FAQ Assignment 2

Implementation:

Q. Any restriction on Programming Language?
A. As long as the code compiles/runs on the linux.student.cs environment, there is no restriction.

Q. Can we reuse some code from A1?
A. Yes

Q. Do we need multi-threading?
A. For the sender, multi-threading is optional.

Q. Can we modify packet.py or the network emulator?
A. No, you should not modify either packet.py or the network emulator unless for testing purposes. We will evaluate your submission using unmodified versions of packet.py and the network emulator.
If you're implementing your assignment in Python, you can make use of the Packet class in packet.py. If you want to use a Packet class with different functionality, you must provide a custom Packet class implementation in a file with a name other than packet.py.

Q. Do we have to worry about files being too large to fit in memory
A. No need to worry about this.

Q. Can the EOT packet get lost?
A. You can assume that the EOT will never be dropped/lost.

Q. Do we need to handle binary files?
A. Not required. Handling only ASCII files is fine.

Q. Why do we use UDP rather than TCP?
A. In A2 you are implementing an RDT to offer reliability on top of an unreliable channel. If you use TCP you are establishing a reliable channel; when packets get dropped by the nEmulator, TCP will retransmit the packets, which defeats the purpose of the assignment.

Q. What exactly are we implementing?

A. What you are implementing here is a pipelined Reliable Data Transfer protocol on top of UDP where:

- ACK(n) signals that the packet with sequence number n (and only that packet) has been received.
- The receiver must buffer out-of-order packets.
- The packets are pipelined in batches of 10 packets.

Testing:

Q. How to test with 0 delay?
A. To test with 0 delay, connect sender and receiver directly without the network emulator in-between.

Q. How to test with FIFO order of packets with the network emulator?
A. The provided network emulator implementation cannot ensure FIFO order of packets. You can expect it to shuffle incoming packets even with delay set to 0. To test with FIFO packet order, you can connect sender and receiver directly.

Q. How will the code be tested?
A. Your code will be tested by setting various levels of delay and loss (e.g., a very lossy super-fast channel, a very slow reliable channel *etc*.) with files of various sizes to test your implementation under different scenarios.

Q. How long should the timer be?
A. Choose an appropriate value (not too small to cause lots of packet re-transmissions and not too long to wait forever for ACKs). A reasonable value would be between 50ms - 200ms, but we may test with a few extreme values (e.g., 1 ms, 1 sec). Be sure to experiment.

Q. How long should the network emulator's timeout be?
A. Choose an appropriate value (not too small that the delay is negligible and not too long that the packets are, in effect, dropped). A reasonable value would be between 30 - 100 ms, but we may test with a few extreme values (e.g., 1 ms, 1 sec). Be sure to experiment.

Q. How to find open port numbers?
A. The following command will return 3 random open ports between 1024 and 65535.

```
comm -23 <(seq 1024 65535 | sort) <(ss -tan | awk '{print $4}' | cut -d':' -f2 | grep "[0-9]\{1,5\}" | sort -u) | shuf | head -n 3
```

Q. What's the format of the packet discard probability?
A. A float in the range of 0 to 1

A few notes:

General:

- If there is a port collision while running your programs, that is because other people are running theirs as well. Just pick a different port number.
- You can use any library that is already installed in the student.cs.linux environment
- You can assume that the input file is ASCII (not bytes)

Sender:

- All data packets except for potentially the last data packet must be 500 characters long.

- If using multiple threads to implement your sender, you will likely need to use locks.
- The data sent over a socket needs to be a byte object. In A1, you convert strings to bytes by using encode(). Similarly, for A2 you need to find a way to convert packet information to bytes. (Hint: check out packet.py.)

Receiver:

- You can assume no corrupt packets.
- The receiver should not discard packets that have never been received. Make sure that the buffer is big enough.
- As soon as the expected packet is received, both data from the expected packet and consecutive packets that are buffered should be written to file immediately and removed from the buffer.