

COMP 4985 ASSIGNMENT #2 COMPARING TCP AND UDP OVER WIRED LAN

Patrick Wong
A00778993
February 19, 2020

Table of Contents

Abstract.....	2
Introduction	2
Results.....	2
Table 1: TCP Data	2
Table 2: UDP Data	3
Analysis	3
Conclusion.....	3
Appendix A: State Transition Diagram.....	4
Application View	4
Running Server View	5
Running Client View	6
Appendix B: Pseudocode	7
Application View Pseudocode	7
Server View Pseudocode	9
Client View Pseudocode.....	12
Appendix C: Test Cases	15

Abstract

This report examines protocol statistics of UDP and TCP through by creating an application to handle client/server requests. Due to buffer issues in the TCP implementation of the application, I can only obtain information regarding the transmission of TCP packets at 1024 bytes. Comparing UDP and TCP packet transfers with that much packet size indicates that UDP is faster than TCP.

Introduction

TCP and UDP are two popular protocols that devices today rely on to transmit information. In this study, we are going to examine the throughput of both protocols through the transmission rate, total data transferred, and total packets lost. At the end of this study, we can explain some properties that TCP and UDP and conclude their intended applications.

In order to compare the two protocols, we design a program to handle client and server traffic. One program will serve as a client, and another program will serve as a server. Based on user input parameters, such as packet size, sending frequency, and protocol, the user should be able to send either TCP or UDP over a local area network. The packet statistics will be examined through Wireshark.

Results

Table 1: TCP Data

Packet Size	Frequency	Number of packets received	Initial Time (s)	Final Time (s)	Time (s)
1024	10	10	82.62434	82.67658	0.052241
	100	100	344.627616	345.084365	0.456749
4096	10	N/A	N/A	N/A	N/A
	100	N/A	N/A	N/A	N/A
20000	10	N/A	N/A	N/A	N/A
	100	N/A	N/A	N/A	N/A
60000	10	N/A	N/A	N/A	N/A
	100	N/A	N/A	N/A	N/A

Table 2: UDP Data

Packet Size	Frequency	Number of packets received	Initial Time (s)	Final Time (s)	Time (s)
1024	10	10	61.78278	61.80321	0.020433
	100	100	94.77546	95.00712	0.231663
4096	10	10	23.97233	23.99953	0.027203
	100	100	35.19018	35.43241	0.242225
20000	10	10	45.54247	45.58175	0.03928
	100	100	10.02783	10.41748	0.389646
60000	10	0	0	0	0
	100	0	0	0	0

Table 3 Comparing UDP and TCP Data (1024 bytes)

Packet Size	Frequency	Number of packets received	TCP Transmission Time (s)	UDP Transmission Time (s)	%
1024	10	10	0.052241	0.020433	1.556697
	100	100	0.456749	0.231663	0.97161

Analysis

In UDP, I have trouble receiving packets of 60KB. This might be because of the limitation of UDP, where larger packet sizes seem to decrease the throughput. In Wireshark, the transmission of the packet resulted in a timeout and blocked the server from echoing the packets back to the client.

In TCP, there is a major issue in receiving data beyond 1024 bytes. The problem is not because of the nature of the protocol, but how I designed the application. Most likely a buffer overflow error happened in the program, but I could not locate where that arises. This problem affected me from collecting more data to test the integrity of the TCP protocol.

