Web Implementing Window-based Reliable Data Transfer Protocol

Group:

- Frederick Kennedy (404667930) Discussion 1D
- Johnathan Estacio (404491851) Discussion 1B

Implementation

Client

To implement a reliable data transfer protocol on the client side, we implemented reliable data transfer principles on top of UDP sockets.

1. The client creates a UDP socket.
2. The client builds the server's internet address.
3. We then initiate the SYN/SYN-ACK handshake to the server, to establish connection.
4. We then wait for a response from the server. We created a timeout interval to keep a finite waiting time on packets.
5. From there, we wait for a response from the server through the socket similarly.
6. After that, we begin file transmission. We send packets of 1024 bytes until the file has been fully transferred. To know that the transmission was successful, we look at the ACK sent in the packet from the server. If it's what we expected, then we're good and respond with the next ACK accordingly. Otherwise, we simply don't write to file and don't send the ACK and let the server retransmit that packet.
7. When the file is done transmitting, the server will send a FIN. Upon this, we will complete transmission and begin closing the connection.
8. We exchange FINs and ACKs between the server and client to close the connection. We then close the file and close the socket, ending our connection with the server.

Server

To implement a reliable data transfer protocol on the server side, we implemented reliable data transfer principles on top of UDP sockets.

1. The server creates a UDP socket
2. This sets the sock opt value to see if the port number is already used
3. Gets the server address
4. Binds socket
5. We then create a while(1) loop, where the server will continually wait for a packet to receive from the client.
    a. The receive from function is within a while loop while SYN is = 0. Once the client establishes connection to the server, the SYN is set to 1.
    b. We then proceed to send a SYN-ACK.
        i. When sending this SYN-ACK, we created a timeout interval to keep a finite waiting time on packets sending the packets while the client receives it.
        ii. After waiting a certain amount of time, if the SYN-ACK isn't sent, we send a retransmission packet.
    c. After this, we receive a request packet from the client
        i. This request packet contains the filename of the file we want to open.
        ii. After opening the file, we begin transmitting data.
    d. The next set of instructions fall inside a while loop when FIN is still 0.
        i. We created a while loop when the offset is still less than the filesize and we've assigned 5 packets, more than that and the program will continue.
            1. We get the offset of the file which is essentially the starting point of the locations of the data.
            2. We send a packet of size payload which is essentially the data we want to transfer.
                a. Payload size is the size of the packet subtracted by header size.
            3. We store all the information after sending this to a packet array (where it has its own custom structure)
                a. location: starting location of data
                b. location_next: next starting location of data
                c. SEQ: sequence number
                d. ACK: ACK number

        e. ACKed: if the packet is ACKed or not
        f. size: size of packet
    ii. We then get the longest time to transfer the packets.
        1. If the packet with the longest time to transfer is greater than the timeout, then it will have to retransmit
        2. If the packet is somehow lost while waiting for client to respond, then it will have to retransmit
    iii. However, if there's no timeout necessary, then we will proceed. If however there was a retransmit from the client side, then we will detect it.
    iv. After this, we will check if the packet that was sent to the client was a FIN packet
        1. The client will respond appropriately telling the server it was a FIN packet
        2. We then begin with the FINACK handshake
        3. Eventually, this FIN packet will tell us that the connection between the client and server is over.
    v. If it's not a FIN packet, then we will continue the loop.
    vi. If the current packet that's being accessed at the moment is no longer ACKed, then we will update all the information to access the next packet (in the array)

Difficulties

1. Handling the packet loss and retransmit
2. Creating the structure of the packet, what it contains and the header size and the data size
3. Determining the order of packets when they are not properly ACKed in the right order
4. Timeout was not that difficult, we just used a select function