Lab 3 Report

Dallin Christensen

March 29, 2014

1 Congestion Control

Using my code from the previous lab (Reliable Transport), I was able to implement congestion control using TCP Tahoe with just a few modifications. Those modifications include the implementation of three new features: slow start, a slow start threshold, and AIMD (Additive Increase Multiplicative Decrease).

1.1 Slow Start

At the start of the connection and after any kind of loss event, the cwnd (congestion window) is set to 1 MSS. Every time the sender receives an ACK for new data, the cwnd is incremented by the number of new bytes of data acknowledged.

Set cwnd to 1 MSS at start of connection

```
1 def send(self,data):
2   self.send_buffer += data
3   self.cwnd = self.mss
4   self.send_if_possible()
```

Set cwnd to 1 MSS on loss event

```
1 def loss_event(self):
2   self.threshold = max(self.cwnd/2, self.mss)
3   self.cwnd = self.mss
```

1.2 Slow Start Threshold

Slow start ends when the congestion window exceeds or equals the threshold. Per the project specifications, the initial threshold was set at 100000 bytes.

1.3 AIMD

1.3.1 Additive Increase

Additive increase, rather than slow start, is used to increase the size of the congestion window once the congestion window is larger than the threshold. Every time the sender receives an ACK for new data, the congestion window increases by MSS*b/cwnd, where MSS is the maximum segment size (1000 bytes) and b is the number of new bytes acknowledged. The threshold is also increased to the size of the new congestion window.

1.3.2 Multiplicative Decrease

When a loss event is detected (a timeout), the threshold is set to max(cwnd/2,MSS) and cwnd is set to 1 MSS. By only cutting the threshold in half and by using slow start, we prevent throughput from decreasing significantly when loss rates are low. However, if multiple loss rates occur in succession, the threshold will quickly drop to 1 MSS.

Decrease threshold on loss event

```
def loss_event(self):
  self.threshold = max(self.cwnd/2, self.mss)
  self.cwnd = self.mss
```

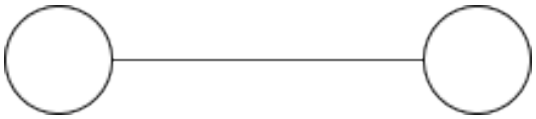
Slow start and AIMD

```
# the sender getting an ACK from the receiver
def handle_ack(self, packet):
  ...
  new_bytes = packet.ack_number - self.received_ack_number

  if self.cwnd >= self.threshold:
    # Additive Increase
    self.cwnd += (self.mss*new_bytes)/self.cwnd
    self.threshold = self.cwnd
  else:
    # Slow Start
    self.cwnd += new_bytes
```

2 Experiments

A simulator was used to set up a simple network consisting of two nodes and and one bidirectional link.



2.1 One Flow

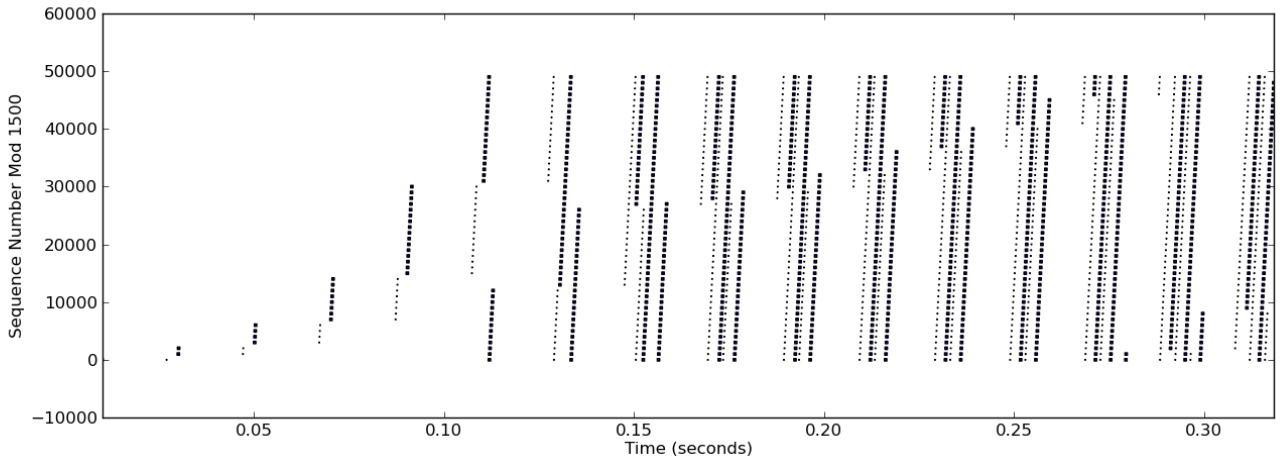
The link bandwidth was set to 10 Mbps, the propagation delay to 10 ms, and the queue size to 100000 bytes (or 100 packets). I transfered a 1 MB text file across the link.