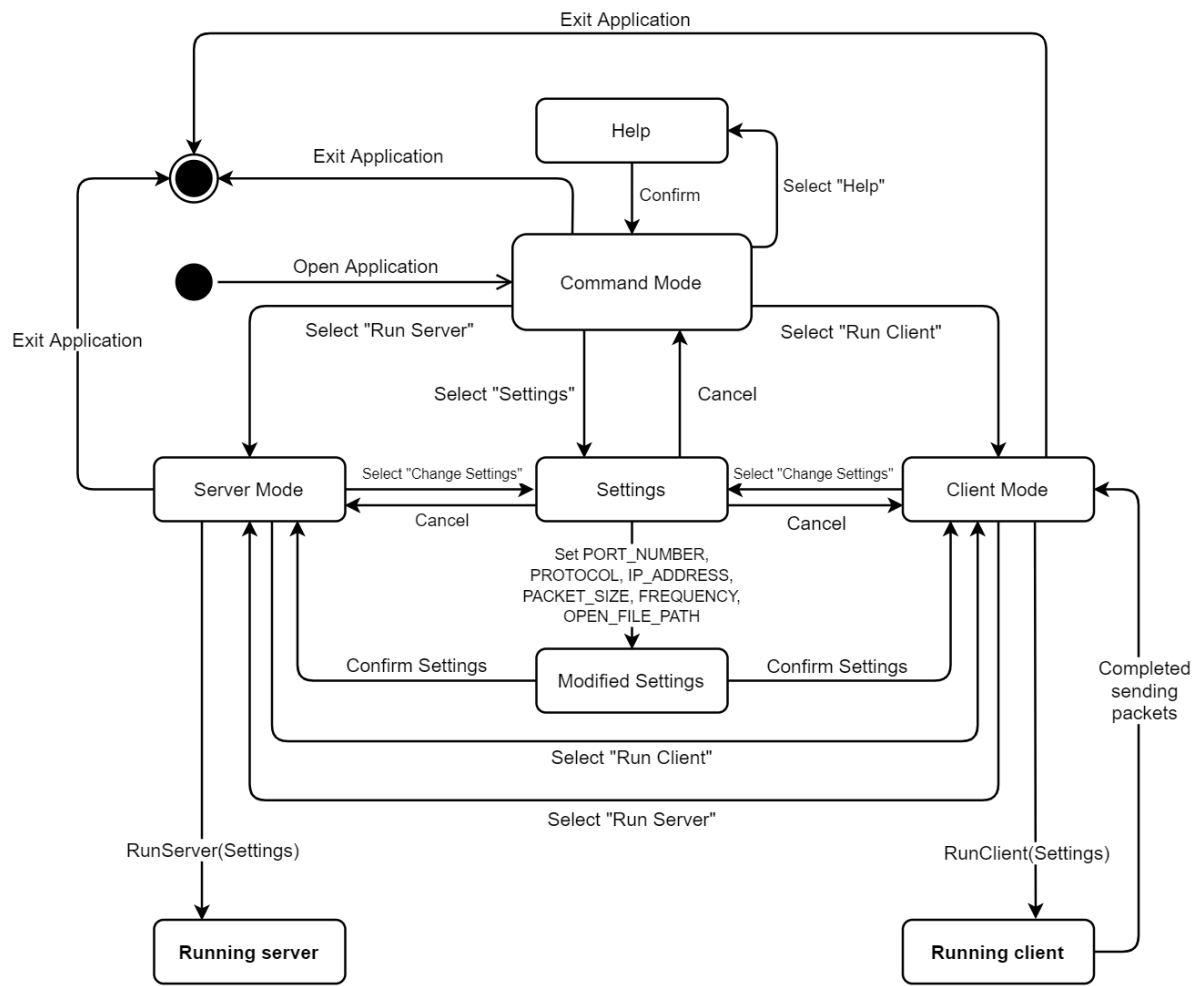
Based on the limited sample sizes, I can only compare the differences between sending 1024 bytes in TCP and UDP. Both TCP and UDP are able to receive the full amount of bytes and packets; however, the transmission speed is different. For both 10 and 100 number times, UDP transmission speed is 1.556697% faster and 0.97161% faster than TCP times respectively.

