CS460 Lab 3 Report: Congestion Control

Michael K. Eagar

March 24rd, 2014

1 Congestion Control Implementation - TCP Tahoe

I used my Reliable Transport Protocol implementation as the basis for implementing Congestion Control using the TCP Tahoe model. My implementation is saved in the file *my_RTP.py*. I needed to change my *transfer.py* file as well, in order to use the Simulator scheduler to schedule different events to start at different times. I modified the *transfer.py* file to add the window initialization method as a scheduled event. I also implemented new versions of *transfer.py* to handle more than one file transfer (two and five flows each) at a time. These are saved as *transfer2.py* and *transfer5.py*. To begin, I implemented *Slow Start*, the *Slow Start Threshold*, *Additive Increase*, and loss event handling:

Slow Start - At the start of the connection, or after any kind of loss event, set *cwnd* (congestion window) to 1 MSS. Every time the sender receives an ACK for new data, increment *cwnd* by the number of new bytes of data acknowledged.

Threshold - Stop *Slow Start* when *cwnd* exceeds or equals the threshold. Start with a threshold of 100,000 bytes (100 packets).

Additive Increase - Once *cwnd* is larger than the threshold, use *additive increase*. Every time the sender receives an ACK for new data, increment *cwnd* by MSS * b / *cwnd*, where MSS is the maximum segment size (1000 bytes) and b is the number of new bytes acknowledged.

Loss Event Handling - When a loss event is detected (a timeout), set the threshold to the maximum of *cwnd* divided by 2, and MSS, and set cwnd to 1 MSS.

My implementation of these features mainly involved adding a few lines of code to my *slide_window()* and *retransmit* methods. My *slide_window()* method now reads as follows:

Listing 1: slide_window()

```
        def slide_window(self, ack_number):
                bytes_ack = ack_number - self.window_start
                packets_acked = int(math.ceil(bytes_ack / self.mss))
                self.packets_outstanding -= packets_acked
                self.window_start = ack_number

                if self.window_start > self.sequence:
                        self.sequence = self.window_start

                if self.packets_outstanding < 0:
                        self.packets_outstanding = 0

                if self.window_size < self.ssthresh:
                        self.window_size += 1
                else:
                        self.bytes_acked += bytes_ack
```

```
                    if self.bytes_acked >= (self.window_size * self.mss):
                        self.bytes_acked -= (self.window_size * self.mss
                            )
                        self.window_size += 1

            for i in range(self.window_size - self.packets_outstanding):
                self.send_if_possible()

            self.cancel_timer()

            if ack_number < len(self.send_buffer):
                self.timer = Sim.scheduler.add(delay=self.timeout, event
                    ='retransmit', handler=self.retransmit)
            else:
                self.timer_set = False

            self.window_file.write(str(Sim.scheduler.current_time()) + " " +
                str(self.window_size) + "\n")
```

The *retransmit()* method is now:

Listing 2: retransmit()

```
        def retransmit(self,event):
            self.timer_set = False
            if self.packets_outstanding <= 0:
                return

            self.ssthresh = int(max(math.ceil(self.window_size / 2.0), 1))
            self.window_size = 1
            self.bytes_acked = 0
            self.packets_outstanding = 1
            Sim.trace("%d retransmission timer fired" % (self.source_address
                ))
            packet = self.send_one_packet(self.window_start)

            if packet:
                self.sequence = self.window_start + packet.length

            self.window_file.write(str(Sim.scheduler.current_time()) + " " +
                str(self.window_size) + "\n")
```

After making these changes to my code and implementing the features for TCP Tahoe Congestion Control, I needed to add the ability to collect the data necessary to verify that my implementation of the new features was working correctly. In order to do this, I modified Dr. Zappala's *tcppacket.py* to allow the TCP packet to keep track of which flow it was a part of, implemented some new methods in *sim.py* to allow for exporting transfer rate and packet loss data, and additional functionality in *link.py* and *my_rtp.py* to export queue data and congestion window data.

The TCP packets now keep track of a new variable called *flow_id* that can be referenced to determine which flow they were part of. The rate, packet loss, queue, and congestion window data are all exported to individual files to make it easier to parse the data and plot the various graphs that are shown below. It was somewhat challenging for me to put the method calls in the correct places in order to get the proper data I needed to be able to show that my implmentation was working correctly, but I was able to get it to work after quite a bit of trial and error.

2 Experiments

To run the experiments that were required in the lab specification I used the *transfer1.py*, *transfer2.py*, and *transfer5.py* scripts that I had set up earlier. Each is set up to transfer the same number of files as is specified in the file names. I had to change them to allow the multiple file transfers and for them to have each transferred file saved with a different name in order to verify that they all transferred correctly. The network is set up with bandwidth of 10 Mbps and a propagation delay of 10 ms. For the two and five flow networks, the queue size was set to 100 packets, because there would be loss when transferring two or five 1 MB files. For the one flow network, however, I needed to set the queue as 80 in order to see any loss. I also ran the one flow experiment with a queue size of 100, for comparison.
Experiment 1 required the transferring of a 1 MB file over the link. The exported data for Experiement 1 is saved in the *lab3/1_data* directory, in the text files that begin with **1**. Figures 1 through 8 show the plots constructed from the data received from this experiment. *Figure 1 - Queue Size vs Time with Max Queue Size of 80*

