Project 4 Report

Samuel Bhushan

Introduction

The focus of this final project is the Transport Control Protocol (TCP). This protocol maintains a connection between two parties for data to be transferred. Before data can be sent, the connection has to be established via several messages. Once this is done, data can be transferred. The connection is closed after the termination stage is completed.

Our purpose in this project is to act as a server to a client, by writing a program that accepts connections. Once a connection is made, we store any bits that the client sends into a file. The client program has already been created for us, and sends an unknown message when a successful connection is made.

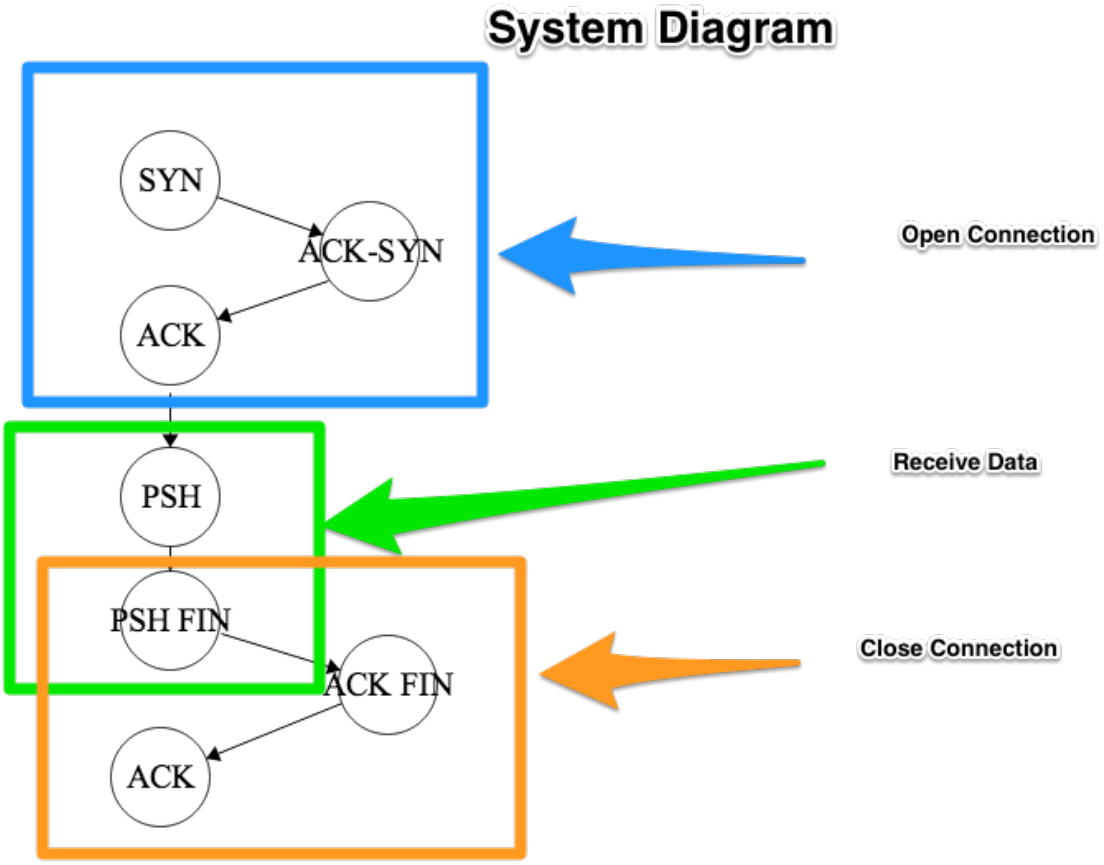
Program Code

To establish TCP connections in C++, the <sys/socket.h> library was used. The basic steps of the program are as follows:

