COMP 3825

Term Project

Final Report

Alex Ziegenhorn

1 December 2017

Design Description

For this project, I decided to use Python. I decided to use Python because I have not had many opportunities to use it during my college career, and I especially have not been able to use it in a networking capacity. I figured that this would be an excellent way to learn about Python's networking capabilities.

The purpose of the project is to learn more about exactly how messages are exchanged between two entities on a network, and to specifically note how UDP can be coded to handle potential packet loss. This is what Alternating Bits Protocol and Selective Repeat Protocol help to detect.

I decided to split up the implementation of the Alternating Bits Protocol and the Selective Repeat Protocol into different server and client programs. So, in total I have four programs in the form of executable python files. They are as follows: AlternatingBitServer.py, SelectiveRepeatServer.py, AlternatingBitClient.py, and SelectiveRepeatClient.py. During the design portion of this report I will generally be speaking as if there are only two implementations, if not just to simplify things. I will make specific references to the four programs where applicable.

During initial execution of the client and server program, it does not matter which order they are turned on, nor does it matter how long one is on. However, the server does need to be turned on before a message is sent from the client.

When the server is turned on, it immediately begins listening for a message from an outside source. Upon receiving an initial message, it immediately parses it. After parsing, depending on the ACK/SEQ numbers and the message contained, it will do one of a few things:

- Close the connection
- Initiate handshake
- ACK and wait for message with Window Size specifications (Selective Repeat)
- ACK and initiate the file transfer process (Alternating Bit)

From here, there is some specific branching depending on whether we are dealing with the Selective Repeat server or the Alternating Bit server. I will go ahead and outline the process for both the server and the client, because 95% of the calculations and data manipulation takes place in the server program.

- Selective Repeat:
    - Server waits for message specifying window size.
    - Once received, server puts together its buffer details, which it calculates using the window size, and sends this buffer information to the client.
    - Upon receiving ACK from client regarding buffer details, server initiates the file transfer process by starting to send the first frame of packets.
    - Once each frame is finished being sent, server waits until it receives all of the ACKs associated with the frame, and then begins sending the next frame.
    - Once all frames have been sent, connection is closed.
- Alternating Bit:
    - Server initiates the file transfer process with a SEQ number of 0, sends first packet.
    - Client receives packet, sends ACK back with 1 as the value.
    - Server receives ACK, sends next packet with SEQ number of 0.
    - Repeat until file has been successfully transferred.

<u>Proposed vs Implemented Functions</u>

When I made my design document, I made a lot of mistakes. I had to pretty much overhaul everything mentioned to ensure that everything worked properly.

<u>Implementation</u>

Here I will outline the file structure of my project, as well as include brief descriptions for each of my functions within each python file.

<u>File Structure</u>

```
Core Project/
|       ClientSpace/
|       |       AlternatingBitClient.py
|       |       SelectiveRepeatClient.py
|       |       mobydick.txt – (the file I was transferring from server to client)
|       ServerSpace/
|       |       AlternatingBitServer.py
|       |       SelectiveRepeatServer.py
|       |       alice.txt -- (another file I was transferring)
|       |       mobydick.txt
|       testparser.py – (used for testing simple parsing code)
```

Function Outline

AlternatingBitClient:

parse(recv)

Parses received data into string form.

waitForOK(seq, filename, clientSocket, serverAddress)

Waits for ACK from server that it has received the filename request.

waitForData(seq, file, clientSocket, serverAddress)

Waits for new segments of the file from the server, and sends an ACK if a segment is correctly received.

main()

Sets the connection information, handles file, calls functions.

AlternatingBitServer:

parse(recv)

Parses received data into string form.

waitForFilename(serverSocket, mACK)

Waits upon program start for the client to send the filename request

sendTheFile(serverSocket, mACK, file, numberData)

Sends the file separated into segments in packet form, and also closes connection with client upon upload completion.

SelectiveRepeatClient:

parseBufferData(recv)

Specifically parses the information received from the server regarding buffers. Stores the updated totalPackets and totalFrames inside of their respective global variables.

parse(recv)

Parses received data into string form.

waitForServerWindowSizeRequest(filename, clientSocket, serverAddress)

Called from main, loops until a windowSize request has been received from the server.

waitForServerBufferDetails(clientSocket, serverAddress)

Called from main, and then loops until a segment containing the necessary buffer details arrives from the server.

waitForData(file, clientSocket, serverAddress)

Waits for the new segments of the file from the server, and sends an ACK if a segment is correctly received. Also regulates some of the variables needed for Selective Repeat.

SelectiveRepeatServer:

parseLazy(recv)

Specifically parses the ACK number from received packet.

parse(recv)

Parses received data into string form.

waitForFilename(serverSocket, mACK)

Waits upon program start for the client to send the filename request.

waitForWindowSize(file, serverSocket)

Waits for window size from client.

waitForAckOfBuffer(serverSocket)

Wait for ACK of buffer details from client.

SendFrame(serverSocket)

Sends the five packets that make up a single frame.

receiveAck(serverSocket)

Receives all 5 ACKs that are associated with the current frame.

Skills Learned

During this project, I was able to learn a lot about how Python and UDP work together to send/receive packets. I had to use the code posted on elearn initially, to get myself started out (Only during the Alternating Bits section). None of it actually ran on my computer though, so I had to rewrite it myself from the ground up. That in and of itself was actually a great tool for helping me to better understand the code. I have a much greater understanding of how the Alternating Bits and Selective Repeat protocols work as well. I am really impressed with how well python handles network traffic.

Overall, this project really built on my past experiences with socket communication in C++/C# that I had to learn during a research assistant position over the summer, and gave me a nice different perspective on the subject.