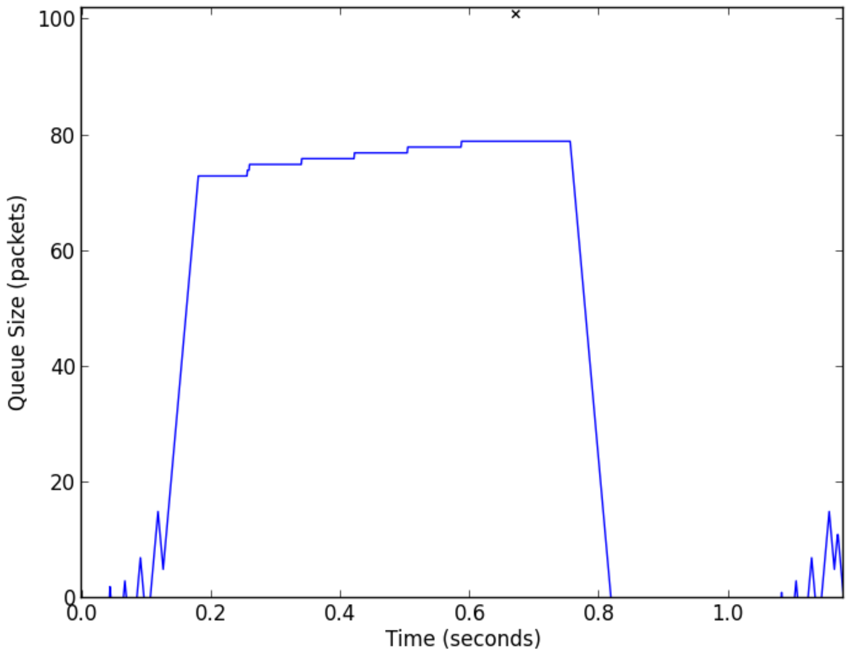


Figure 2 - Queue Size vs Time with Max Queue Size of 100

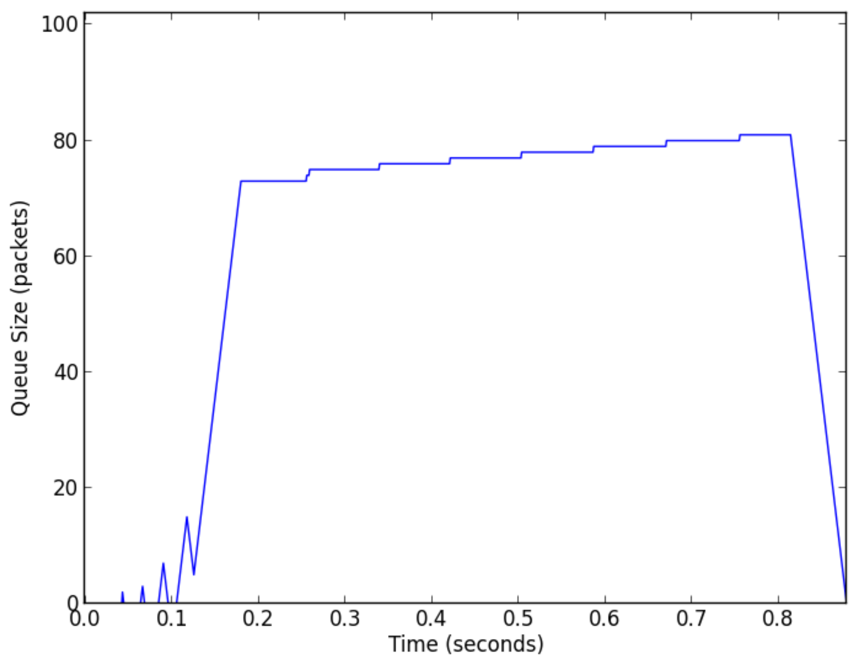


Figure 3 - Receiver's Rate vs Time with Max Queue Size of 80

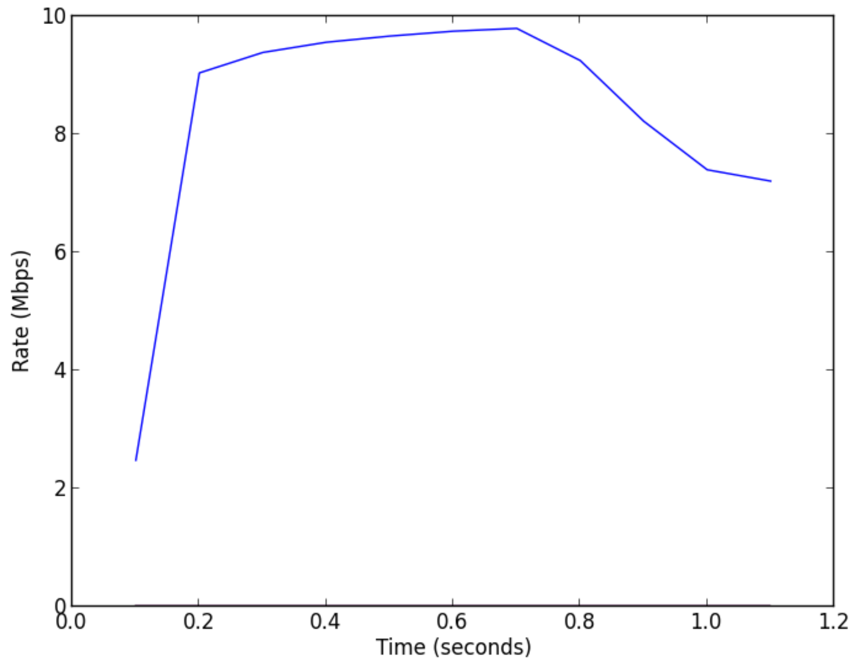


Figure 4 - Receiver's Rate vs Time with Max Queue Size of 80

