Alan Kha 904030522

CS 118 Project 2

UDP Data Transfer using Go-Back-N (GBN) Protocol

Setup

1. Run `$ make` to compile client.c and server.c.

2. Start server with `$ ./server portno win_size PLoss PCorrupt`.

3. Start client with `$ ./client hostname portno filename PLoss PCorrupt`.

Implementation

Packet Format

int type	Packet type: 0. Request message 1. Data message 2. ACK message 3. FIN message
int seq	Packet sequence number.
int size	Packet data size.
char data[]	Packet data with capacity PACKET_SIZE. Used to store request filename or file data.

Server

The server first listens for a file request message from a client. Assuming the file exists, the server determines the number of packets required, and then begins transmitting the file to the client in sequential packets using the GBN protocol.

The server will send up to `win_size` packets from the last acknowledged packet before timing out and retransmitting the window. When an expected ACK is received, the server slides its window over appropriately. This is done in a loop, first listening for 1 second if there are any new messages, and then sending the packet in the window.

Once all packets have been sent to the client and acknowledged, the server then sends and waits for a FIN signal before considering the transaction complete and readying itself for another file request.

Client

The client first sends a file request message to the server. Assuming the file exists, the client should then begin receiving sequential data packets from the server under the GBN protocol.

As expected data packets are received, the client appends the data to file and sends an ACK message back to the server. If the client receives a previously processed data packet, it will resend an ACK for that packet, while ignoring out-of-order packets or wrong packet types.

After receiving a FIN message with the proper sequence number from the server, the client will consider the file transfer complete, send a FIN message of its own to the server in acknowledgement, and exit.

Difficulties

The largest difficulty was implementing the listening timeout for the server, since the `recvfrom()` function used to read incoming packets was blocking by default, which prevented alternating between listening and sending. It took quite a bit of research before choosing and implementing the `select()` function correctly.

The only other significant difficulty was implementing correct error handling for lost and corrupted packets, as initial implementations sometimes resulted in a deadlock between the client and server. This problem was eventually solved through testing and tweaking.