Discussion:

1) The stop and wait performance for drops is roughly 200 (1% of 20k) and the run time is roughly 700,000usec. For the sliding window when at window size of 1 it's drops is also roughly 200 (1% of 20k), and it's run time is greater at 750,000. This increase in run time is because more calculations are being done with more variables but since the window size is the same as stop and wait there is no benefit. However, once the window size is greater like at 30 its drops are roughly 160 a difference of roughly 40 drops. The run time at window size 30 is also lower at roughly 450,000usec an improvement from the stop and wait of 250,000usec. These benefits are from sending more messages at once and better utilizing the available bandwidth. This better bandwidth utilization also allows for less delay and chance for timeouts reducing the number of drops.

2) As discussed a bit before above the window size makes a big difference on the performance on the sliding window algorithm. For the sliding window when at window size of 1 its drops are roughly 200 (1% of 20k), and its run time is greater at 750,000. Then, once the window size is greater like at 30 its drops are roughly 160 a difference of roughly 40 drops. The run time at window size 30 is also lower at roughly 450,000usec which is an improvement from a window size of 1 of 300,000usec. These increases happen I would say fairly linearly in our simulated tests where the window size is increasing linearly. In situations with congestion avoidance and slow start, we would see both linear growth and exponential growth in the benefits. But at a base level, I think the benefits of a bigger window size are just linear.

3) In the first question, I kind of touched a bit on the impact of the number of resends so here I'll focus on the parts of my algorithm that affected that. First off I used selective ACKs so my server could send the highest in-order ACK that it had. I was also storing any out-of-order ACKs that were received allowing the client to not have to resend the whole window. This has a direct benefit from the client not resending the whole window and a benefit from how I also simulated* a decrease in the amount of resends based on the window size that would be expected to be seen in a real network. I did this because with a constant drop chance of 1% that doesn't change based on the window size the number of resends would always be roughly 200.

* A note on how I attempted to simulate the decrease in resends when the window size increases that are more likely to occur in a real network. I decided to simulate a decrease in the number of resends as the windowSize increases. So that a rough decrease in resends would occur. The first step to doing this was using selective ACKs. So now when an out-of-order message arrives it is stored until it can be ACKed properly. However, unlike sending the ACK messages which has a 1% chance of being dropped, this logging of the out-of-order messages does not have a chance of being dropped. This then means that the out-of-order messages never have a chance of being

dropped. This is also affected by the number of drops occurring from the 1% drop chance and how many messages come in the window after the one that is dropped. These variables give the sliding window algorithm a much more dynamic chance of less packets to be dropped when the windowSize increases, this effect also increases as the windowSize increases. I think that this then helps simulate less drops from less delay with the messages going through the network when using a sliding window.