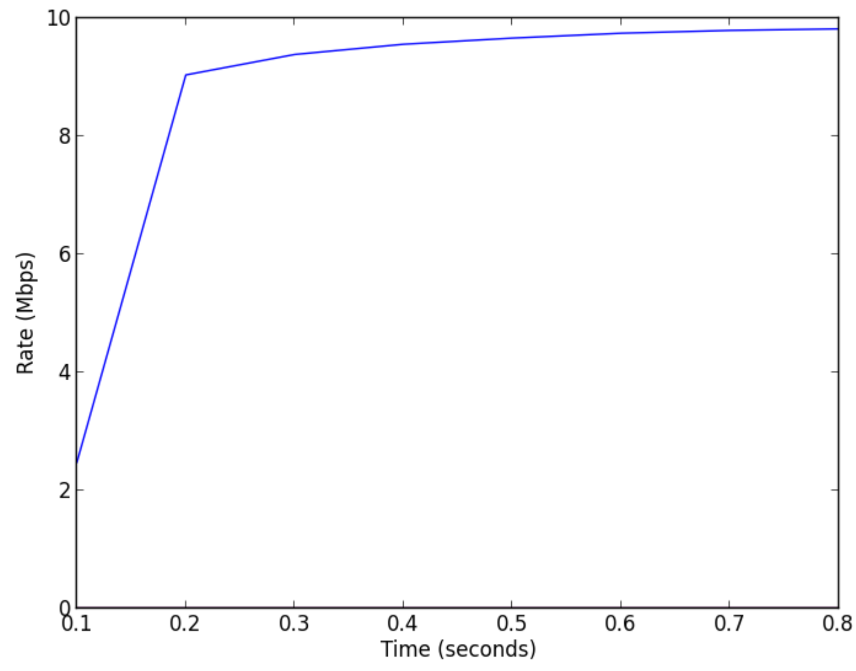


Figure 5 - Congestion Window vs Time with Max Queue Size of 80

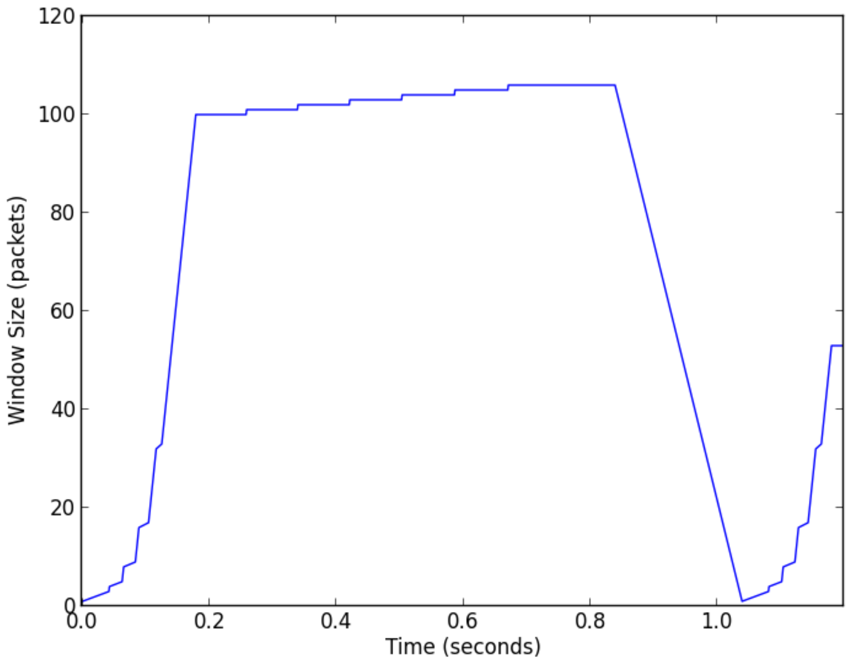


Figure 6 - Congestion Window vs Time with Max Queue Size of 80

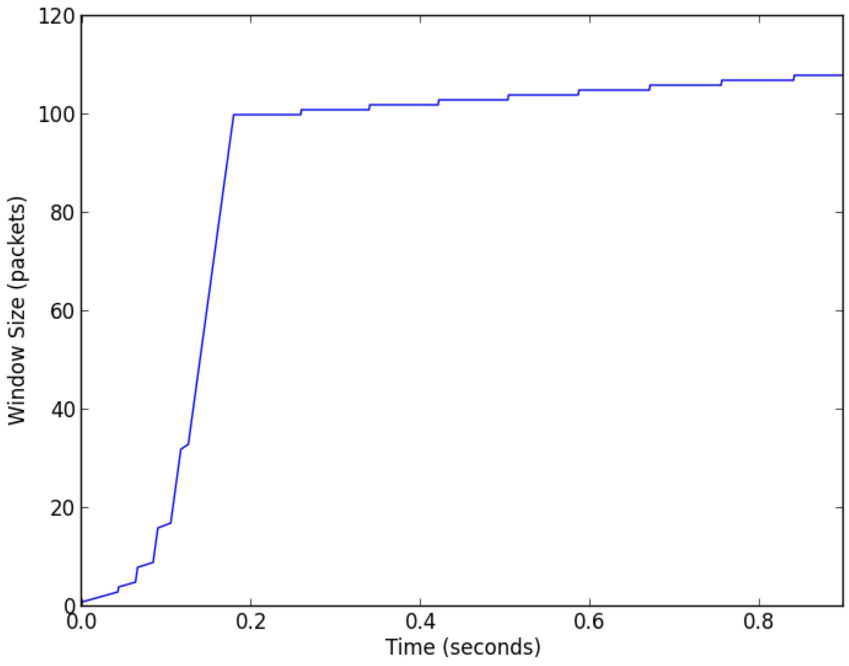


Figure 7 - Sequence No., ACK, and Packet Loss over Time with Max Queue Size of 80

