Part 2:

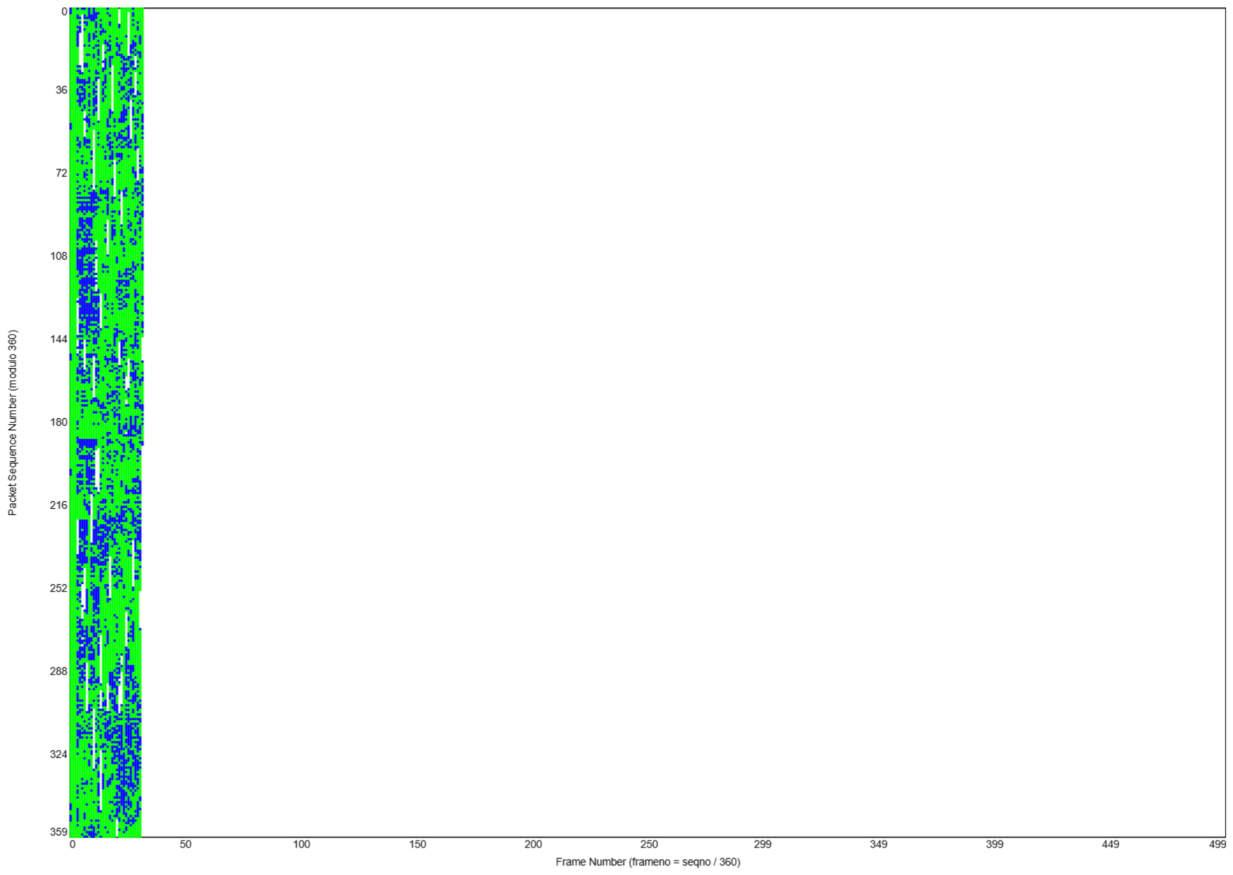
I ran the stop and wait protocol for a few minutes and sent 1030 packets, or 1.48 MB of data. The throughput it achieved was 8.14 KB/s. The entire video is 180,000 packets, and each packet's payload is 1440 bytes, so there's around 259.2 MB of data which would take about 8.8 hours at 8.14 KB/s.

Using ping, the RTT is about 173.8 ms, and the two datacenters I chose are located in Asia and America so this ping makes sense. This is pretty consistent with the observed behavior. Using the RTT, we can calculate that the throughput would be 1440B/.173s = 8.32KB/s, which is similar to the observed 8.14KB/s.

After sending 1030 packets, there were no duplicates and no losses.