1. Create a Socket with socket()
2. Set the address to look at requests coming to our computer on port 5600
    1. Since we are looking for any connection requests, the IP address used was a macro “INADDR_ANY” that corresponds to any network controllers attached to this computer.
3. Bind the address to the socket with bind()
4. Wait for a connection request with listen()
5. Accept the connection with accept()
    1. This sends the SYN and ACK messages that start a TCP connection.
6. Recieve the data
    1. We used a buffer size of 1 and kept reading until all bits had been received.
    2. The data was first sent to a file as characters
    3. Then the same data was printed to the console
7. Then the Socket was closed with close()
    1. This sends the termination messages.

```
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <errno.h>
#include <fstream>
using namespace std;

//
// default IP address (if not specified on the command line
//
#define IP_ADDR "192.168.1.20"



int main(int argc, char *argv[])
{
    //FILE
    fstream file;
    file.open("out_from_client.txt",fstream::out);
    //file<<"file opened"<<endl;

    int thesocket;
   thesocket = socket(PF_INET, SOCK_STREAM, 0);
   if(thesocket == -1) cout <<"socket not created"<<endl;
   // larger container for a generalized address
   sockaddr address;

    // address container for an inaddress
    sockaddr_in *socketIn = (sockaddr_in *) &address;
    // Zero out inaddress part and set listening location (server addr)
    memset(socketIn,0,sizeof(sockaddr_in));
    socketIn->sin_family = PF_INET;
    socketIn->sin_port = htons(5600);
    socketIn->sin_addr.s_addr = htonl(INADDR_ANY);

    int error;
    error = bind(thesocket, (const struct sockaddr*)&address, sizeof(sockaddr_in));
    if(error == -1)
    {
        cout<<strerror(errno)<<"  unsuccessful bind"<<endl;
    }

    int backlog = 5;
    error = listen(thesocket, backlog);
    if(error != 0) cout<<error<<"  unsuccessful listening"<<endl;

    int client_data_size = sizeof(sockaddr_in);
    int acceptedSocket;
    acceptedSocket = accept(thesocket, &address, (socklen_t*)&client_data_size);

    if(acceptedSocket == -1) cout<<"acceptance error"<<endl;

   char buffer = 0;
   while(read(acceptedSocket,&buffer,1)>0)
   {
      file<<buffer;
       cout<<buffer;
   }

   file.close();
    close(thesocket);
}
```

System Model



Results

Establishing Connection

The provided client program, running on address 192.168.1.10, was able to make a successful connection to our program. Our server computer's IP address was at 192.168.1.20. Wireshark was able to capture the SYN – SYN ACK – SYN message sequence.