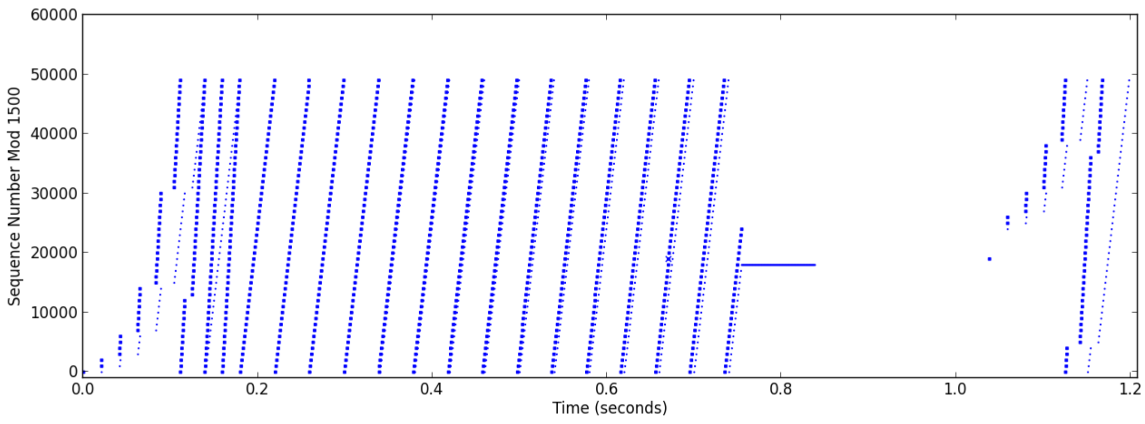
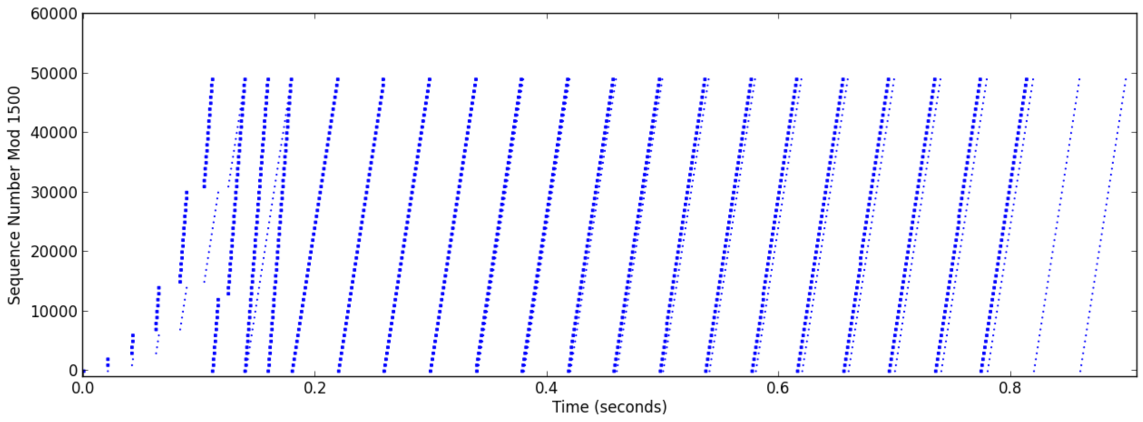


Figure 8 - Sequence No., ACK, and Packet Loss over Time with Max Queue Size of 100



Experiment 2 required that there be 2 flows accessing the network at the same time in order to perform a file transfer over each flow. The exported data for Experiment 2 is in the text files (*.txt* extension) in the *2_data* directory. The plots created from the data are displayed as figures 9 through 13. *Figure 9 - 2 Flows, Queue Size vs Time*