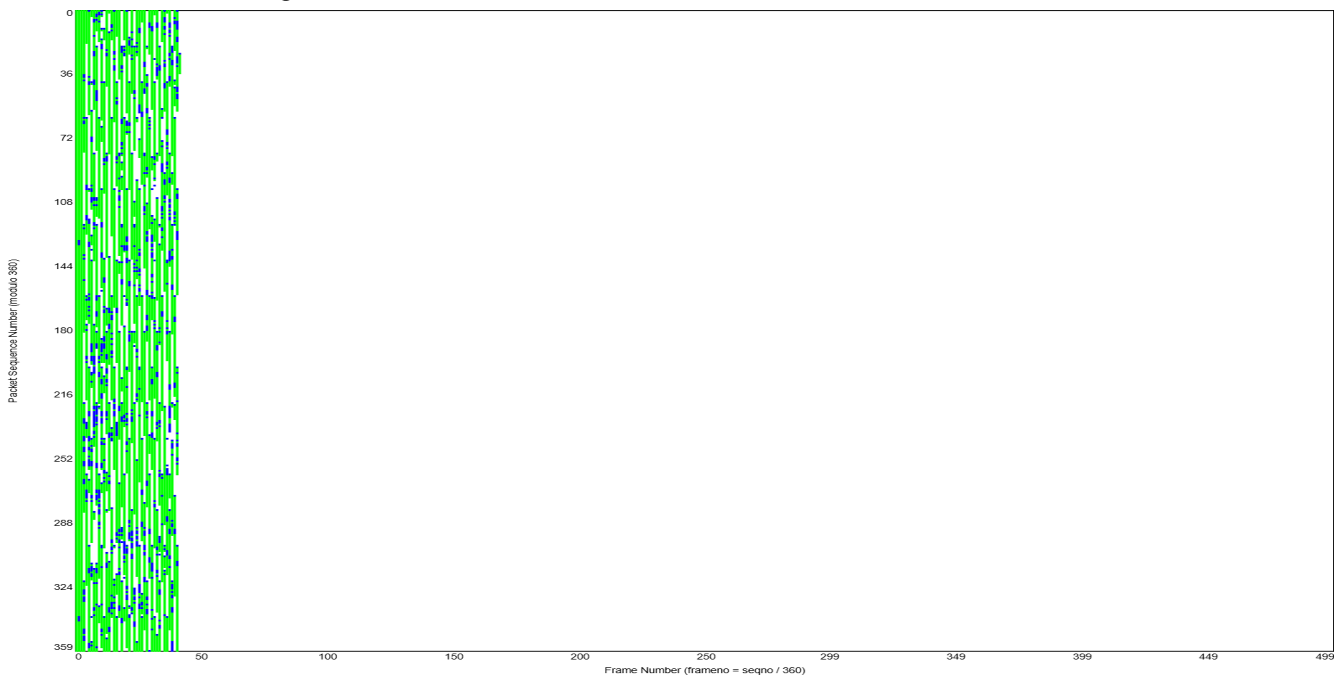
Part 3:

First time running burst1 I used N=50 and T=200 ms.