2.1.1 One Flow - 0% Loss

Initially I tried this experiment with a 0% loss rate. The congestion window size starts out at 1 MSS and increases exponentially (using slow start) until it reaches the threshold. Because the threshold (100000 bytes) is the same size as our file, slow start increases the congestion window size at a rate faster than the node sends packets, so no queue ever forms and there is no queueing delay.

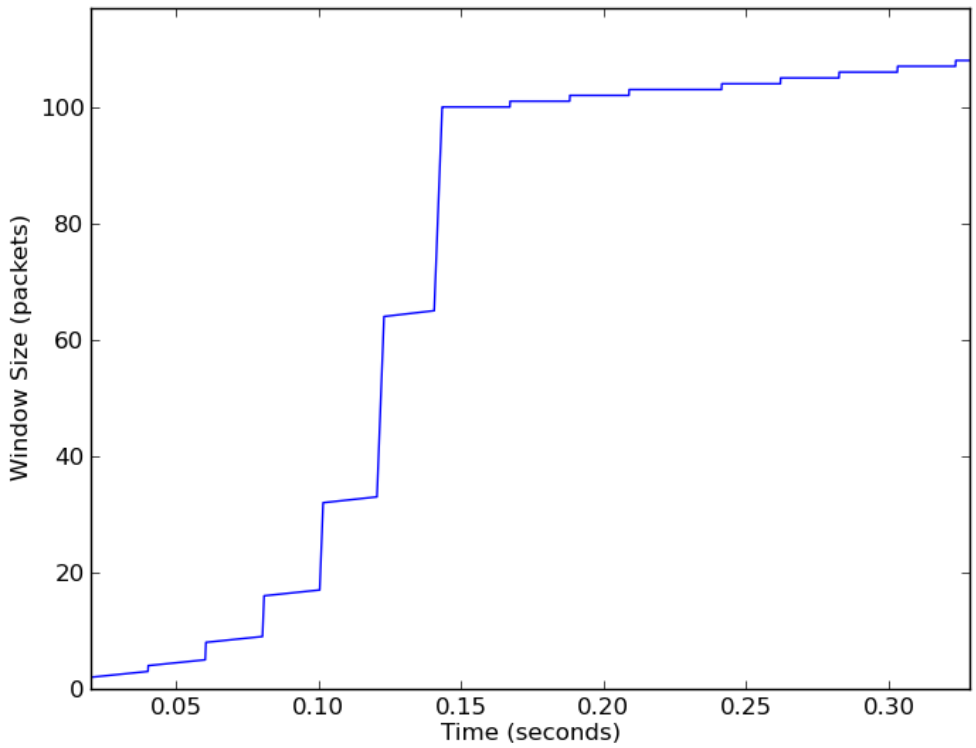
Slow start is evidenced in the sequence plot created from the initial experiment. To start off, only one packet is sent. Once the ACK for that packet is received, the congestion window is doubled in size and two packets are sent. Once those two ACKs are received, four packets are sent, then eight, etc.