Conclusion

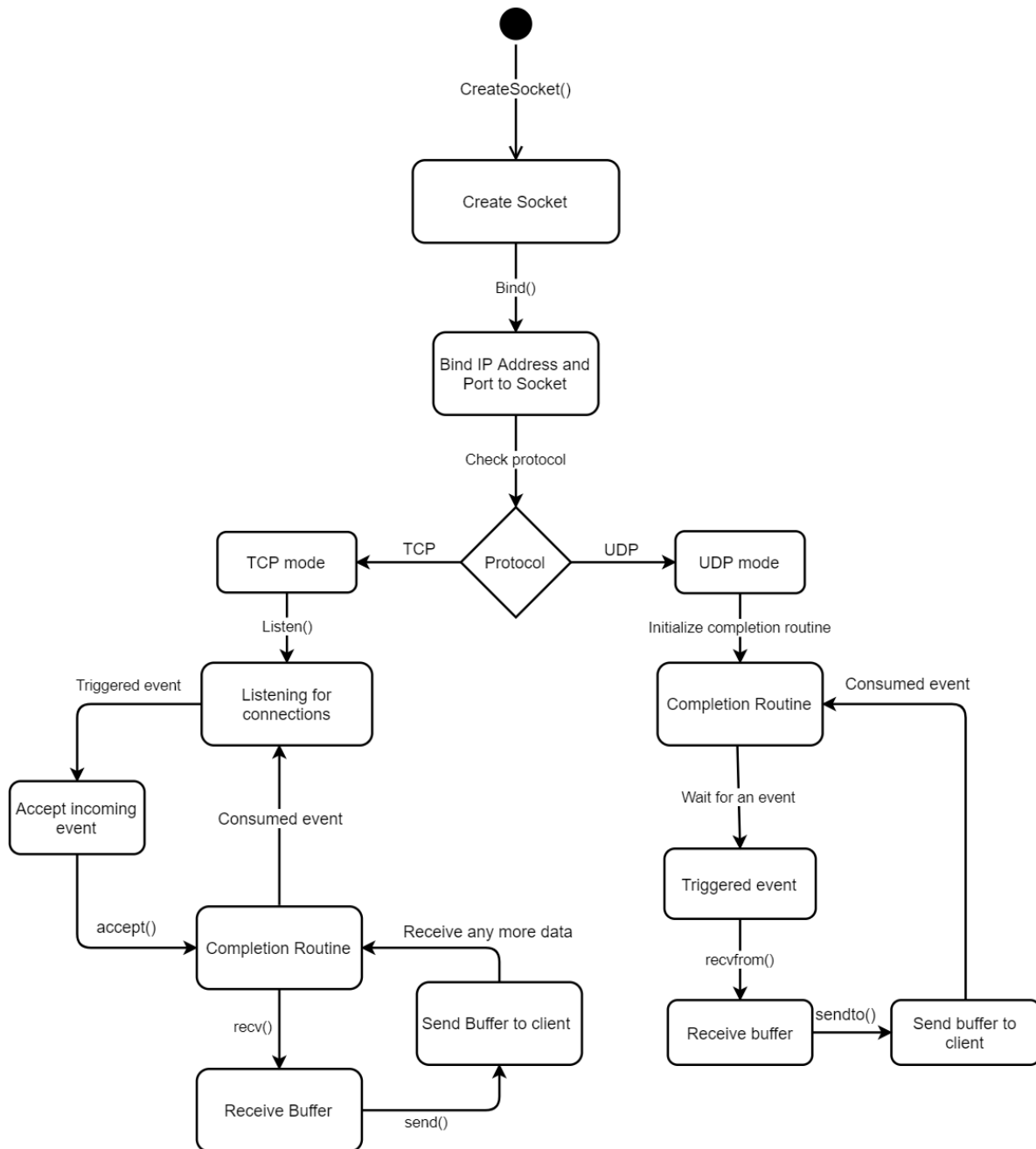
TCP transmission is generally slower than UDP transmission; however, UDP is unreliable at packet sizes of at least 60,000 bytes.

