Organization and Language

I used Python3. It's organized into Sender.py, STPSegment.py, and Receiver.py.

Program Design: SENDER

Sender establishes connection, sends data, and terminates connection. It does this with 3 threads - one to handle ACKs, one to send data, and one to check if transmission is complete. It logs events as they happen and logs the requested statistics once all threads finish.

Connection Establishment

Sender sends a SYN to the receiver and waits for ACK. If it times out then it retransmits. After retransmitting 3 times it will send a RESET and give up. If it receives the ACK, the sender updates the sequence number and window base it tracks internally.

Sending Data

Sender reads files in chunks and makes segments to fill the temporary window, then it transmits all the segments in the window. Retransmission is handled by the ACK handler because otherwise it gets really funky and gets really stubborn and keeps resending untransmitted segments when there's no ACK received and doesn't move on. After sending data, it adds the segment to the time tracking window.

Connection Terminate

I had to use threads here because the ACK handler was doing retransmissions and it had to know if the main method was done sending all of the original segments. In hindsight, I could've maybe simplified it? I'm not sure but I for sure should not have started the assignment this late.

Program Design: RECEIVER

The receiver listens for incoming STP segments, initializing the timer upon receiving the first SYN segment. Once again this uses the "hey start timer started" thing but only ONCE and I know it's like wow what a waste of memory :(.

The receiver handles SYN, FIN, and RESET segments by sending the appropriate ACKs if the loss probabilities allow. For incoming DATA segments, the receiver stores the segment in the buffer if it matches the expected sequence number, and sends an ACK for the expected sequence number.

If a received DATA segment has a sequence number less than the expected value, the receiver sends an ACK for the expected sequence number. It doesn't store this in the buffer.

If there are no incoming packets for 2 seconds, the receiver assumes the sender has terminated the connection and stops listening for data.

Data Structure Design

There is no design here. Just adding on more and more stuff as I needed them. Oh, I need to somehow make these two dumb threads realize that the fin-ack was already received? Just add an event. I don't even know if ack_timeout is an okay way to do things I just x_x please.

Design Trade-Offs

I don't know, I just did the basic stop-and-wait and just kept adding the required functionalities, then it would break and I'd debug and add or remove pieces and kept going until the code was done. I honestly probably can't tell you how it all works together x_x. Like, the individual lines I added when I was debugging? Easy. How do all these random lines I added to hot-fix different bugs as they popped up while I was under a caffeine-fueled and panic-driven code fugue work together? I really can't.

Things that don't work as it should

There is a strange bug where the sender program will not end after complete-ing transmission - I can't narrow it down because it occurs maybe once every 30 or 40 runs and I don't know WHY. I can only guess it's something to do with the random ISN but I tested the edge cases and it seems to run when I try so I'm not sure.

It also (very rarely) just sends a FIN right after receiving an ACK for the SYN. It has happened TWICE in all the (probably) thousand(s) of times I ran this code while debugging and working on the logs. The first time it did this I thought I was finally passing out, but when it did it a second time I tried to fix it. I THINK it's because for some reason SOMETIMES it's like ??? no data here to send, keep going boss :). Which I hate. So I added a "please send at least once even if it's empty I'm begging" condition to send_data. Hopefully this resolves that but if it happens when you test the code please please please pretend it didn't happen.