Jerry Gu
Computer Networks
RELIABILITY ASSUMED

Prologue

For this assignment, I read over the assignment and then didn't work on it for a couple of days. Then I read it again, and then did my first handshake attempt. Then I procrastinated for about 3 weeks and then I realized what I wrote was wrong. Did I get your attention? Do you guys even read these? Well either way I'll make it more entertaining for you because God knows Joshua writes way too much. This doesn't even count for credit, so why am I even putting this much effort in? I guess if I put more effort in, but I have a more fun time doing it, it's better than putting in a little bit of annoyed effort.

Chapter 1: Handshakes are hard in real life and in coding.

This will now start from that point of realizing my handshake was written incorrectly. The initial handshake was actually easy enough, but it was confusing because the initial state of being closed is sort of implied by just opening and connecting in itself. So, it's a bit strange to think about because the state is real, but I didn't really code it explicitly in the code until the end where I set the state to done. But if the context is done, then that's really the "closed state." Either way, it was weird to wrap my head around.

Then came the FIN handshake. This was extraordinarily more confusing because at some point we had to also begin writing control loop. The handshake initially is done before the control loop, but because control loop is called before FIN, FIN's implementation relied on it. Also, because we didn't need to implement the TIME WAIT state, it was strange to implement the active closing. After consulting with the resident German Internet Man, we agreed that it made the most sense to have CLOSING and FIN_WAIT_2 receiving an ACK or FIN respectively lead to the same result: the closing of the connection. We just pretended that there would be a TIME_WAIT state but since that didn't exist, it just went straight to wherever TIME_WAIT went, which was closing. I have left a comment in the teardown function where this occurs.

Chapter 2: Control Freaks

Now has come time to implement the control loop. Honestly conceptually speaking, this assignment was incredibly confusing. I felt like reading the assignment was far too much information. If you must have a "what do I actually need to do?" section on the assignment description, I feel like that's indicative that the prerequisite knowledge might be a little bit overwhelming. Then again, I think I feel like this on every assignment, even if at the end of the day it was quite simple. Anyways, implementing the actual state machines would have been quite simple had I been less confused on what was actually happening between the network and the app. I was confused for the longest between what should be happening with the app and the network. But eventually after bothering the German Internet Man (GIM), he clarified what was going on conceptually between the APP and the NETWORK with the app/network sending/receiving.

Jerry Gu
Computer Networks
RELIABILITY ASSUMED

Chapter 3: My sanity has sent the FIN packet to the heavens of Campus Wire, and Professor Park has ACK'd it back to the Earth.

As I focused mostly on the skeletal implementation of making the sending of the packets work, now all that was left was to ensure the requirements of the sliding windows, ACK packets verification, and the in-flight packets. Luckily, I had set up my context well beforehand so managing the windows was largely built in due to the abstraction of ACK packets working out quite well for me.

When I had to send data, it was simple to just increment the window by the number of bytes sent. Thus, I was able to manage both of those quite well. I would have abstracted sending data and making those packets a bit more, but it was done so few times I didn't find it necessary. Sorry if it's a bit messy.

Either way, I was really confused about what was going on with the timers, reliability, out order, etc. I made some posts on Campus Wire asking about it, and thankfully the professor responded very quickly despite it being the weekend. Maybe he knows that we students will wait until the last minute, so he was ready on the weekend since most of us are probably working on it then. I know I was. Once I got that clarified away, it was trivial to only implement only sending 3072 in flight packets. I just checked whether the difference between the last ACKed packet and the base of the window was larger than the window size. If it was, then just wait. This was done by calling *continue* in the larger while loop which would then loop back through the state and check again.

I couldn't find any bugs, but I also don't think I checked certain aspects of the code as much as I could have. Testing this can be very nebulous because so much of the code is abstracted away, but from printing the numbers on my code, it seemed to be right. I'm unsure if you all even read the code anyways, since if we fail the test your credit gets absolutely blasted.

Final chapter: Are you even reading this part?

Honestly, I had a lot of fun writing this because I get so bored writing the normal READMEs. This is despite this README not even counting for any credit so putting this much effort into something that's not going to read is purely for my own amusement. Anyways, as I mentioned before, I collaborated with Joshua and Aaron just talking about the implementation and clarification of certain finer points that I mentioned earlier. It was a bit strange that despite there being so much information on the document, a lot of the assignment was unclear of what to do I found. Maybe it's just because I have a hard time paying attention in class. Well either way, this isn't going to be graded until I get back to America probably because God knows my patience hit the time_out waiting for the last assignments, but I hear it's just one TA grading all of these. That's pretty messed up, we have like at least 5 undergrad TAs and at least 1 grad TA grading, so it all gets done. Okay I'm not going to force you to read any more, hope you're having a good day!