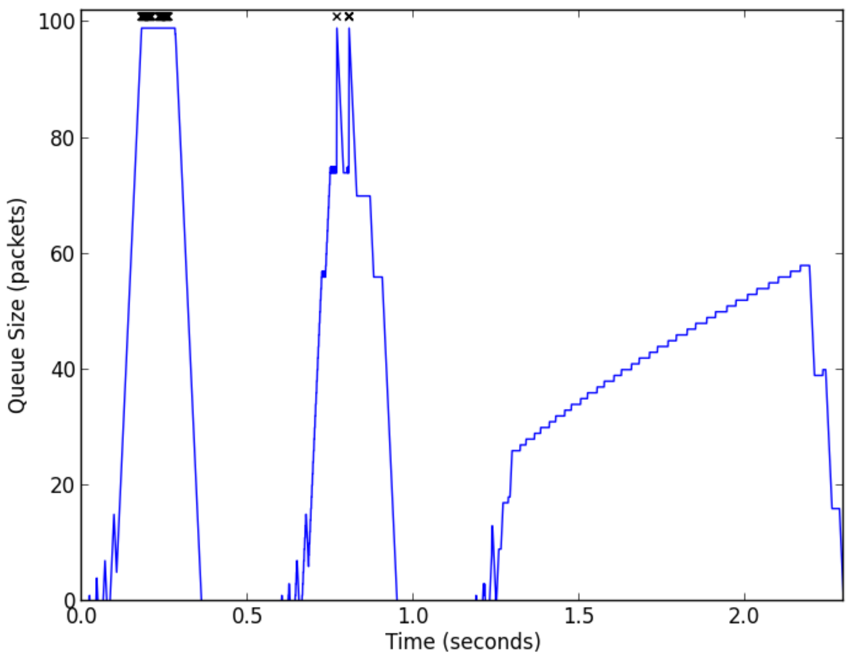


Figure 10 - 2 Flows, Receiver's Rate vs Time

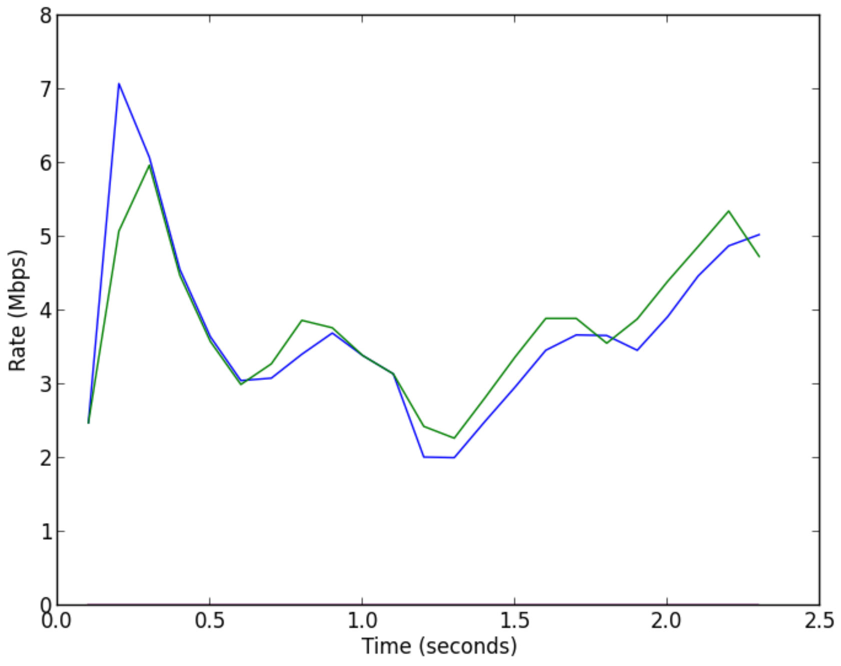


Figure 11 - 2 Flows, Flow 1 Congestion Window vs Time

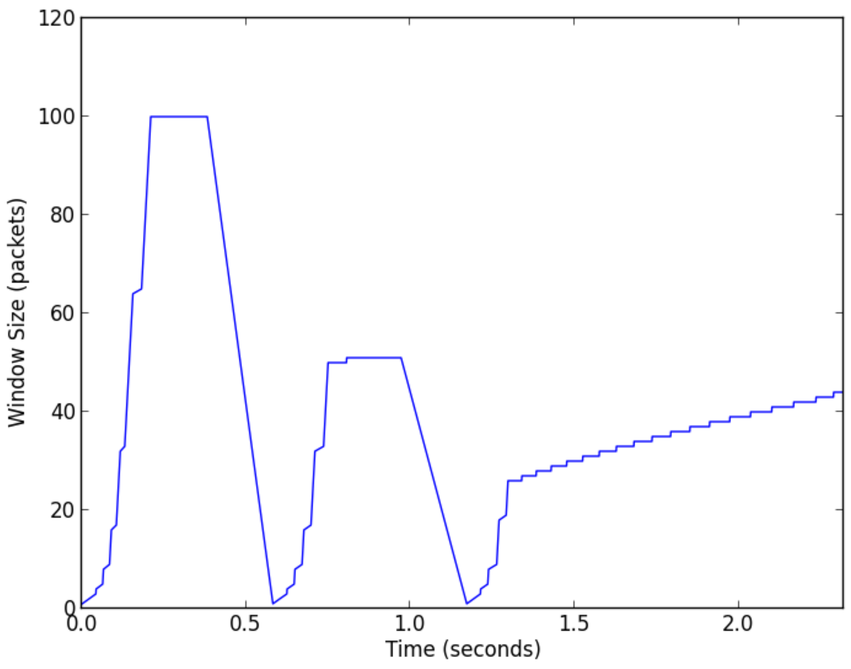


Figure 12 - 2 Flows, Flow 2 Congestion Window vs Time

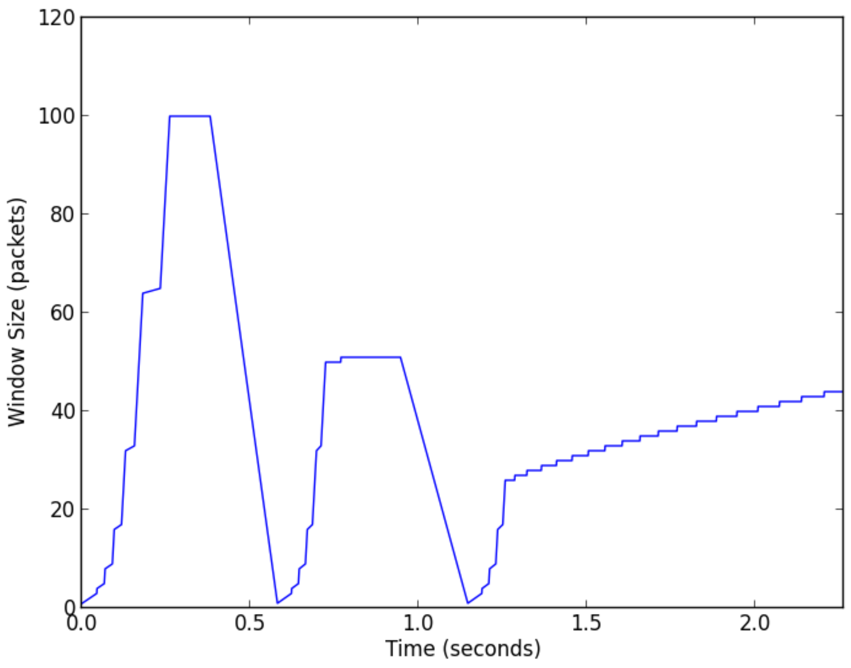
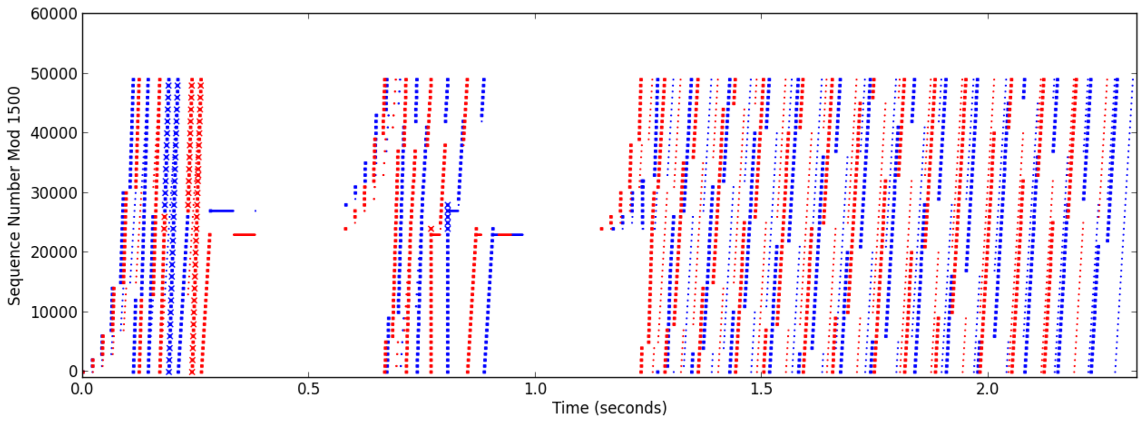


