



Networking with Windows

Sockets and File transfer



TYLER TREPANIER

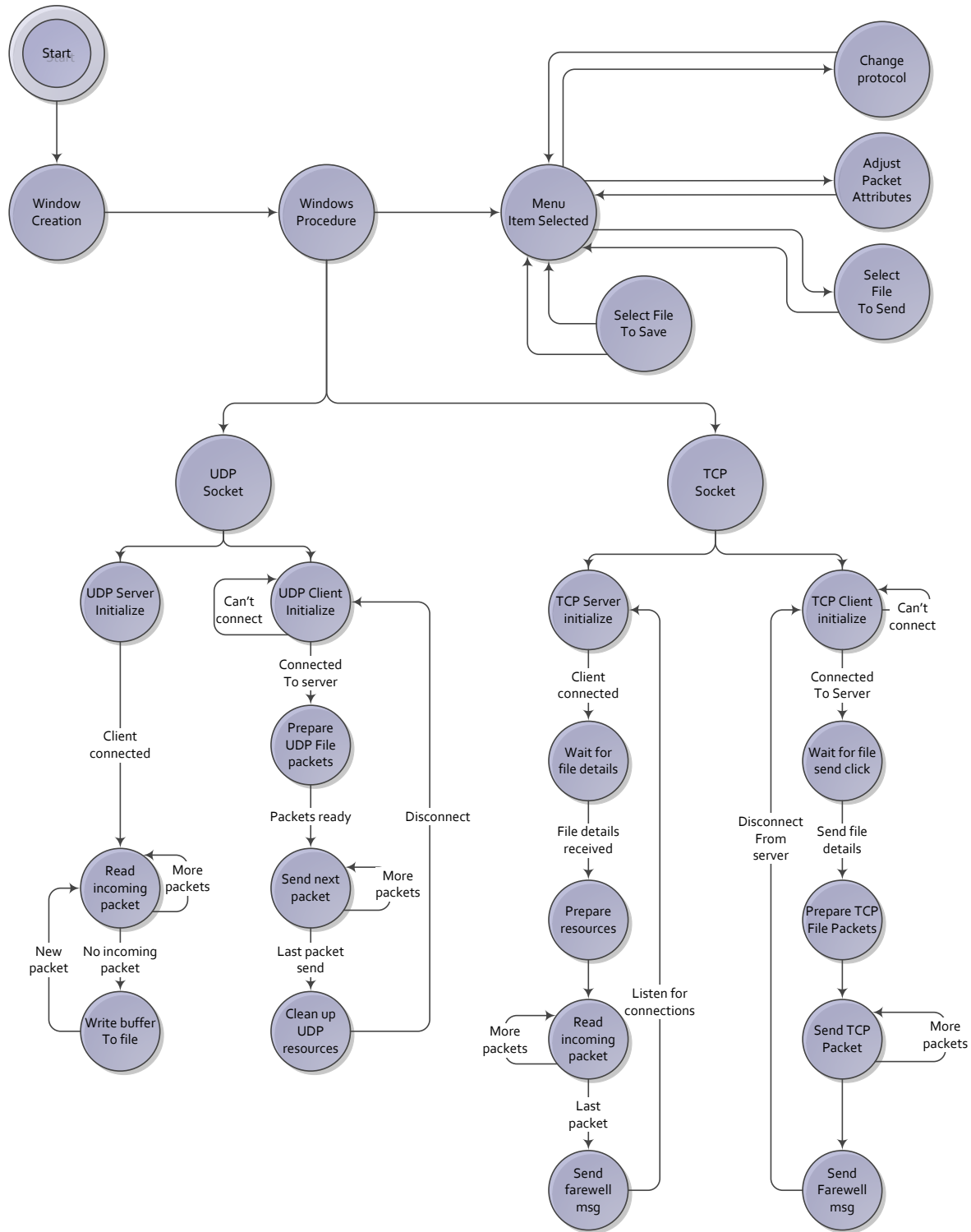
A00850517

Table of Contents

State Chart Diagram	4
Pseudocode	5
Main Windows Procedure	5
Win Main	5
Windows Procedure	5
UDP Socket Operations	8
UDP Socket	8
UDP Client Initialize	8
Prepare UDP File Packets	8
Send Next Packet	9
Clean up UDP resources	9
UDP Server Initialize	10
Read Incoming Packet	10
Write Buffer to File	10
TCP Socket Operations	11
TCP Socket	11
TCP Client Initialize	11
Wait for File Send	11
Prepare TCP File Packets	12
Send TCP Packet	12
Send Farewell Message	12
TCP Server Initialize	12
Wait For File Details	13
Prepare Resources	13
Read Incoming Packet	13
Send Farewell Message	13
Set Output Text	14
Append Output Text	14