Appendix A: State Transition Diagram

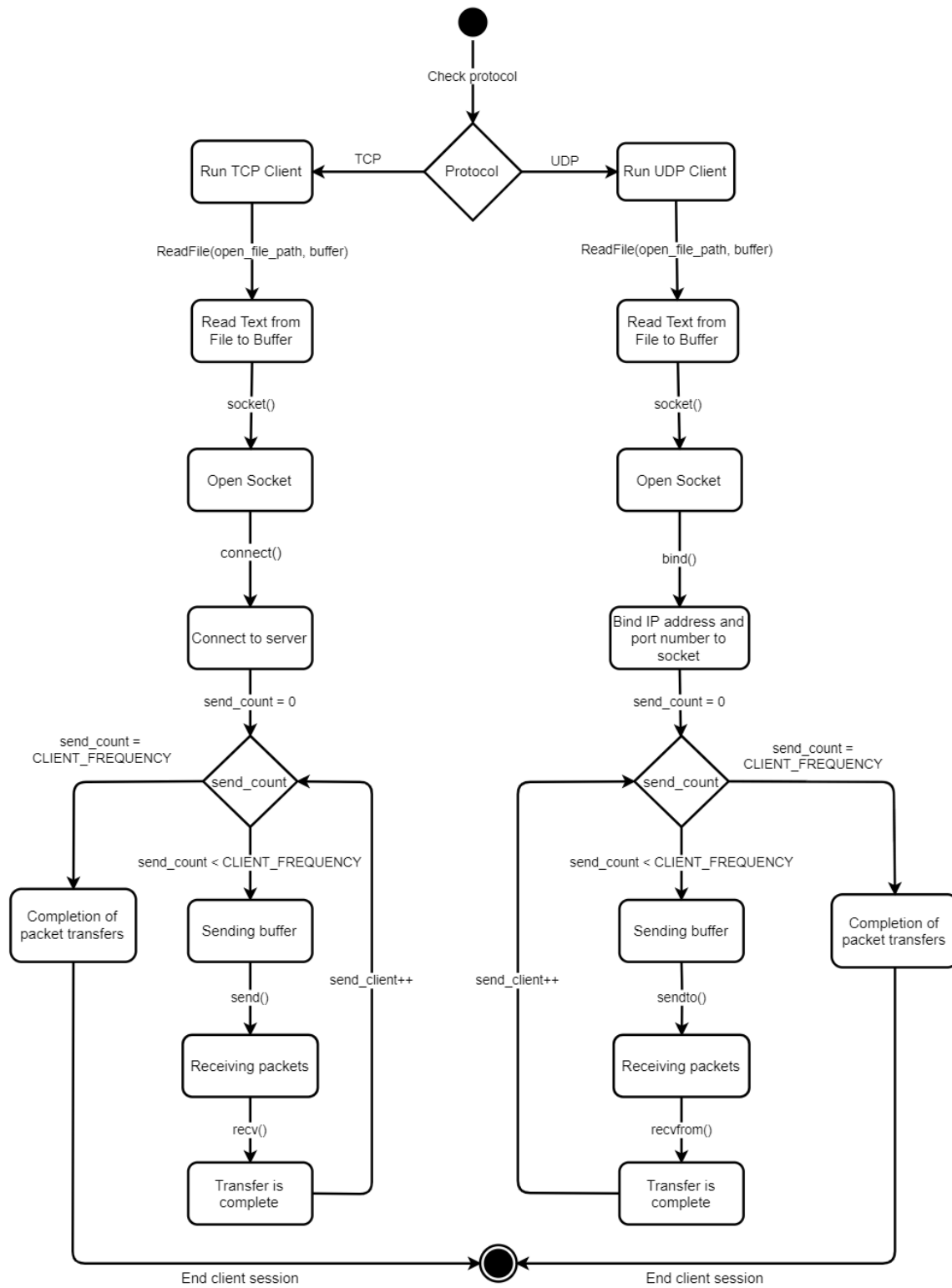
Application View



Running Server View



Running Client View



Appendix B: Pseudocode

Application View Pseudocode

Main

```
{  
    Initialize Windows structure  
    Fill default connection settings  
    Create Main Window  
    Create Server child window  
    Hide Server child window  
    Create Client child window  
    Hide Client child window  
    Show Main Window  
    Update Main window  
    Loop messages  
        Translate message  
        Dispatch message  
}
```


MainWindowProc

```
{  
    Loop through messages  
        If message is a menu  
            If 'Run Server' is clicked  
                Hide Client child window  
                Show Server child window  
            If 'Run Client' is clicked  
                Hide Server child window  
                Show Client child window  
            If 'Run Settings' is clicked  
                Show Settings dialog box  
}
```