Sequence Plot



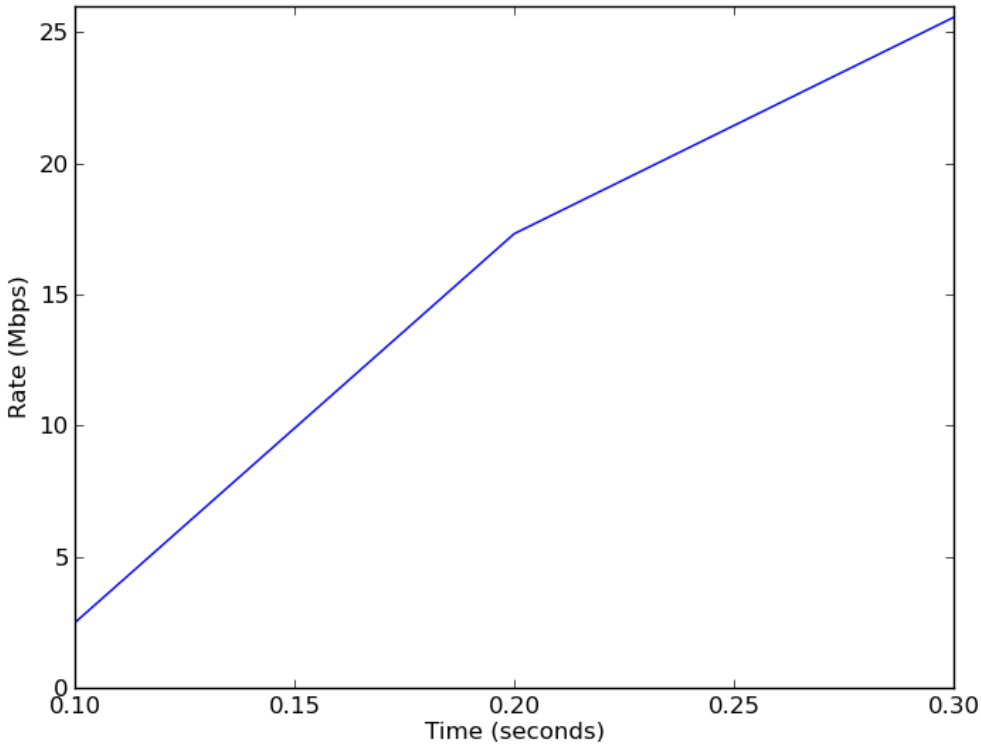
Slow start is also evidenced in the window size plot. The congestion window doubles in size until it reaches the threshold (100 MSS), then increases additively after that. Had there been a loss rate, we would expect a sawtooth pattern, showing that the congestion window gets cut in half with every loss. But because this experiment had a 0% loss rate, that sawtooth pattern is not shown.

Window Size Plot



The rate plot is clearly wrong, but I can't figure out why. I expected the rate to increase quickly until it reached the 10 Mbps bandwidth of the link, platueing there until the file finished transfering. However, when the experiment was run, the rate continued well beyond the 10 Mbps bandwidth and never platued.

