We started our project by doing steps 1 and 4 first to test TCP on our own machine as well as to the server, and this ended up taking quite a bit of time due to differences in build environment as well as issues with the server, which we discussed with both Dr. Christensen and our TA Steven. Most of this was resolved through proper port selection and using the IP of the server rather than the standard URL, and then we could start the coding of UDP to act like TCP. On a single machine from console to console, the speed was around 8s for the 100MB and to the server it was around 30s or so.

For our protocol design, we checked on the internet through a few sources, the textbook, peers, as well as asked the Dr. Christensen for advice, and we ended up using Stop-and-Wait (SAW). We understood due to the nature of the protocol, it would be slower than Go-Back-N (GBN) and Selective Repeat (SR), which were the other two automatic repeat request (ARQ) protocols that use sliding window. All three protocols were discussed in the lecture, but we also went online to search for more information about them. From discussion with some other groups that started the project early, we learned that SR was far too difficult to implement properly and some of the groups had to start over with a different protocol due to that, which ended up costing time. For the group that did get some sort of SR working it also seemed that their speed was extremely slow, so we might as well just go with SAW to save time, then optimize the code to try to get it close to TCP speeds after the code was working Our professor also advised us to use SAW due to the simplicity of it, even after telling us to look at UDT and NETBLT first (these were even more complicated than SR).

Additionally, to make UDP reliable like TCP, there were a few things to consider and various ways to implement the protocol. Packet size could not exceed network maximum transmission unit (MTU), we should probably consider latency for speed, we could add sequence number to each packet, could use gap detection and gap fill to fix errors, and we can also use ACKs like TCP to make sure all frames are received. We were also not supposed to use any WSA socket function (Windows only) since the server would be running on a Linux distribution.

Some things to note is that SAW is better on a low link speed than high speed link [3], so it isn't too appropriate for this project since we are operating at over 50Mbps and transferring a 100MB file. Source three had some calculation but basically on a 10km link and low speed of 50kbps, the channel utilization would be 99.94%, while on a high speed 300Mbps link, channel utilization would only be 21.07% [3]. However, our priority in this project is first working code then to worry about speed. The main issue here is that each frame can only have a certain size, and we calculated this to be 1472 bytes as the Network MTU max (did pings to see when the packet would fragment). Additionally, even if the packet is large, having a higher loss rate would mean entire frames would need to be resent which take up even more time.

Here is a basically how the SAW protocol would work in our UDP transfer.

SAW Sender:

1. Open + bind socket
2. Open file for reading/sending
3. Send one frame
4. Wait for acknowledgement
    a. Start timeout clock
    b. If timeout clock expires, send frame again
5. Go back to 3 until file finishes sending

SAW Receiver:

1. Open + bind socket
2. Open file for writing
3. Receive frame
    a. Send ACK if frame is valid
4. Write data to file

Citations

1. https://en.wikipedia.org/wiki/Automatic_repeat_request
2. https://en.wikipedia.org/wiki/Stop-and-wait_ARQ
3. http://www.mathcs.emory.edu/~cheung/Courses/455/Syllabus/3-datalink/stop-and-wait-anal.html
4. http://www.pcvr.nl/tcpip/udp_user.htm