Assignment 2 - FAQs

Q. How to test with 0 delay?
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?
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 to generate specific scenarios to test the implementation/logging?
You are encouraged to modify the network emulator code to generate different testing scenarios. For example, to test timeout at a specific window size, you can modify line 51 or line 107 to deterministically drop a packet with a specific seqnum.

Q. Submission format on Learn
Zip all of your scripts into a single zip file. The name of the zip-file should be your quest username.

Q. How long should the timer be?
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

Q. Do we need multi-threading?
For the sender, multi-threading is recommended but not required.

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

Q. Can the EOT packet get lost?
EOT will not be dropped by the network emulator.

Q. Can we reuse some code from A1?
Yes

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

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

Q. How will the code be tested?
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*. to create different scenarios.

Q. How to find open port numbers?
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. Does the sender restart the timer when receiving a duplicate ACK?
If 3 duplicate ACKs are received for a packet while the next packet is not yet ACKed, this next packet is retransmitted and the timer is restarted.

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

Q. Why do we use UDP rather than TCP?
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 network emulator, TCP will retransmit the packets, which defeats the purpose of the assignment.

Q. What exactly are we implementing?
What you are implementing here is a congestion-controlled RDT that is a simplified version of the congestion control mechanism TCP Tahoe, where

- ACKs are cumulative
- The window increases by 1 packet every new ACK (new and not a duplicate or old ACK); that is the way the window increases in slow-start mode.
- There is only one timer and it is associated with the oldest to-be-ACKed packet
- In the event of a timeout the sender resets N to 1 and only retransmits the packet associated with the timer
- The receiver buffers out-of-order packets
- In order to evaluate your program and see if you implemented the algorithm correctly, you would set the initial window to N=1

Q. What if due to a significant delay, the sender/receiver receives an old "Ack" whose seqnum is the same as the one expected?
No need to worry about this. We will use appropriate parameters to avoid such scenarios.

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)

Timestamp and Logs:

- Count sequence number as packet counts, not bytes

- Log values of N in edge-case as well. For example, if it is already 1 and gets reset to 1, log it again.
- We only increment t after each *event* (*new* packet sent, receiving an ack, and on timeout) takes place, but N changing isn't itself an event. Duplicate ACK packets increase timestamp. Timeout+Resend packet is a **single** event and has the same timestamp. (Don't increment timestamp twice here)
- Log the initial value of N so that N.log is never empty even if no timeout occurred.
- Receiver logs *all* received sequence numbers.

Sender:

- All data packets except for potentially the last data packet must be 500 characters long.
- You may want to use locks with multi-threading in the sender program
- 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.
- A packet gets *retransmitted* if it is the first in the window (the oldest yet-to-be ACKed packet) and the timer times out or if 3 duplicate ACKs are received for the most recently sent and ACKed packet.
- There must be only one timer in the sender. This means when you receive an expected ACK, you restart that timer for the next in-flight packet. You do not keep separate timers for each in-flight packet.
- Discard ACKs for out-of-window packets even if they have a higher seqnum.

Receiver:

- You can assume no corrupt packets.
- If the first received packet is out-of-order then the ACK will have seqnum 31
- The receiver should not discard packets that are within the next 10 sequence numbers and can be buffered.
- 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