Figure 13 - 2 Flows, Sequence No., ACK, and Packet Loss over Time



Experiment 5 used the same network set up as experiments 2 and 1, but this time there were five file transfers, and each one started at a different time in the Simulator. This is apparent in the data and on the graphs below. The data for experiment 5 is saved in the *5_data* directory. Figures 14 through 21 show the plots created from the data:

Figure 14 - 5 Flows, Queue Size vs Time

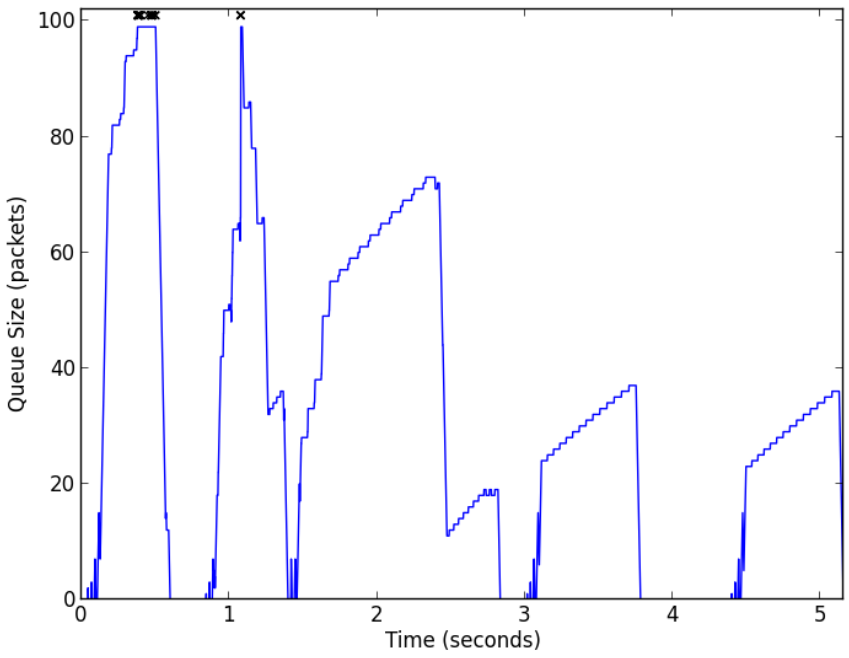


