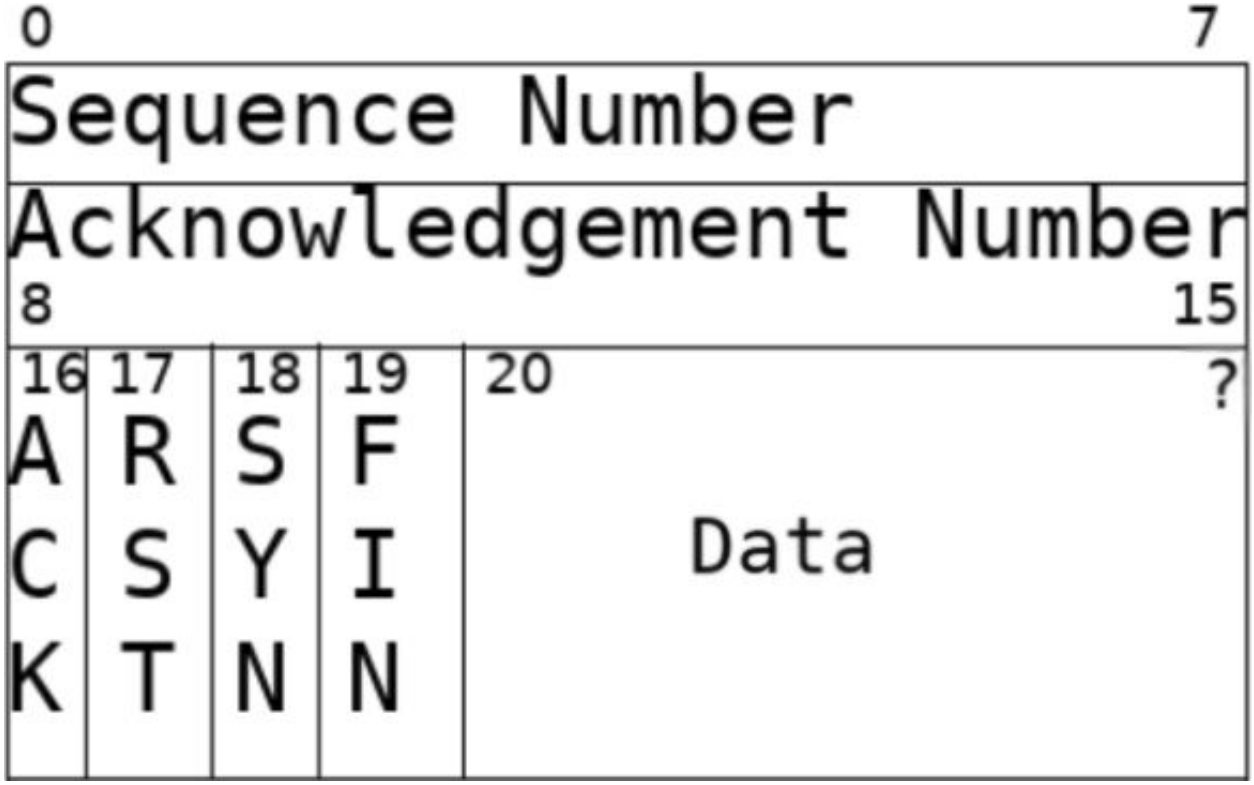
CPSC 3780 - Project Report

By:
Matthew Wilbern
Ugonna Osuji
Justin Onoferychuk
Amanda Munden

Due: December 7, 2019

Header:

Our header includes several key components. The sn is the sequence number and the sequence number is arguably the packet identification number. The acknowledgement number sends the previous sequence number which was received. Arsf is the packet flag. The packet flag determines the type of the packet. The rest of the packet is the data sent. The packets data size is determined by the protocol class. This is the overview of the header.



Protocols:

We initialize our connection with a three way handshake. The client starts by sending a SYN packet to server. The server then sends back a packet that contains an ACK and SYN. Client sends ACK back resulting in a connection established.

After the connection is established, we implemented a sliding window protocol for the rest of the packets sent from the client to the server. We used multithreading to send packets and to listen for incoming acknowledgement packets. We used a window size of 8.

Finally, we implemented a proxy server that relayed the packets being sent from the client to the server and the acknowledgements from the server back to the client. The proxy was set up with an error chance variable that could be changed to test different amounts of packets being randomly dropped and different amounts of packets being randomly delayed. We used a

counter to see how many packets were dropped and how many packets were delayed. We also used a timer to see how long it took to send all the packets.

Testing Procedure:

We ran multiple tests in which we sent 655 packets through our proxy. The information we were generating was the number of packets sent total, the number of packets dropped, the number of packets delayed, the number of packets that had to be resent and the time it took for all of this to occur. We generated a base case in which the proxy was set to not drop or delay any packets.

For the first test, we set the error chance variable to 15 which would result in a 2/16 (as the random number generator runs from 0 - 15, giving 16 numbers in total) chance that the packet is either delayed or dropped (1 in 16 chance of delay and 1 in 16 chance of drop).

For the second test, we set the error chance variable to 9 so that it would be a 1 in 10 chance of dropping a packet and a 1 in 10 chance of delaying a packet.

For the third test, we set the error chance variable to 99 so that it would be a 1 in 100 chance of dropping a packet and a 1 in 100 chance of delaying a packet.

Results of Testing:

Our base case was 0 errored packets. 655 packets sent in total, and it took 28.88 seconds.

After running the first test, we found that 161 packets were delayed, 133 packets were dropped and 294 packets were resent. It took 36.39 seconds. The batch was originally 655 packets, so with 294 packets resent that means we sent 45% more packets than our base case.

After running the second test, we found that 324 packets were delayed, 338 packets were dropped and 662 packets were resent. It took 42.37 seconds. The batch was originally 655 packets, so with 662 packets resent that means we sent 101% more packets than our base case.

After running the third test, we found that 10 packets were delayed, 13 packets were dropped, and 23 packets were resent. It took 29.35 seconds. The batch was originally 655 packets, so with 23 packets resent that means we sent 4% more packets than our base case.

Problems:

Every 255th packet, a single packet of data is lost. We know the cause for this but did not have the time to fix it unfortunately. Another error we encountered was when a reset ACK

packet was lost the client seems to loop while attempting to send a reset packet. If we drop the very first packet our program fails.

How to use our programs:

1. Run the proxy with the following arguments, in order:
    a. Client (sender) IP address
    b. Server (receiver) IP address
2. Run the server with no arguments
3. Run the client with the following argument:
    a. Proxy IP address

    Note that the client sends data that is inputted via stdin, so you can pipe commands/files into it.

Link to our Github repo for the project files:

https://github.com/fatboychummy/Lets-Talk