Illustration 1: Client Program Reports Successful Connection

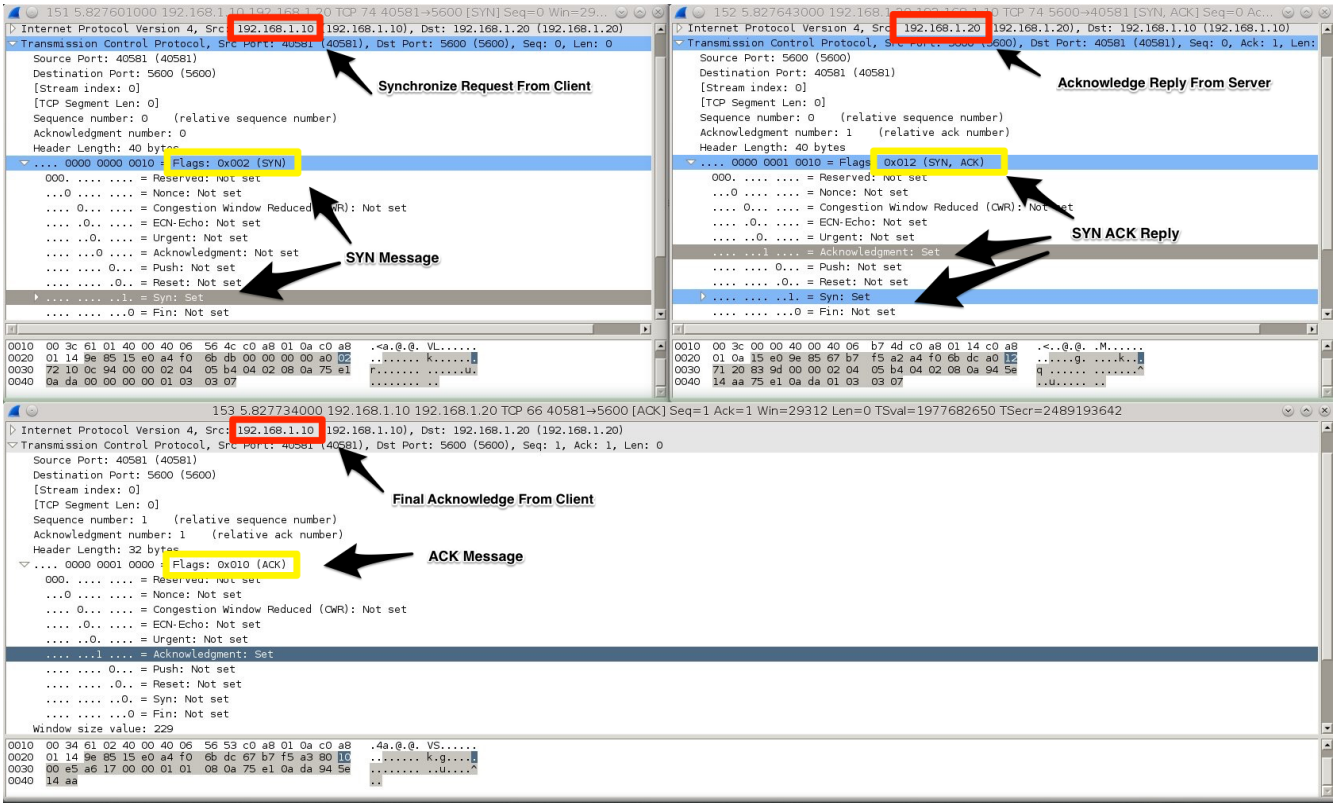


Illustration 2: Three Way TCP Handshake

Data Transmission

Two separate sets of data were send by the client. The data was the text of the Wabberjocky poem. They were almost the same message, except one letter was switched to 'z' in ascii. The Wireshark screenshots below show the data being sent.