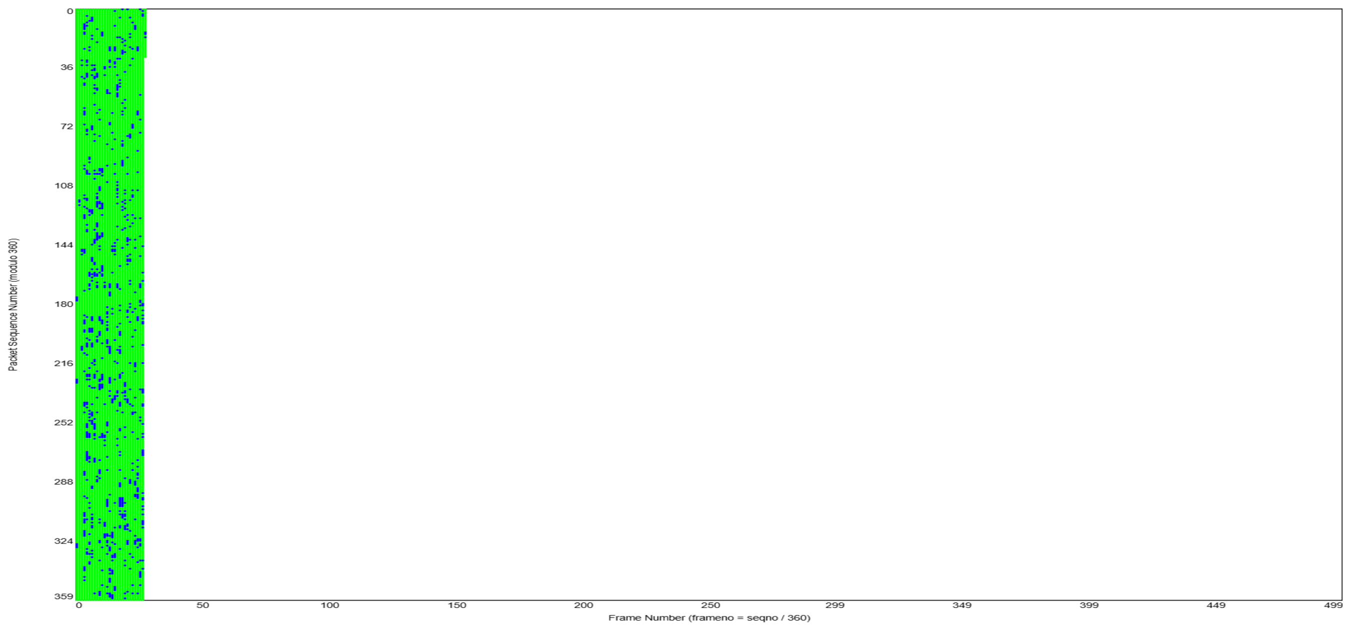
There were probably a few hundred missing packets, and a lot of out of order packets. The throughput was 326 KB/s, a lot faster than stop and wait.

Second time running burst1 I used N=100 and T=200 ms.



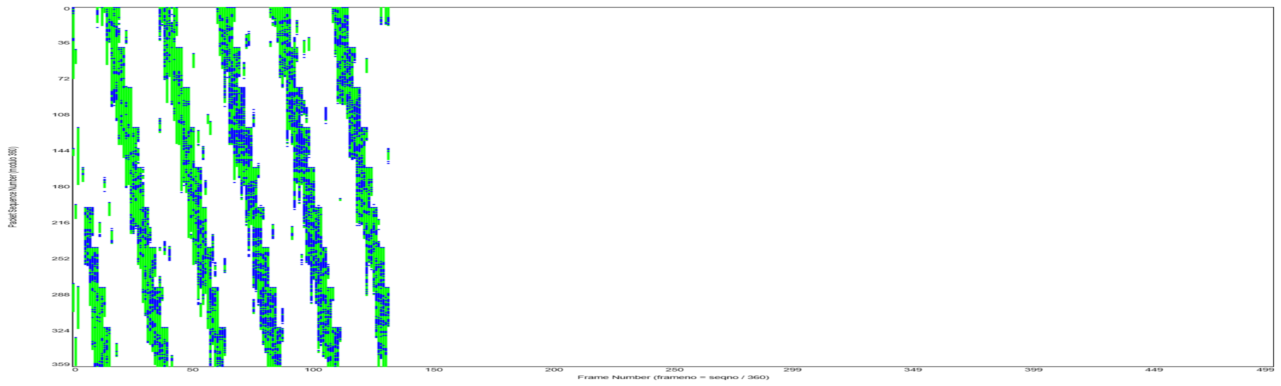
There were a lot more missing packets when doubling the number of packets in each burst. The throughput only increased to 455 KB/s even though we doubled the packets because there were a lot more losses.

Third time running burst1 I used N=25 and T=200 ms.



There were no missing packets at all for this time around, and the throughput was 176 KB/s. The throughput when N=25 is about half of the throughput when N=50, and I'm assuming it's not exactly half because when N=50 there is packet loss.