Server View Pseudocode

RunServer

```
{  
    Initialize WinSock  
    Create a ListenSocket socket  
    Fill IP address and port number to the sockaddr structure  
    Bind sockaddr to ListenSocket  
    If protocol is TCP  
        Output to application that protocol is TCP  
        Attach a listener to ListenSocket  
        Create an AcceptEvent object  
        Create a TCPWorkerThread to handle accepted TCP events  
        Loop forever  
            Accept ListenSocket and assign its result to  
            AcceptSocket  
            Set AcceptEvent object  
    If protocol is UDP  
        Create a UDPWorkerThread to handle accepted UDP events  
}
```

TCPWorkerThread

```
{  
    Loop forever  
        Loop forever  
            If program receives an accepted event,  
                Break loop  
        Reset Event  
        Initialize socket information structure  
        Fill accepted socket information  
        Call Recv, passing in TCPWorkerRoutine to callback  
        task completion  
}
```

TCPWorkerRoutine

```
{  
    If error in transferring bytes  
        Return  
    If BytesRECV = 0  
        BytesRECV = ByteTransferred  
        BytesSEND = 0  
    Else  
        BytesSEND += BytesTransferred  
  
    If BytesRECV > BytesSEND  
        Databuffer = Buffer + BytesSEND  
        Bufferlen = BytesRECV - BytesSEND  
        Call Send, passing in TCPWorkerRoutine to callback  
        task completion  
    Else  
        BytesRECV = 0  
        Call Recv, passing in TCPWorkerRoutine to callback  
        task completion  
  
    If BytesRECV > 0  
        Get BytesRECV  
        Output to application about number of bytes received  
}
```

UDPWorkerThread

```

{
    Initialize socket information structure
    Fill socket information structure
    Loop forever
        Call Reform
        If Reform is pending
            Wait for overlapped events
            If there is an overlapped event,
                Output to application about number of bytes
                received
                Call SendTo to client
}

```

Client View Pseudocode**RunClient**

```

{
    Clear output application messages
    If protocol is TCP
        Create TCPClient thread to handle data transfers
    If protocol is UDP
        Create UDPClient thread to handle data transfers
}

```

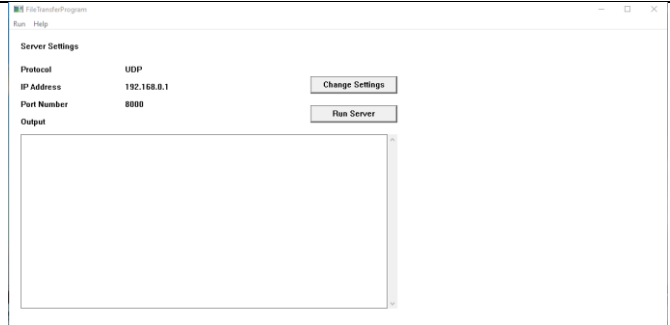
TCPClient


```
{  
    Initialize buffer to send  
    Read from file path and store contents in buffer  
    Initialize WinSock  
    Open Socket  
    Fill IP address and port number to sockaddr structure  
    Connect to server  
    Output successful connection message to application  
    Output buffer message to application  
    Output sending packet size to application  
    Get CLIENT_FREQUENCY from settings  
    Set count to 0  
    Loop CLIENT_FREQUENCY times  
        Call Send to send buffer to socket  
        Initialize buffer to receive  
        While socket is receiving  
            Call Recv to store receiving data to buffer  
    Increment count  
    Output count number of successful packets sent to  
    application  
}
```