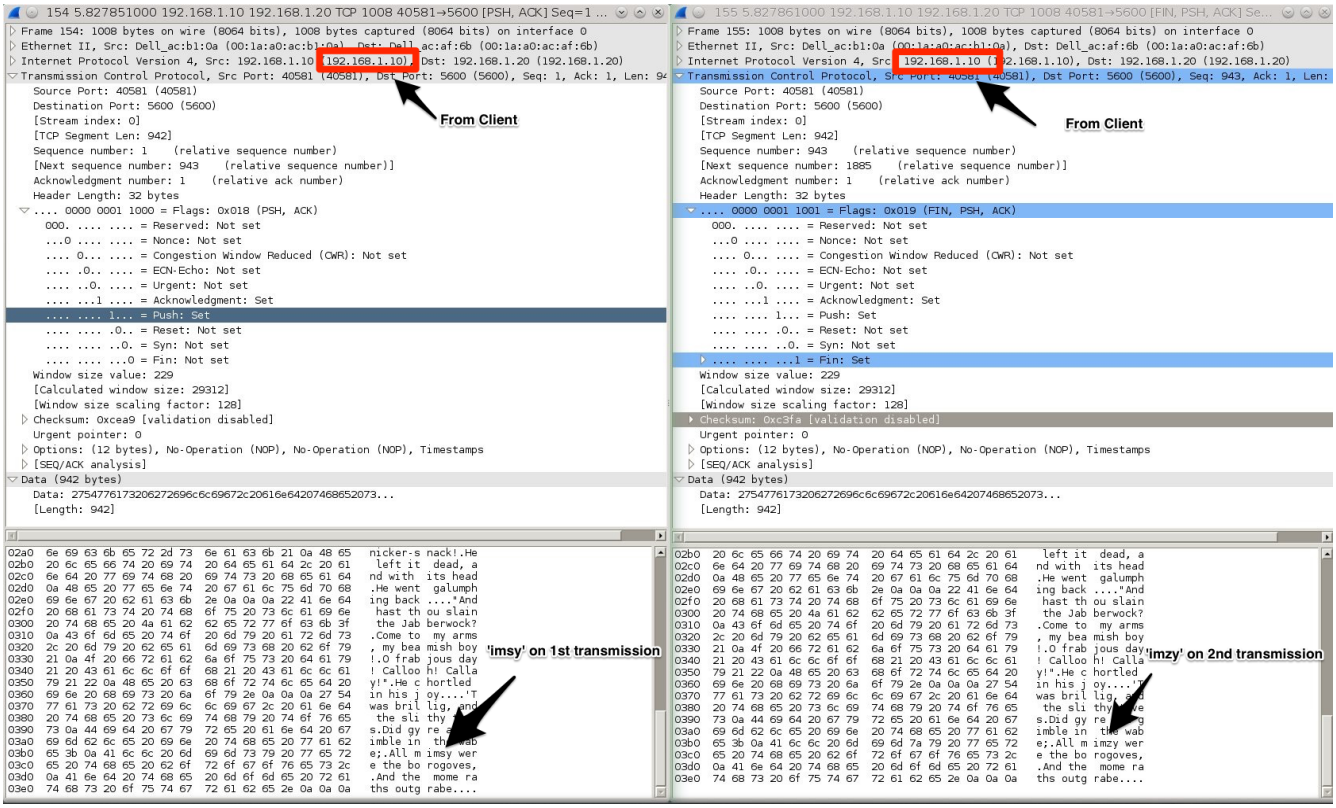


Illustration 3: Data Sent From Client

Connection Closure

The connection closure was initiated by the client. In the client's last data packet (figure above), a FIN flag was set. The server then responded with an ACK and also requested a FIN. The connection was closed once the client sent the final ACK. The Final packets are shown below in Wireshark.

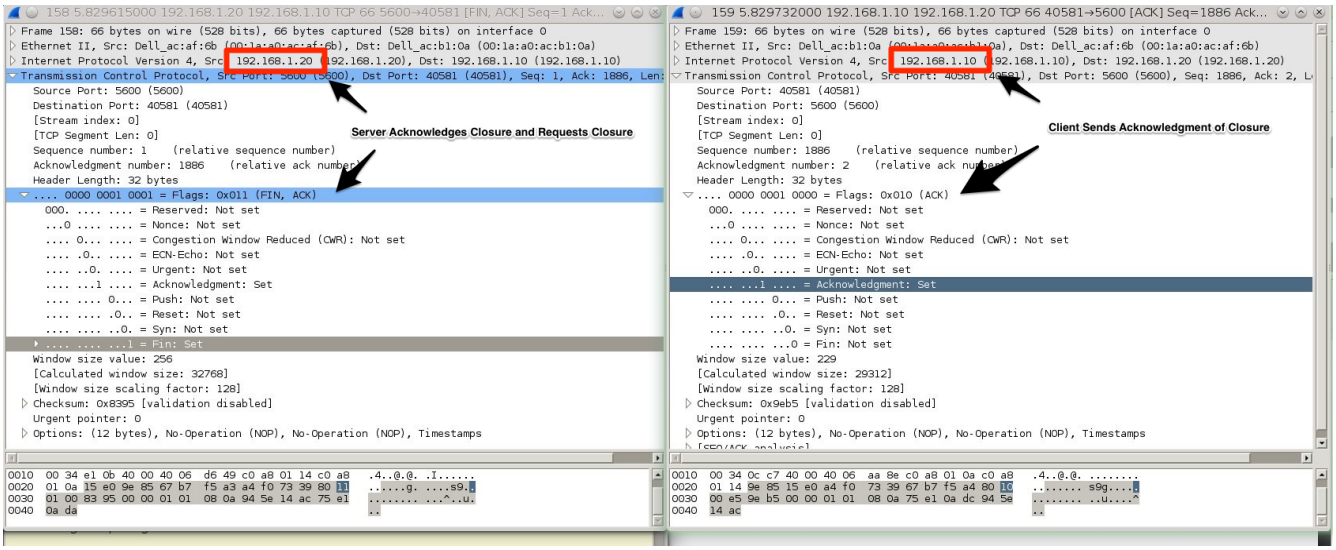


Illustration 4: Transmission Closure Packets

Conclusion

This project was extremely successful and allowed us to visually see how TCP connections are made. We ran into a problem when using a large buffer, where the data received became corrupted. This was solved by receiving one char at a time until no data remains.

It has been remarkable to see the progression of projects from the lower network levels up to this project, which reaches the application layer.