Figure 15 - 5 Flows, Receiver's Rate vs Time

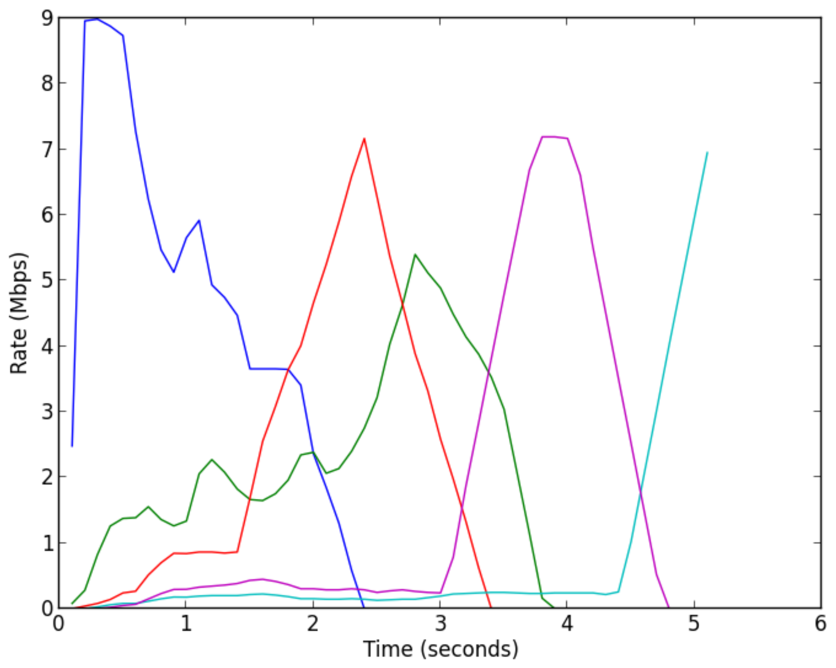


Figure 16 - 5 Flows, Flow 1 Congestion Window vs Time

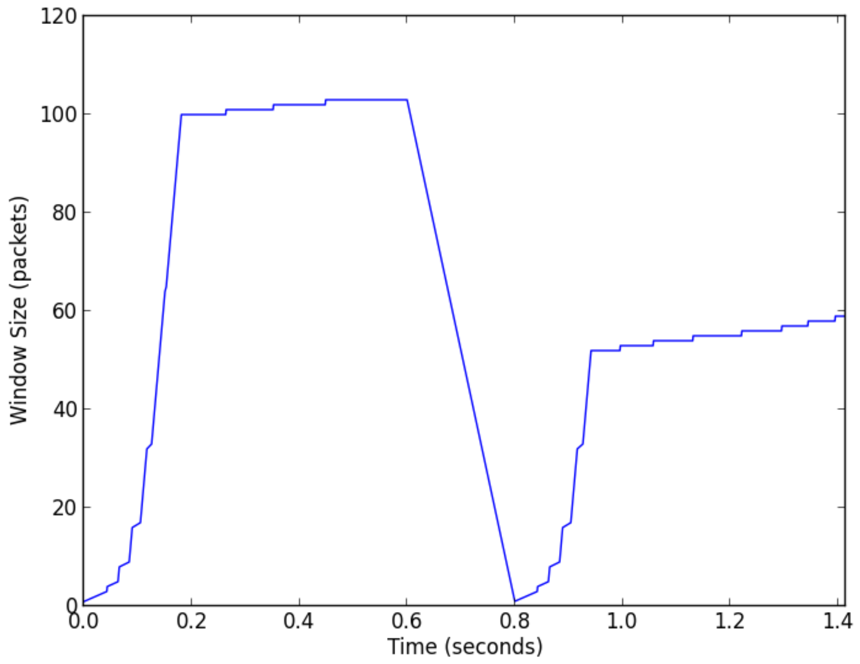


Figure 17 - 5 Flows, Flow 2 Congestion Window vs Time

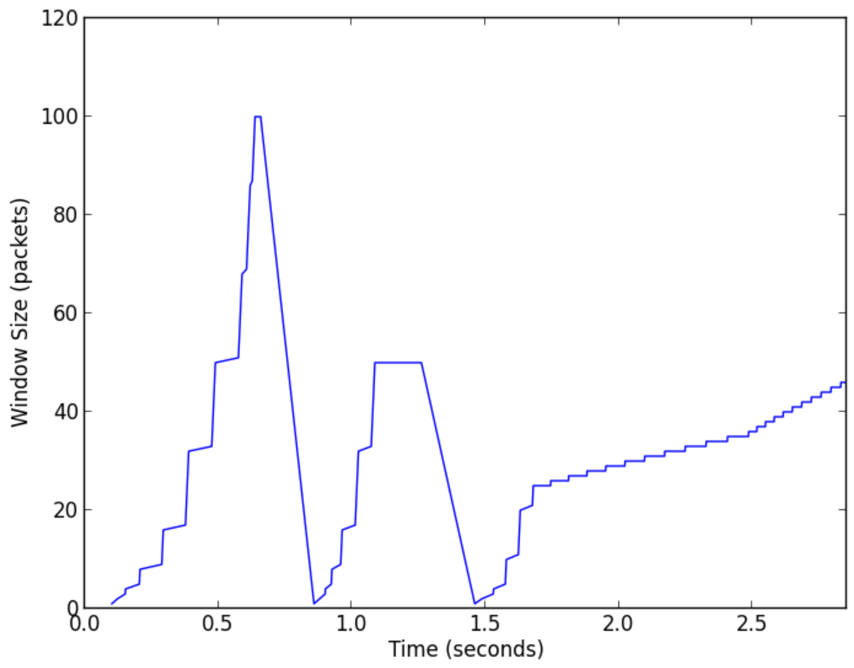


Figure 18 - 5 Flows, Flow 3 Congestion Window vs Time

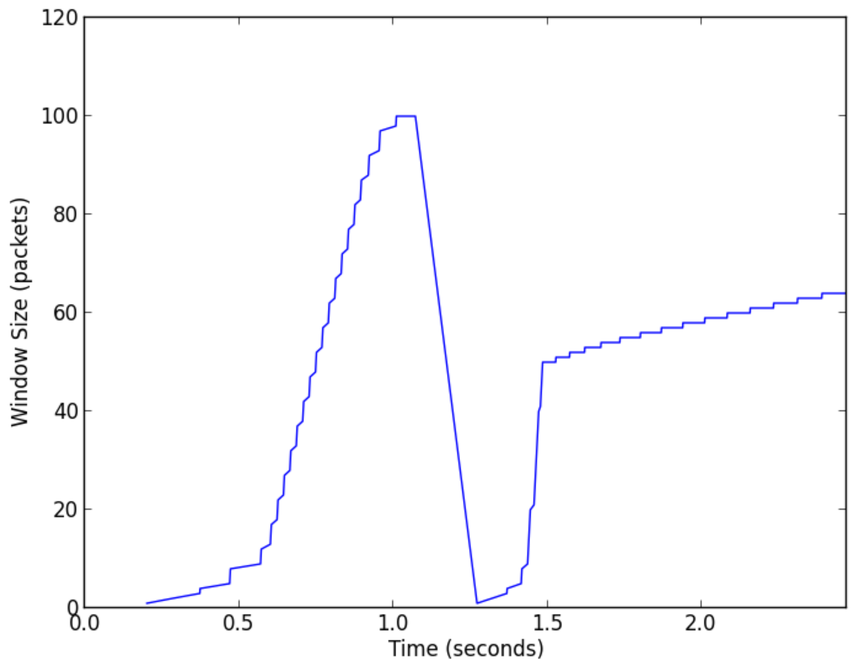


Figure 19 - 5 Flows, Flow 4 Congestion Window vs Time

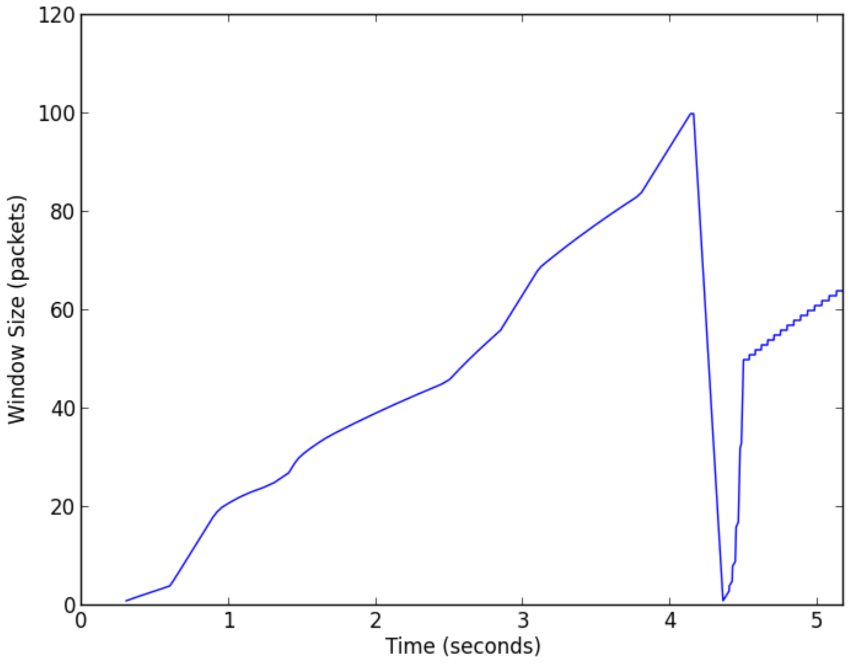


Figure 20 - 5 Flows, Flow 5 Congestion Window vs Time

