**Discussion**

1. Stop&wait protocol and sliding window protocol are very different forwarding techniques, but they still share a bit of similarities. Such as, both require acknowledgement from server, both have time out mechanism, and etc. The main attribute about sliding window protocol that set itself apart from stop and wait protocol is of course its windows. Stop and wait protocol’s window size is always limited to one. Whereas sliding window protocol’s window size can be any size (within reason). With the increased window size, sliding window can send a burst of packets to the server. It does not have to wait for any acknowledgement until it hits the window limit. This greatly improve sliding window’s ability to send more packets and more often as well. Stop and wait on the other hand requires the sender to wait until the retriever gets the packet first and then send back an acknowledge before itself can operation again. This greatly hinder the sender’s performance because the sender will be idle while waiting. While being idle, the sender cannot do anything besides wait which mean it is wasting valuable resources.

1. Sliding window with only size 1 of window size is essentially stop and wait. We can see on our result that stop and wait’s performance is the worst between the two. The benefit of sliding window is that it can send more packets before waiting on an acknowledgement from retriever. The bigger the window is, the more packets can be sent at once. This will result in improvement on performance. But bigger does not always mean better. As the window size get to bigger and bigger, more packets can be sent, but like every other packet sent, they need an acknowledgement from the retriever to be consider fulfilled. In a large window, if one packet gets timed out, the protocol then has to reset back to the original state and resends all unacknowledged packets. This brings a lot of volatility to the protocol because at any moment, all non-acknowledged packets can be lost. Another disadvantage of the big window size is that eventually it will hit a threshold *(figure 1)*. As stated before, all packets are at the mercy of retriever. When the packets within the window size are sent, which mean that all of them are now waiting for acknowledgement. This will result in the sender becoming idle. The bigger the window is, the longer the waiting time will be. Protocols with smaller window size won’t have the initial burst of speed but as time passes, they eventually could catch up to the big windowed protocols.

*Figure 1 – shows increasing of window size will eventually cause the speed to converge*

*A close up of text on a black background

Description automatically generated*

1. While it is true that stop and wait protocols are slower than sliding window protocols, but it doesn’t mean that stop and wait don’t have their own advantages. Stop and wait protocol in fact are more reliable than sliding window. Based on our result, we can see that stop and wait has less retransmissions than sliding window *(figure 2)*. The reason for this is because stop and wait take its time to ensure the correct acknowledge is received before sending the next packet. Sliding window on the other hand release as many packets as much as possible then wait for acknowledgements. The trade-off for sliding window is speed for reliability and for stop and wait is reliability for speed.
2. The performance of any protocols heavily dependent on how reliable the connection is. As discussed above, stop and wait is more reliable than sliding windows. We can further see more of the difference here on *(figure 2)* that window size of 30 has to retransmit exponential amount as the drop chance increases. Whereas, the increase of retransmission with window size of 1 is nowhere near the amount size 30 is experiencing. The reason for such growth in retransmission for size 30 because the protocol can send 30 packets in sequential matter without waiting for confirmation from retriever. With the higher the chance of a packet drop, once one packet drops and time out occurs, the sender will need to resend all of its packet again. For example, take the highest drop chance being 10%, which mean a packet has 90% of being fulfilled. For all 30 packets to send and acknowledge without any packet drop, the probability of that happen can be model like so (9/10) ^30. The result is 9e-30. As we can see that the number is so close to zero which mean it is close to impossible for all 30 packets to make a safe round trip. In size 1 window case, the protocol only needs to wait for only 1 acknowledge at a time. This allow the protocol to have more head room to fix one mistakes verse the maximum 30 different mistakes.

*(figure 2) – shows bigger window can cause more packet to drop and decay its speed*

A screenshot of a cell phone

Description automatically generatedA screenshot of a computer

Description automatically generated