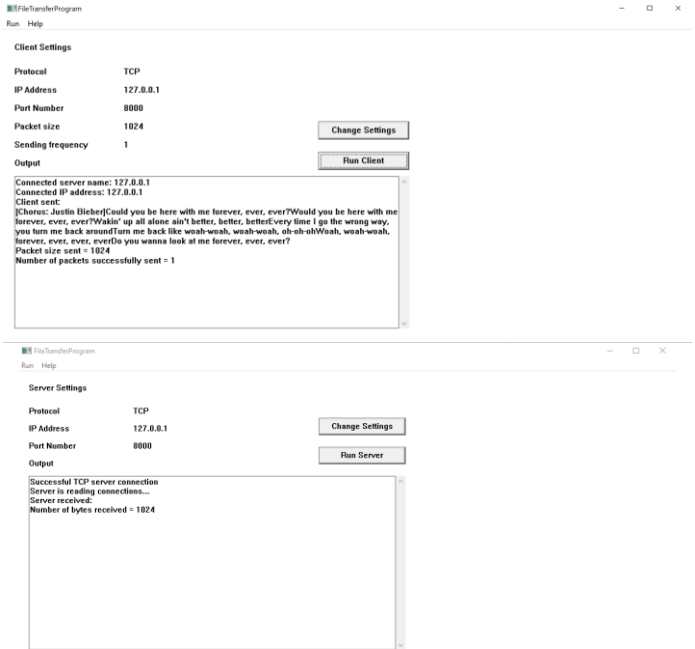
UDPClient

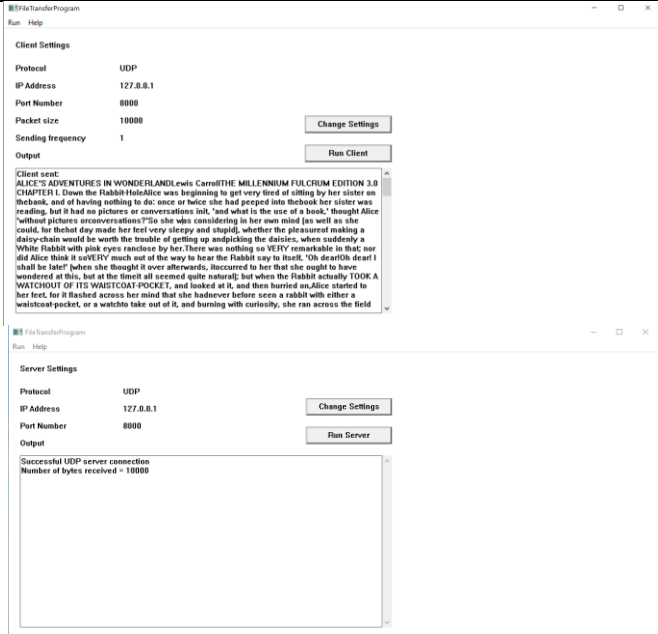
```
{  
    Initialize buffer to send  
    Read from file path and store contents in buffer  
    Initialize WinSock  
    Open Socket  
    Fill IP address and port number to sockaddr structure  
    Bind sockaddr to socket  
    Output successful connection message to application  
    Output buffer message to application  
    Output sending packet size to application  
    Get CLIENT_FREQUENCY from settings  
    Set count to 0  
    Loop CLIENT_FREQUENCY times  
        Call SendTo to send buffer to socket  
        Initialize buffer to receive data  
        Call RecvFrom to store receiving data to buffer  
        Increment count  
    Output count number of successful packets sent to  
    application  
}
```

