CSCI 321 – Assignment 4
Winter 2021

Instructions

This assignment is to be completed individually and is due on **Friday, April 2, at 11:59 pm**. All submissions should be made on Canvas. For this assignment put all your answers in a folder, create a zip archive and name it as follows: **assignment_4_<W&L_ID>** (e.g. assignment_4_17xxxxxx).

The Assignment

In this assignment you have been provided with code that simulates an unreliable link between sender and receiver. This link has a very constrained buffer (only two packets can be 'in flight' at a time), and can have arbitrary delay and loss rates. Your job will be to create and implement a protocol over this connection that correctly transfers data, in a reasonable amount of time.

Implementing Your Solution

This assignment contains several tools that will help you simulate and test your solution. You should not make changes to any file other than **`source.py`**. All other files contain code used to either simulate the unreliable connection, or code to help you test your solution.

Your solution file (**`source.py`**) will be tested against stock versions of all the other files in the folder, so any changes you make will not be present at grading time.

Your solution must be contained in the **`send`** and **`recv`** functions in **`source.py`**. You should not change the signatures of these functions, only their bodies. Your solution must be general, and should work for any file for testing purposes.

Your task is to modify the bodies of these functions so that they communicate using a protocol that ensures that the data sent by the send function can be reliably and quickly reconstructed by the **`recv`** function. You should do so through a combination of setting timeouts on socket reads (e.x. **`socket.timeout(float))`** and developing a system through which each side can acknowledge if or when they receive a packet.

Remember that the connection is bandwidth constrained. No more than two packets can be "on the wire" at a time. If you send a third packet while there are already two packets traveling to their destination (in either direction), the third packet will be dropped, so it is important that you get your timeouts and your acknowledgments right.

Testing Your Solution

You can use the provided **`tester.py`** script when testing your solution. This script uses the **`receiver.py`**, **`sender.py`**, and **`server.py`** scripts to simulate an unreliable connection, and to test your solution.

The **`tester.py`** script contains many parameters you can use to test your solution under different conditions, and to receive different amounts of debugging information to better understand the network. These parameters and options can be viewed by calling **`tester.py --help`** and are also reproduced below.

```
usage: tester.py [-h] [-p PORT] [-l LOSS] [-d DELAY] [-b BUFFER] -f FILE
                 [-r RECEIVE] [-s] [-v]

optional arguments:
-h, --help          show this help message and exit
-p PORT, --port PORT  The port to simulate the lossy wire on (defaults to 9999).
-l LOSS, --loss LOSS  The percentage of packets to drop.
-d DELAY, --delay DELAY The number of seconds, as a float, to wait before forwarding a packet on.
-b BUFFER, --buffer BUFFER The size of the buffer to simulate.
-f FILE, --file FILE The file to send over the wire.
-r RECEIVE, --receive RECEIVE The path to write the received file to. Results are written to a temp file otherwise
-s, --summary       Print a one line summary instead of the verbose mode output.
-v, --verbose       Enable extra verbose mode.
```

For example, to see how your solution performs when transmitting a text file, with a 5% loss rate, and with a latency of 100ms, you could use the following:

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

Additional Hints

- A key part of this assignment is determining how long to wait before resending a packet. You should estimate this timeout value using the weighted moving average (discussed in class) technique for estimating the RTT, and use this in determining your timeout. With correctly tuned timeouts, lower RTT will result in higher throughput.

- A good way of determining the timeout to use is the "**estimated RTT + (deviation of RTT * 4)**". You should check with your book for more details.

- Use the included **`--verbose`** option to include very detailed information about what your code is sending over the network, and how the network is handling that data.

- Use the included **`--receive`** option to see the results of your file transfer. By default, the testing script will store the results of your code to a temporary location. This option may be useful if you're not sure how or why the received file does not match the sent file.

- Make sure you try your solution under many different loss ratios and latencies by changing the parameters in the tester.py script.

- Keep your packets smaller than or equal to **`assignment4.MAX_PACKET`** (1400 bytes).

- Pay attention to the end of the connection. Ensure that both sides of the connection finish without user assistance, even if packet losses occur, while guaranteeing that the entire file is transferred. Look at the FIN/FINACK/ACK sequence in TCP for ideas.