investigated techniques used by TCP senders to improve throughput in a competitive, bottle-necked network. We examined the difference between a sender pausing between transmissions versus sending junk data to keep the connection alive. Sending junk data significantly improves the second senders throughput. If the second sender pauses it only has a 13% utilization while the junk transmission allows a 22% utilization.