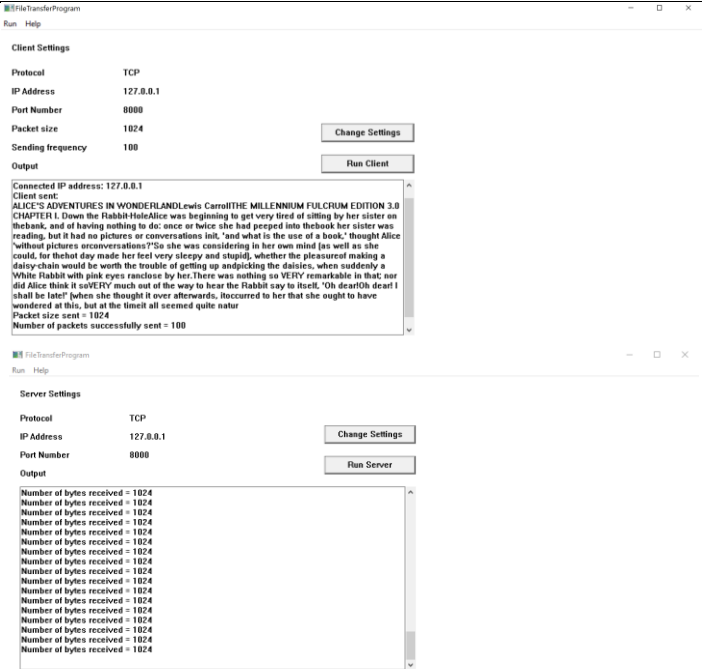
Appendix C: Test Cases

Test Case #1	Setting information on server window
Procedure	[Run] >> [Run Server] >> [Change Settings] >> Change IP Address from 127.0.0.1 to 192.168.0.1 >> [OK]
Expectation	You should see the updated IP Address on the server window
Screenshot	
Result	IP Address now reads 192.168.0.1
Test Passed?	Yes

Test Case #2	Running a UDP server to localhost
Procedure	[Run] >> [Run Server] >> [Run Server]
Expectation	You should see a message to confirm server connection.
Screenshot	
Result	Message output, "Successful UDP server connection"
Test Passed?	Yes

Test Case #3	Running a TCP client on localhost
Procedure	Assuming that a TCP server is connected to localhost, [Run] >> [Run Client] >> [Change Settings] >> Check "TCP" >> Open a file >> [OK] >> [Run Client]
Expectation	On the client, the user should see that the file content has been sent to the server. On the server, the user should see that the number of bytes received is the same as the packet size sent.
Screenshot	 <p>The screenshot displays two windows from the FileTransferProgram. The top window is the 'Client Settings' dialog, showing configuration for a TCP client connecting to 127.0.0.1 on port 8000 with a packet size of 1024. The bottom window is the 'Server Settings' dialog, showing configuration for a TCP server listening on 127.0.0.1 port 8000. Both windows include an 'Output' pane showing the results of the connection and data transfer.</p>
Result	Server outputs information when client sends a message.
Test Passed?	Yes.

Test Case #4	Sending a 10,000 byte datagram on UDP through localhost
Procedure	Assuming that a UDP server is connected to localhost, [Run] >> [Run Client] >> [Change Settings] >> Open a file >> Change "Client Packet Size" to 10,000 >> [OK] >> [Run Client]
Expectation	On the client, the user should see that the file content has been sent to the server. On the server, the user should see that the number of bytes received is the same as the packet size sent. The file used in this example is the first 10,000 byte text from the story, Alice in Wonderland.
Screenshot	 <p>The screenshot displays two windows from the FileTransferProgram. The top window is the 'Client Settings' dialog, showing configuration for a UDP client on localhost (127.0.0.1) at port 8080, with a packet size of 10000 bytes and a sending frequency of 1. The bottom window is the 'Server Settings' dialog, also configured for UDP on localhost at port 8080. Below the settings, the 'Output' window shows the message: 'Successful UDP server connection' and 'Number of bytes received = 10000', indicating that the server has successfully received the data sent by the client.</p>
Result	Server outputs information when client sends a message.
Test Passed?	Yes.

Test Case #5	Sending 100 packets of size 1024 bytes through TCP
Procedure	Assuming that a TCP server is connected to localhost, [Run] >> [Run Client] >> [Change Settings] >> Open a file >> Change "Client Send Frequency" to 100 >> Check "TCP" >> [OK] >> [Run Client]
Expectation	On the client, the user should see that the file content has been sent to the server. On the server, the user should see that the number of bytes received is the same as the packet size sent. Since there were 100 packets sent, the server displays a message for each packet. The file used in this example is the first 1024 byte text from the story, Alice in Wonderland.
Screenshot	
Result	On the client view, the user sees that all 100 packets of size 1024 bytes are successfully sent and echoed back. On the server view, the user sees a message for each client.
Test Passed?	Yes