CPSC 3780

Project Report

Group Members:  
Kyle Akerstrom  
Jonathan Flores Avalos  
Mark Wagner

# **INSTRUCTIONS**

## **Scenario #1**

1. Open Terminal console
2. Navigate to Networking-Protocols/S1/server
3. Run the “make” command
4. Run “Server”
5. Open another Terminal console
6. Navigate to “Networking-Protocols/S1/client”
7. Run “make”
8. Run “Client”
9. At the prompt, enter “localhost” (or the IP of the Server)
10. The file should display on the client's screen

## **Scenario #2**

1. Open Terminal console
2. Navigate to Networking-Protocols/S1/server
3. Run the “make” command
4. Run “Server”
5. Open several other Terminal consoles
6. Navigate to “Networking-Protocols/S2/client”
7. On one console, run “make”
8. On all Client consoles, run “Client”
9. At the prompt for each Client, enter “localhost” (or the IP of the Server)
10. In each Client window, enter the name of the desired file, which will be found in Networking-Protocols/S1/server/docs. Example inputs are “lorem.txt” and “shunned\_house.txt”.

# **Design & Implementation**

## **Data Structures**

Before starting our project, we designed a suite of classes to help us with formatting, serializing, and parsing the information being transmitted. We've named these classes after networking terms, but although we attempted to more closely emulate their namesakes, we eventually decided to simplify them into the forms described below.

At the lowest level, we've created a “Frame” class, which does all the heavy lifting for data parsing. Each Frame contains fields for Sequence Number, Frame Type, Even-Parity Bit, and Data.

The Frame also contains methods for functionality such as calculating the correct Even-Parity Bit, checking if the Even-Parity Bit is valid for the given Data, easily creating an ACK/NAK frame for a given sequence number, as well as serializing and deserializing the Frame as a string to be sent and received on the network. The frame serialization format is as follows:

1. The Sequence Number, as a numeric character. (i.e., '0', rather than 0)
2. The Frame Type, as a numeric character.
3. The Even-Parity Bit, as '0' or '1'.
4. The Data

At the higher level, we created a Packet class. In our implementation, the Packet class acts essentially as a container for frames with some useful utility functions. Most notably is a function to parse a file into a vector of <=64 character data Frames.

## **Overcoming Challenges**

We encountered several issues while developing this project. Much of this could be attributed to our group not being very familiar with C++, with two of us having only worked with it this semester, and one of us having not used it until this project. Despite this, however, we were able to overcome most of our issues surprisingly without much trouble.

An interesting bug we faced was that the Client would receive data from the socket out of sequence, so that a Frame would include control bits in its data. This was before we implemented ACKs and NAKs, and we discovered it was an issue with the input buffer getting overloaded and reading too much data into the Frames. Once we implemented ACKs, the program worked perfectly.

Another interesting issue we faced was when we implemented randomly changing the parity bit. We forgot to make a copy of the frame to send, and instead changed the parity bit on the Frame as it was in the vector. As a result, the Server would change the parity of a Frame, send it, receive a NAK, and continue to send the incorrect Frame until it randomly changed again, at which point it would continue as normal. This was an easy fix, but we found it to be an interesting bug and a nice demonstration that our project works as intended even if the Server continually sends incorrect Frames.

An issue that we haven't overcome was that we weren't able to get a shared library to work, so we were forced to copy any common files to each folder manually. Fortunately, very few changes needed to be made to the common files, and it was quick to copy and paste, but it isn't an ideal solution.