Jared Hynes (jlh484) and Mark Malysa (mbm206)

Implementation Summary: explain handshake, data transfer (with delay), and teardown logic.

- Our RUDP uses a three-way handshake to set up connection between the client and the server. We use three message types in this, (1) is SYN, (2) is SYN-ACK, and (3) is ACK, to confirm that the connection is complete. The client will send a SYN packet to the server which will wait for a response of a SYN-ACK after the server has set the client address. Once the client receives a SYN-ACK, it will send an ACK back to the server to confirm.
- For the data transfer, it uses a stop-and-wait data transfer that will timeout and retry to confirm packets make it. The client will encode a message and split it into chunks of 200 bytes. It will send each one of these chunks as a (4) DATA packet, and waits to receive a matching (5) DATA-ACK from the server. During this time, the server is confirming that the sequence matches the expected order sequence. Once it does, it will send a DATA-ACK to the client. If the data is received out of order, it will re-ACK the last in-order sequence to allow the client to know what packet it is transmission wise. For us we add a random delay that simulates the network being not perfect (since we are using localhost).
- For the teardown, after all data chunks are sent and ACKed, the client will send a (6) FIN packet and wait for a (7) FIN-ACK from the server. When the server receives a FIN, it will send back a FIN-ACK and reset its state (like client address) and print out a message that the connection was closed, while closing the socket. If the FIN-ACK is delayed past the retransmission time, the client will retransmit the FIN, but the connection will still be successfully closed once it receives the FIN-ACK once.

Capture Analysis: describe what each .pcap file shows and how retransmissions appear.

- Project3_handshake.pcap - The first packet is a SYN from the client that begins with the UDP payload (01 00 00 00 00 00 00, type 1), and the server sends a SYN-ACK (02 00 00 00 00 00 00, type 2), and finally the last packet in the handshake is the client sending an ACK (03 00 00 00 00 00 00, type 3).

- Project3_data.pcap - In this capture we can see packets alternating from client sending DATA (04 00 00 00 00 00 00, type 4) and DATA-ACK (05 00 00 00 00 00 00, type 5) from the server. We can see a retransmission here when there are multiple type 4 DATA packets being sent in a row, because the client is sending the packet again after not receiving the DATA-ACK in time before the retransmission time is up. Another telling factor in Wireshark is when the receiver has the same address and the counter is the same number, it is the same packet.

- Project3_teardown.pcap - In this capture we can see the teardown begin, with a type 6 FIN from the client, and then we receive a FIN-ACK from the server. In our specific capture, you can see a type 6 FIN from the client right around the same time, as this happens because the FIN-ACK was received in the middle of the client sending the

retransmitted FIN. When we increased the RTO to 1 second, there was only one FIN and FIN-ACK, however, to mimic retransmission during the data phase we keep the RTO at 0.5 seconds.

Discussion: why ACK delay triggers retransmission and how your RUDP ensures reliable delivery. Only .pcap captures are accepted; screenshots are not required.

- The server's random ACK delay will cause retransmission, because it is between 100-1000 milliseconds, and our retransmission timeout is 0.5 seconds. This means that on average about 50% of our packets will need to be retransmitted because it will take more than 0.50 seconds during our random delay. Our RUDP ensures reliable delivery by the server sending back an ACK when we receive a packet out of order, which tells the client which packet the server has most recently successfully read, and allows the client to understand where the issue is with the packets. Also, by using stop-and-wait, the client should only be able to send packets after it has received a sequence specific ACK, so out-of-order delivery will not continue in the event that it has happened.