Project 2: Simple Window-Based Reliable Data Transfer

Christopher Thai 804595489

Ning Hu 904751747

COM SCI 118 Spring 2018

Professor: Songwu Lu

Introduction

This project explored basic packet transmissions through UDP sockets. Due to UDP's unreliable nature, we had to implement features such as a window-based protocol and packet retransmission on timeout to ensure that all packets are received in the proper order. Additionally, to establish and terminate the connection between the server and the client, we implemented a three-way handshake and teardown. This lets the server and client know when to begin sending packets and when the file is finished being transferred.

Implementation Design

We divided this project into three sections—basic packet transmission, window-based protocol implementation, and timeout implementation. This allowed us to approach the project in incremental steps and divide the workload.

Packet transmission

To implement basic packet transmission between the client and server, the three-way handshake, and the connection tear-down, we modified the provided client and server template files from project 1. Due to UDP's irrelevant nature, portions of the template were changed to accommodate UDP.

When this portion of the assignment was first implemented, we assumed that there was a window size of 1 and that packets would always be transmitted in the proper order. We created a packet structure which would include all necessary values in our header which are the sequence number, the ACK number, the payload length, the type of packet, and the payload. More values were added in later sections. Unfortunately, a structure cannot be transferred over a socket, so to easily transition a pack to and from bytes, we also created a union which stored a packet and a character array of bytes that is the size of a packet.

After setting up the connection to the server on the client's side, we send a SYN packet for the three-way handshake. Once a SYN-ACK packet is received, the client sends an ACK packet with the name of the requested file. Later ACKs are sent to acknowledge the reception of payload packets whose payloads are then stored in received.data. Once a FIN packet is received, the client knows that the server is finished sending the file and sends a FIN-ACK packet. Upon receiving a final ACK from the server, the client closes the connection on its side.

Once the socket is set up on the server side, it blocks and waits for the first SYN packet from the server. Upon reception, it sends a SYN-ACK packet and waits for an acknowledgement from the client. The server knows that the first ACK packet received will contain the filename of the requested file which we assumed to be in the current directory. The server determines the number of packets needed to transfer the file and allocates memory for an array of pointers to packets. The data from the file are then stored in these packets' payloads and sent to the client one at a time. Once the last packet in the array is sent, the server begins the connection tear-down by sending a FIN packet to the client. After receiving a FIN-ACK packet and sending an ACK packet in response, the server closes its socket.

Window-based protocol

For the window-based protocol, we only added a window to the server.

The server sends all packets in the current window before waiting to receive any ACKs from the client. Upon receiving an ACK for a specific packet, the server moves the window ahead by one.

Timeout implementation

We added several fields to the packet structure to aid in this section of the project: received, sent, offset. If no connection is verified between the server and the client, the connection is dropped and must be restarted.

In the server, we used the select function to discover if there was timeout. Upon timeout, the server resends all packets in the current window.

In the client, if any packets are received out of order, the client drops that packet and resends a packet with the ACK number it is looking for.

Difficulties

A major issue we ran into while implementing basic packet transmission was the inaccurate transmission of data from the requested file. On the server side, it appeared as if the payloads were added properly to the packets, but on the client side, the same data was not being received. This inconsistency led to difficulties diagnosing the underlying problem. After reviewing the code, we realized that this dilemma was due to an incorrect assumption. In previous courses, we easily transitioned between structures and bytes by typecasting to pointers. However, the server and client processes do not share memory, so this technique failed to work. To fix this issue, we decided to create a union of a packet and an array of characters.

Project Manual

The following files must be in the same directory as the Makefile: server.c, client.c, helper.c, helper.h. The `make` command will compile the server and client programs. Each program must be run in a different terminal window, and the server must be run before the client. The following commands should be used to run each program:

```
./server [port]
./client [hostname] [port] [filename]
```

`[port]` should be the same for both files, and `[hostname]` should be `localhost` if both programs are being run on the same computer. `[filename]` should specify a valid file in the current directory.

Conclusion

Through this project, we learned about the difficulties of implementing reliable data transfer and its importance in sending files. It also reinforced the concepts we learned in class by allowing us to view the process firsthand; we were able to construct a TCP-like data transfer program that was connection oriented in order to implement reliable data transfer.