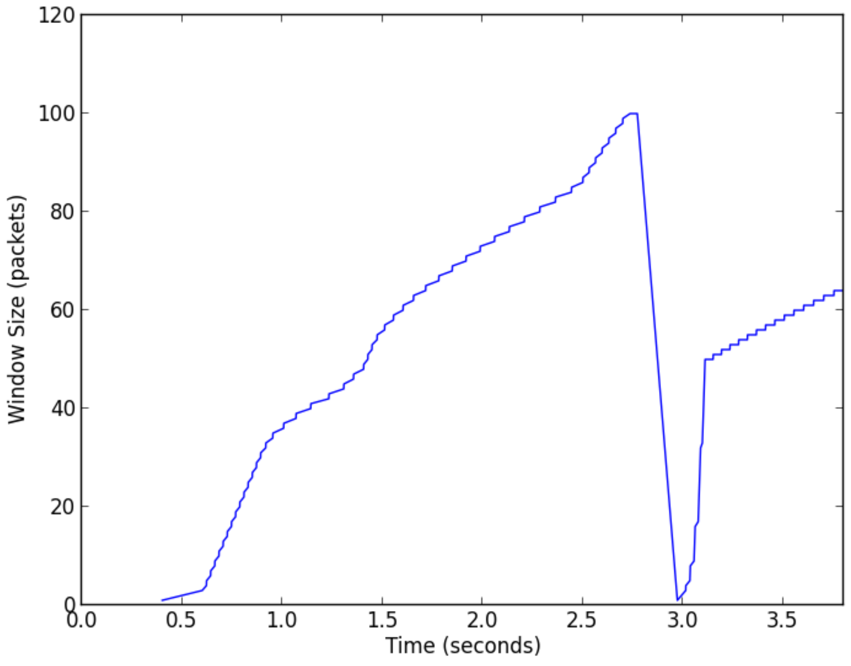
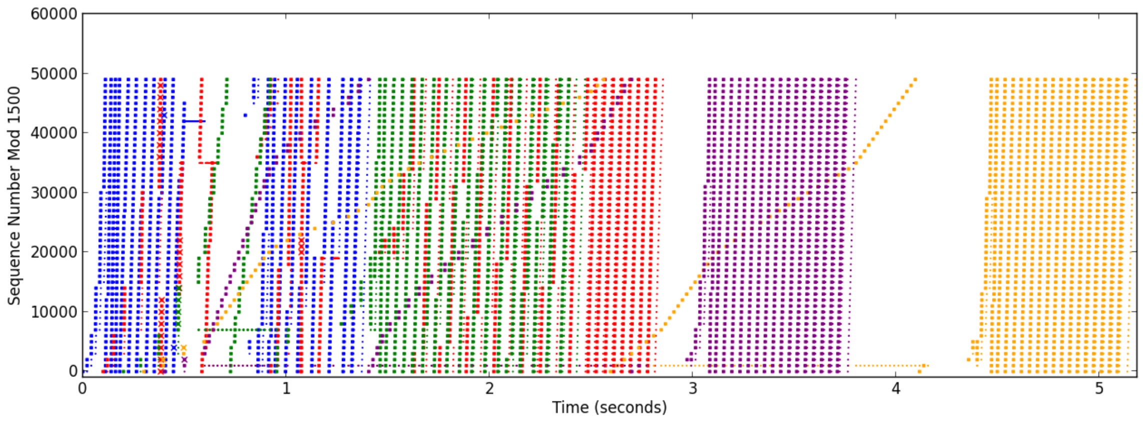


Figure 21 - 5 Flows, Sequence No., ACK, and Packet Loss over Time



3 Conclusion

After running the experiments, I believe that the data I collected shows that my implementation of TCP Tahoe is correct. The queue plots have the shape expected. The rate plots show the interaction between the multiple flows and how they are affected by each other. The window plots show the window sizes adjusting correctly. And the sequence plots are also correct.