Rate Plot

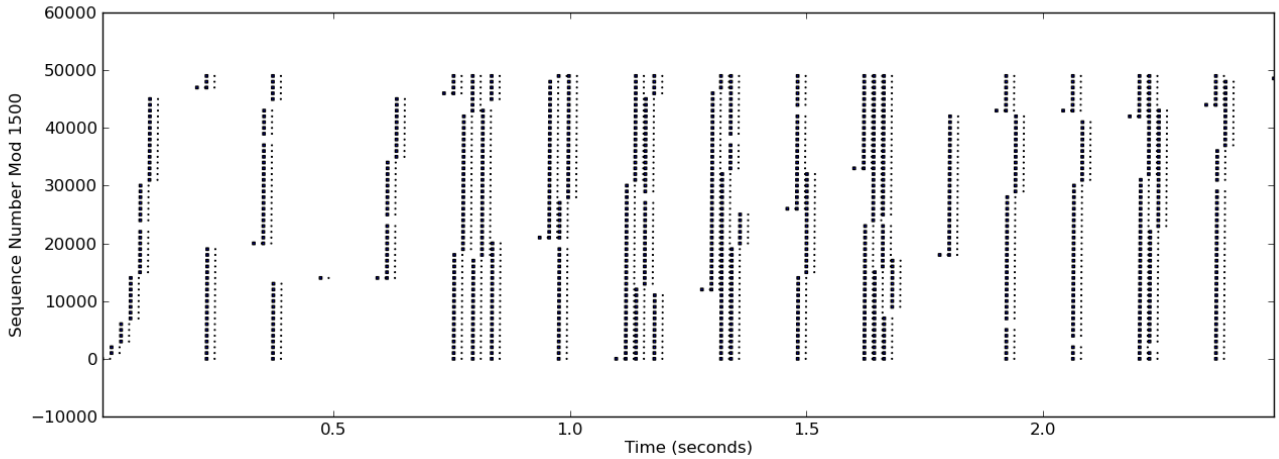


2.1.2 One Flow - 1% Loss

The experiment was run again, but with a 1% loss rate.

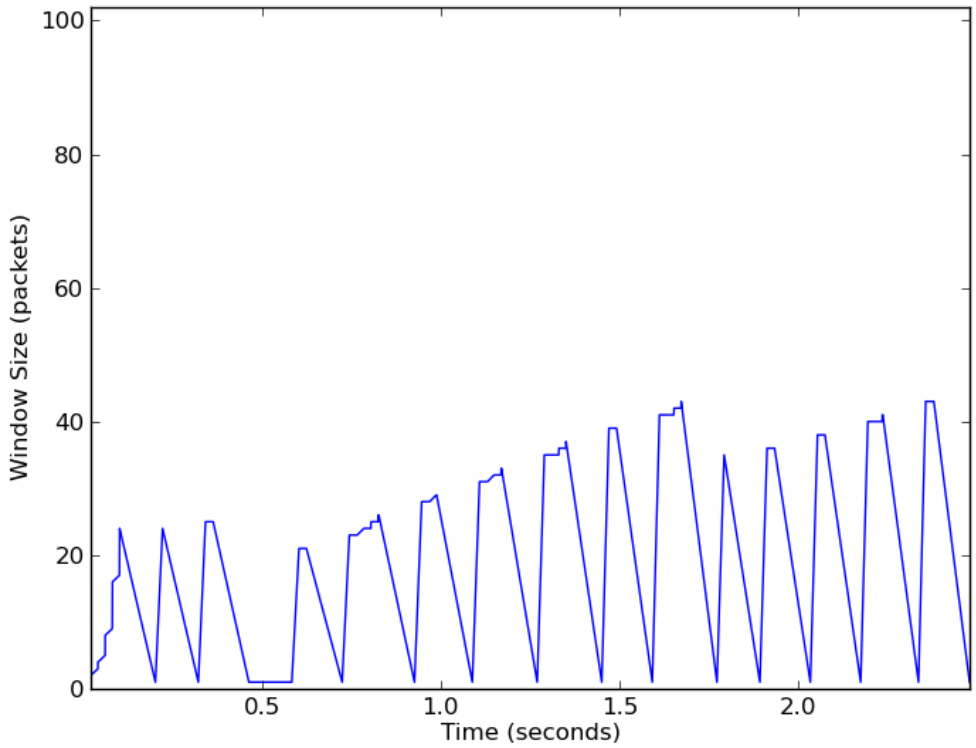
Loss events are noticeable on the sequence plot by the 0.1 second timeout that triggers a retransmission. Slow start is visible at the beginning of the connection, but does not seem to occur after retransmission events. This seems to contradict the code, but it was the observed behavior.

