# Programming Project 3 - Congestion Control

Richard Voragen - 917981018
Jonathan Yeung - 918630041
Ulysses Morgan - 918847915

Team Name: Pee Pee Island
Custom Protocol Name: TCPeePeeIsland
Custom Protocol File: sender_custom.py

All Project Files: sender_custom.py, sender_stop_and_wait.py, sender_fixed_sliding_window.py, sender_tahoe.py, sender_reno.py

## 1 Description of Protocols

### 1.1 Custom Protocol

Our custom protocol runs as a combination of TCP Reno and sliding window protocol. During experimentation we noticed that we could either have a protocol like TCP Reno where the average packet delay is very high but the throughput is also very high. Or we could have a protocol similar to stop and wait where the average packet delay is very low but the throughput is also very low. This algorithm combines the best of both to get a higher throughput while keeping the average packet delay low.

The way this is accomplished is that on a window size reset we set the window size back to one and the SSThreshold to be equal to the max between 20 and the old window size. This is because as we approach 20 in slow start we are not moving much faster than congestion avoidance and because most networks can handle a window size of at least 20. The major innovation came in when instead of just resetting the window size on the event of a timeout or fast retransmit, we also reset the window size when the delay for a received packed is larger than a specified delay.

When we first start the code we run a function that will send out the first 10 packets with a stop and wait protocol and returns the maximum delay of the 10 packets. We then multiply that maximum value by 1.25 to get our maximum delay. Then we run the algorithm, which is very similar to the way TCP Reno runs but with a sliding instead of fixed window. If any of our singular packet delays exceed the given maximum delay value then we reset our window size as specified above. This implementation ensures that our average packet delay will be less than the specified maximum value and still allows us to get a throughput that is about 6 times faster than the stop and wait protocol.

### 1.2 Stop-and-wait protocol

In our stop-and-wait protocol, we send out one packet and wait for an acknowledgement. Once it receives the acknowledgement for the packet from the receiver, it sends the next packet in the sequence. Only one packet is transmitted at any time and we incorporated a timeout of 0.5 seconds in case an acknowledgement isn't received in time to reduce average packet delays.

### 1.3 Fixed sliding window protocol

Our fixed sliding window protocol sends 100 packets at a time and waits for the corresponding acknowledgements from the receiver. Once the sender receives these acknowledgements, our window slides and sends out the next packets in the sequence. Compared to our stop-and-wait protocol we are able to improve bandwidth utilization as multiple packets are transmitted at any given time.

### 1.4 TCP Tahoe

Our TCP Tahoe implementation borrows elements from the sliding window protocol. During slow start, the window size increases exponentially until the SSThreshold of 64 is reached. Once it is passed, we increased the window size linearly instead due to congestion avoidance. In the case of packet loss or timeout, we re-entered the slow start phase and set the SSThreshold to $\frac{1}{2}$ * window size and window size to 1.

### 1.5 TCP Reno

Our implementation of TCP Reno shares similarities with how we programmed TCP Tahoe, such as how we used a sliding window of increasing size to send out packets. However, for a fast retransmit after 3 duplicate packets are received, we set the SSThreshold to $\frac{1}{2}$ * window size and then set the window size to SSThresh + 3 (the +3 accounts for the 3 duplicate packets received).

## 2 Throughput, delay, and performance metrics

Table 1: Stop-and-wait protocol

| | **Average** | **Standard Deviation** |
|---|---|---|
| Throughput | 9,709.44 | 47.09 |
| Average Packet Delay | 0.10 | 0.00 |
| **Performance Metric** | **92,911.78** | **940.85** |

Table 2: Fixed sliding window protocol

| | **Average** | **Standard Deviation** |
|---|---|---|
| Throughput | 85,292.01 | 2,301.79 |
| Average Packet Delay | 1.18 | 0.03 |
| **Performance Metric** | **72,204.90** | **3,799.75** |

Table 3: TCP Tahoe

| | **Average** | **Standard Deviation** |
|---|---|---|
| Throughput | 69,051.64 | 4,278.42 |
| Average Packet Delay | 0.59 | 0.07 |
| **Performance Metric** | **118,871.83** | **11,263.89** |

Table 4: TCP Reno

| | **Average** | **Standard Deviation** |
|---|---|---|
| Throughput | 81,582.78 | 2,638.44 |
| Average Packet Delay | 0.62 | 0.04 |
| **Performance Metric** | **131,918.13** | **8,793.71** |

Table 5: Custom protocol (ORIGINAL, NO MULTITHREADING)

| | **Average** | **Standard Deviation** |
|---|---|---|
| Throughput | 63,412.07 | 1,051.03 |
| Average Packet Delay | 0.18 | 0.00 |
| **Performance Metric** | **347,617.21** | **7,519.69** |