# Lab 3 Report

Nate Fox

## 1 Congestion Control

### 1.1 Class Variables

To implement congestion control, I started by changing some of the TCP class variables. I renamed the `window` variable to `cwnd` in order to match the variable names from the textbook, and I initialized `cwnd` to be one MSS. I also introduced the `ssthresh` variable to keep track of the slow start threshold. Finally I introduced the `ack_count` variable to keep track of duplicate ACKs.

### 1.2 Receiving New ACKs

When I receive ACKs for newly acknowledged data, I update my congestion window according to what state I am in. If my current `cwnd` is less than `ssthresh`, I consider myself in slow start mode and I increment `cwnd` by the number of new bytes acknowledged. If my current `cwnd` is greater than or equal to `ssthresh`, I consider myself in AIMD mode and I increment `cwnd` by (MSS * new_bytes / `cwnd`). I also reset my `ack_count` to 0.

### 1.3 Receiving Duplicate ACKs

When I receive a duplicate ACK, I increment my `ack_count`. If `ack_count` is three after I increment it, I retransmit without restarting the retransmission timer or updating the RTO.

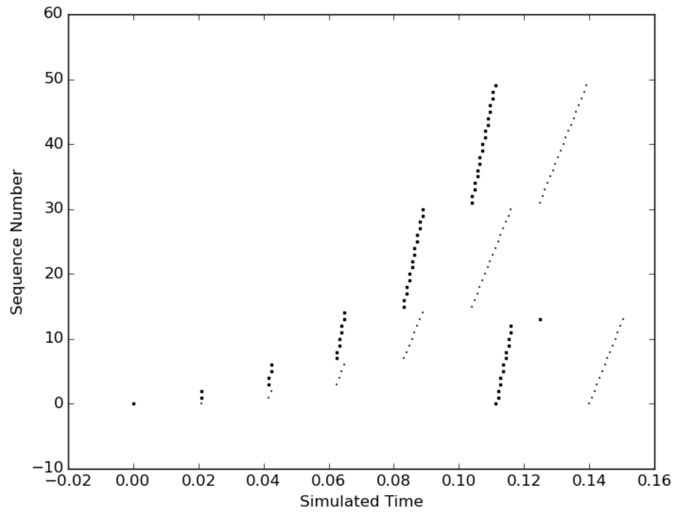
### 1.4 Retransmitting

Whenever I retransmit, either due to the retransmission timer firing or due to three duplicate ACKs, I update the `ssthresh` and the `cwnd`. I set `ssthresh` to one-half of `cwnd` (to a minimum of one MSS) and I reset `cwnd` to one MSS.

## 2 Tests

### 2.1 Slow Start

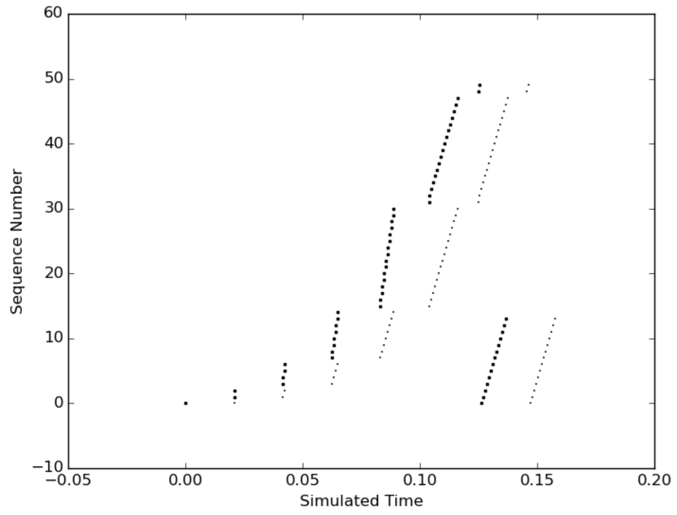
To test the slow start mode, I set the queue size on the link to 100 packets and I set the `ssthresh` to 100,000 bytes. I transfer a small 64 kb file, which is guaranteed to not overwhelm the queue.



The above graph shows that `cwnd` starts at one MSS, then doubles each round. When the `cwnd` size is 64,000 bytes, there are only 1000 bytes left to send, which is why the final round shows only a single packet.

### 2.2 Additive Increase

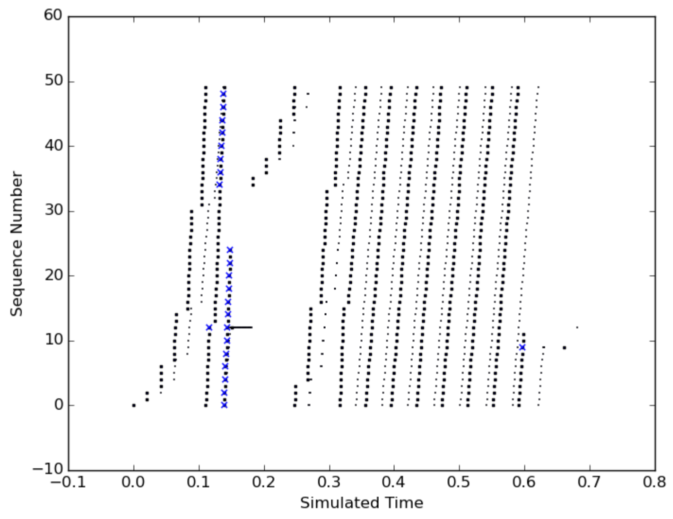
To test additive increase mode, I set the queue size on the link to 100 packets and I set the `sstrhesh` to 16,000 bytes. I transfer a small 64 kb file, which is guaranteed to not overwhelm the queue.



