CS 396: "Mini" Project 3

Due Date: 11:59pm Monday March 29

For this "mini" project, we're going back to programming (this is where you cheer while my fragile Engin-ears are not listening). The goal for this project is to get y'all some experience with reliable communication over an unreliable link, *basically* TCP.

We will have code (on Moodle) to simulate an unreliable link over which you have to provide reliable communication. To keep things simple, the link has a buffer constrained to only allow two packets to be sent at a time (i.e., at most two packets can be unacknowledged). The link also has an arbitrary delay and loss rates. Your mission, should you choose to accept it, is to create and implement a protocol over this link that achieves reliable (i.e., correct) data transfer in a *reasonable* amount of time. As always, should you run into an issues reach out via Slack or office hours. Good luck!

1. **Mission Objectives**

   From the files available on Moodle, **you should only make changes to file MP3.py.** The other files are in charge of simulating the unreliable connection or helping you test out your solution. You will only need submit your version of MP3.py, so making extra changes to any other file to get your reliable communications to work wont be in your favor.

   Modify the code in the MP3.py file so that they use a reliable transmission protocol. This means, the protocol you implement must ensure that data sent via the "send" function can be reliably and quickly reconstructed by the `recv` function. At your disposal are timeouts and socket reads (e.g., setting `socket.timeout(float)`) and developing a protocol through which each side can exchange acknowledgements.

   Emphasizing one thing that might lead to hiccups: The unreliable link will only handle two packets at a time, sending a third packet while there's already two packets in transit will cause the third packet to just be dropped. Figure out your timeouts and acknowledgements!

2. **Test, test, and test again until lions become griffins!**

   For your benefit, using `tester.py` will run tests on your code. Note that `tester.py` uses `receiver.py`, `sender.py` and `server.py` to do the simulation for our unreliable link and to test your solution. There's a few parameters to test your solution under various conditions and to (hopefully) receive varying amounts of debugging info to better grok the network. Calling `tester.py --help` will give you info on these parameters and options.

   E.g., To run a link with a 5% loss rate and a delay of 100ms you would run the command

   ```
   $ python3 tester.py --file test_data.txt --loss .05 --delay 0.1
   ```

   in your CLI. **Strongly** recommend trying out your solution with varying loss rates and delays!

3. **Extra**

   Yes, there's programming but the goal of this is to get you thinking about timeout timers, ewxponentially weighted moving average techniques, and all the fun more "implementation" parts of reliable transport. Having a correctly setup timeout will reduce the round trip time and pump up the throughput. Recall from the slides, a "good" way of determining what timetout to use is via the "EstimatedRTT + (DeviationRTT * 4)".

   There's a `--verbose` option, use it! It'll drop a lot of info on what's going on.

   Using the `--receive` option will show you results of the file transfer, since the default is for the testing script to store results to a temporary location.

   Your packets should be smaller or equal to `miniproject3.MAX_PACKET` (1400 bytes).

   Make sure both sides of the connection finish without user interference (check out the FIN/FINACK/ACK sequence in TCP for inspiration).

4. **Extra extra!**

   For those of y'all wanting to flex your non-Python muscles, I won't limit you but it's up to you to port the code to whatever language you want to flex on. I will grade you on the port PLUS the protocol. To make things "fair", I'll grant a four day extension (i.e., Friday April 2 11:59 PM) if you plan on doing a port to a different language.