With N=40 and T=200 ms the throughput is about 276 KB/s and there is almost no packet loss.
With N=40 and T=400 ms the throughput is about 139 KB/s and there is a similar amount of packets lost compared to when T=200ms.
With N=40 and T=100 ms the throughput is about 542 KB/s and there is a similar amount of packets lost compared to T=200 ms and T=400 ms.
Woah. With N=40 and T=50 ms the throughput is 1.06 MB/s, and there is still barely any packet loss. I assumed that increasing the burst size and decreasing the timeout size would have equally overwhelmed the network, but it seems like the network can handle a lot of small packets better than some big packets. I guess it makes sense with burst1, because the timeout window is from the end of one burst to the beginning of the next, and since we're not getting any acks we can basically just send continuously. I just tried T=25 ms and it's still just getting faster with barely any packet losses. I brought the T down to 6 ms and it finally seems to be breaking. It works fine for a few seconds, then the network finally starts to get overwhelmed and drops nearly all the packets.
Onto burst2:
For burst2 with N=40 and T=200 ms, the throughput is also 276 KB/s, the same as burst1, and there's also not much packet loss.
For N=100 and T=200 ms, the throughput is about 446 KB/s, which is about the same result as for burst1, and there is similar packet loss to burst1 with N=100 as well.
For N=25 and T=200 ms it is again the same results as for burst1.
For N=40 and T=400 ms the results are basically the same as burst1.
For N=40 and T=100 ms the results are basically the same as burst1.
For N=40 and T=50 ms the results are basically the same as burst1.
For N=40 and T=25 ms seems the same as burst1. I guess this means that 40 packets can be sent in less than 25 ms, so there isn't too much congestion yet.
I ran it at T=12.5 ms and everything was the same as burst1 until right at the end of the video it just dropped a bunch of packets. I'll check if burst 1 does the same thing with N=40 and T=12.5ms. Woah. Now when I run burst1 at T=25 ms it's dropping almost every packet, even though before it was fine. It looks like:



I have no clue why there are large diagonals of lost packets to be honest.

Onto pipelined:
I first tried N=40 and T=200 ms, but every single packet got resent. I think this is because I set the timeout too long, so the acks aren't given enough time to make it back and the packets are re-sent even though they weren't lost. The throughput ended up being 163 KB/s because of this, which is way slower than burst1 and burst2 with the N=40 and T=200 ms.
N=40 and T=300 ms, the first frame or two worked really well, but then on the third or fourth frame every packet started getting duplicated again. I tried T=400 ms and I think I see what's happening. Because this protocol ignores out of order acks, that causes packets to timeout even though they shouldn't, and when a packet times out this protocol resends the entire window. I think this means making a larger timeout window will be better for performance.
N=40 and T=1 sec, it still seems to break when there's out of order packets/acks.
I tried changing N to 20 with T=1 sec, and it seems maybe a little bit better but it still resends way too often. I also just tried changing N to 100 with T=1 sec because why not, and it was suuuppper slow, and was still resending packets way too often.

<u>Part 6:</u>
My protocol is similar to the given pipelined one, but I added cumulative acks and a dynamic window size. There's one weird quirk that after running the client once, you have to restart the server to get it to work again after, and I have no clue why or how to fix it. The inputs are the destination ip and port number, and a timeout.
When running the server on a host in US central and the client in Asia east, and a timeout of 1 second, I was getting around 1 MB/s of throughput for the first 2 minutes or so, but at last 30 seconds it started to slow down a bit to 894 KB/s. My protocol gets bogged down a bit when it has to retransmit, and I'm not entirely sure how to fix it. I'm honestly surprised at this result because normally when I run it, it never gets above 300 KB/s of throughput for the majority of its run time. This specific example puts my protocol on par with the best burst1 and burst2 examples I had, and way better than stop and wait. I ran it a few more times with T = 1 sec, and I kept getting the same results, so I guess the network was just busier the other day so it was dropping more packets, causing my throughput to decrease because of retransmissions. I lowered the timeout to 750 ms, and the throughput dropped to 169 KB/s. I ran it with a timeout of 1.5 seconds, and it was having similar performance to 1 sec until there were some retransmissions, and then the performance dropped significantly down to about 477 KB/s.
I ran the server on us east and the client also on us east with a timeout of 1 sec, and even though the client and server were within the same region, because these servers drop a lot of packets the throughput was only 84 KB/s, proving that my protocol does not handle packet losses well. I'm gonna lower the timeout because I just pinged the

server, and the RTT was 1 ms. That worked way better, I did a timeout of 200 ms and the throughput was 380 KB/s. I take back what I said about not handling retransmissions well, I dropped the timeout to 50 ms and I got my highest throughput yet of 1.38 MB/s. My protocol seems to work pretty well as long as I get the timeout window correct. This is so cool, I just got a throughput of 3.78 MB/s with a timeout of 10 ms. I'm gonna try it from the asia server to this na east server that drops a lot of packets and see what the best throughput I can get is. With a timeout of .9 seconds, I got a throughput of about 600 KB/s, so I guess the combination of the server and client being far and the loss of packets is where my protocol lags behind.