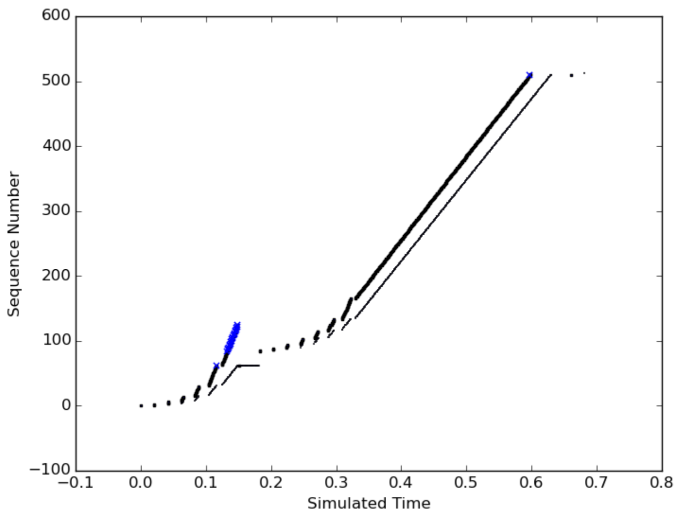
The above graph shows that `cwnd` starts at one MSS, then doubles each round until `cwnd` is 16,000 bytes. Once it hits the 16,000 byte threshold, it goes into additive increase mode. It sends 17 packets for the following round.

## 2.3 AIMD

To test AIMD, I reset the ssthresh to 100,000 and I set the queue size on the link to 15 packets, which causes a packet to be dropped once cwnd grows to 32,000 bytes. I transfer a 512 kb file.

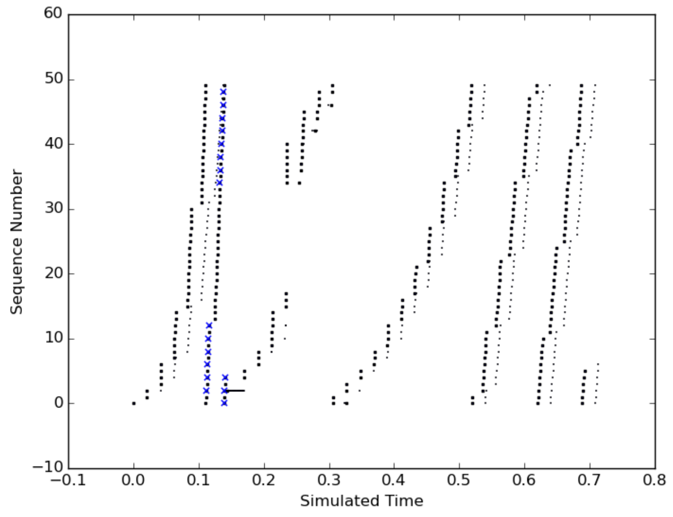


The above graph shows that cwnd starts at one MSS, then doubles each round until cwnd is 64,000 bytes. The last packet sent when cwnd is 32,000 bytes gets dropped, but the sender has not yet received the three duplicate ACKs, so it continues in slow start mode, which leads to multiple other packets getting dropped. Once the three duplicate ACKs are received, the sender resets cwnd to one MSS and goes back into slow start mode, but now with ssthresh set to 32,000 bytes. Once cwnd gets up to 32,000 bytes in slow start mode, it goes into AIMD mode. Below is the same graph without the modulo sequence numbers for added clarity:

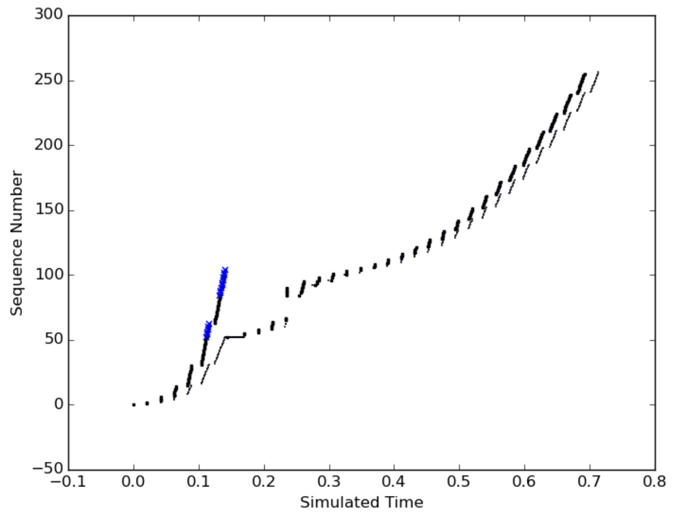


## 2.4 Burst Loss

To test the response to multiple packets being lost at once, I keep the ssthresh at 100,000 and I set the queue size on the link to 10 packets. I transfer a 256 kb file.



The above graph shows that multiple packets are dropped when the cwnd is at 32,000 bytes. The sender gets three duplicate ACKs for ACK values 52000, 84000, 92000, 96000, and 100000. This causes the ssthresh to drop from 100,000 bytes to 1,000 bytes over a 0.3-second period. Once the sender receives the ACK for 102,000, the sender is in a constant state of additive increase. Below is the same graph without the modulo sequence numbers for added clarity:


Sequence Number
Simulated Time