Sequence Plot



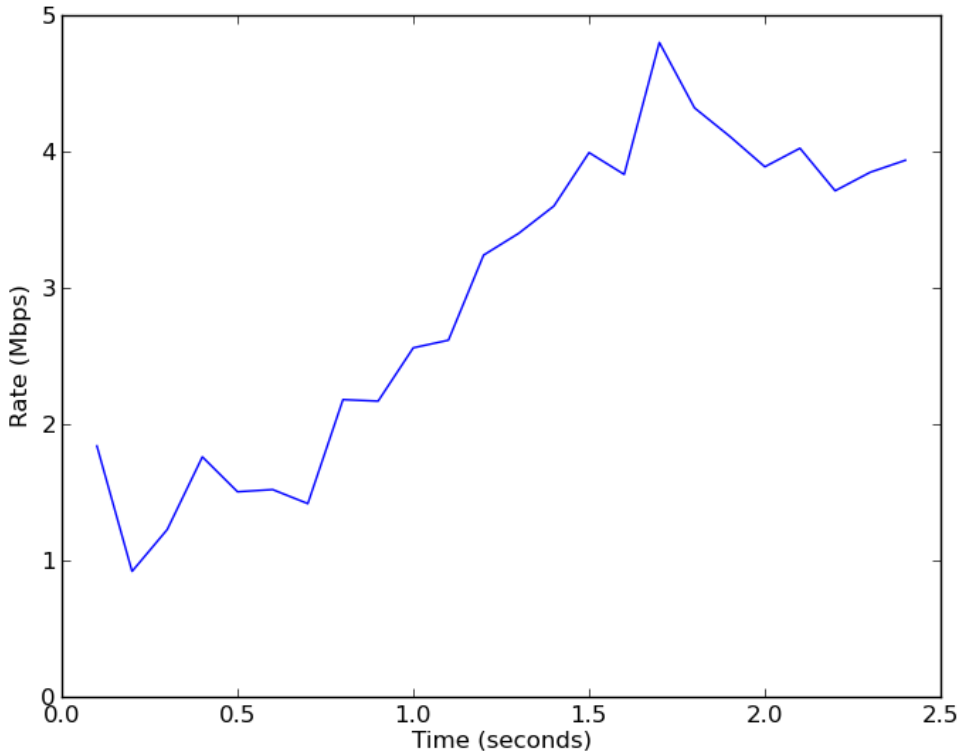
The expected sawtooth pattern is visible in the window size plot. The congestion window gets cut down to 1 MSS with each loss event, and the window size never gets larger than the initial threshold as it did with a 0% loss rate. The largest it ever got was about 45 packets.

Window Size Plot



When the experiment was rerun with a 1% loss rate, the rate was a much more reasonable speed (5 Mbps at it's peak) than the initial experiment.

Rate Plot



The loss events are noted on the queue plot, which is not included in this report, but can be viewed in the attached source files.

2.1.3 One Flow - 5% Loss

The experiment was run again with a 5% loss rate. As expected, loss events occur more frequently than they did in the previous experiments, occuring in a lower rate and smaller congestion window. These observances can be seen in the graphs included with the attached source files.

2.2 Two Flows

I repeated the same experiment, but with two TCP flows (using different ports). Each flow transferred the same 1 MB text file and they started at the same time. Unfortunately, I was unable to get the proper expected behavior when using two flows, even after countless attempts and many frustrating hours.

2.3 Five Flows

I also repeated the experiment with five TCP flows (using different ports). The flows had a staggered start, each one starting 0.1 seconds after the previous one. Unfortunately, as with the two flows, I was unable to get the proper expected behavior when using five flows, even after countless attempts and many frustrating hours.