Technical Report.....	15
Summary	15
Introduction.....	15
Test Case Scenarios and Results.....	16
Test Type: TCP	16
Test Type: UDP	18
Conclusion	19
Appendix.....	20
Figure 2.1: Test 1 for TCP final results.....	21
Figure 2.2: Test 1 for TCP Binary check using windows file checker.....	22
Figure 3.1: Test 2 for TCP [4k packet size] [Sent once]	23
Figure 3.2: Test 2 for TCP Binary check using windows file checker.....	24
Figure 4.1: Test 3 for TCP [4k packet size] [Sent 10 times]	25
Figure 4.2: Test 3 for TCP [File size verification]	26
Figure 5.1: Test 4 for TCP [20k packet size] [Sent once]	26
Figure 5.2: Test 4 for TCP [20k packet size] [Sent once]	Error! Bookmark not defined.
Figure 6.1: Test 5 for UDP [4k packet size] [Sent once]	27
Figure 7.1: Test 6 for UDP [4k packet size] [Sent once]	28
Figure 8.1: Test 7 for UDP [4k packet size] [Sent once]	29
Figure 9.1: Test 8 for UDP [60k packet size] [Send 200 times]	30
Figure 10.1: Final Conclusion	31

State Chart Diagram



Pseudocode

Main Windows Procedure

Win Main

This is the main entry point into the application.

1. Create the main window. Do not allow resizing.
 - a. If we are unable to create the window, terminate the program.
2. Call the Windows Procedure method.

Windows Procedure

This initially creates all child items inside of the main window. After processing the window create event, it begins an endless loop that handles all windows events that occurs in regards to this application. The events that will be handling are the button interactions, window creation and window destruction.

1. Initialize static string array, creates it only once to reduce memory usage.
2. Switch statement on windows events and messages (begin)
 - a. **On Window Creation:**
 - i. Create the menu options the user will use.
 - ii. Create the List where the statistics will show.
 - iii. Create the Text boxes where the user can input their IP address and port number.
 - iv. Default the settings to:
 - Protocol = TCP
 - Packet-size = 1kB
 - Frequency = 1 time
 - b. **UDP Socket Operation:**
 - i. Redirect this task to the UDP Socket function.
 - c. **TCP Socket Operation:**

- i. Redirect this task to the TCP Socket function.

- d. **Change protocol:**
 - i. If the Server or Client is already operating, ignore this request.
 - ii. Otherwise, depending on the user's choice, change the protocol.
- e. **Adjust the Packet Attributes:**
 - i. If the Server or Client is already operating, ignore this request.
 - ii. Otherwise, change the...
 - packet's size
 - how many times to send the file
- f. **Select a file to send**
 - i. Select a file to send and packetize.
- g. **Select a file to save:**
 - i. Select a file to save all incoming data to.
- h. **On Window Destruction:**
 - i. Close of child items of parent window.
 - ii. Close the parent window.
 - iii. Free all allocated server and client resources.
 - iv. Terminate program gently.
- i. **Default case:**
 - i. Put the message back into the Windows Event queue and ignore the message.
 - ii. Restart the Windows Procedure.

UDP Socket Operations

Requires a set protocol and the user to select either a client or a server. Note that a client does not properly work if there is no available server and will throw an error message.

UDP Socket

1. All UDP operations get directed to here.
2. IF the application is directed as a client...
 - a. Perform the UDP Client duties.
3. Else the application is directed as a server....
 - a. Perform the UDP Server duties.

UDP Client Initialize

1. Refuse to change behavior if this application is currently running as a Server.
2. Perform the WSASStartup to import the winsock dll library and start the networking service.
3. Create a designated UDP socket to use.
4. Grab the server's information (grabbed by the GUI) and bind that information to the socket we made.
 - a. If we can't connect to the server, **abort** all operations and deallocate all used resources.
5. Prepare the socket and server information for all windows procedure interactions.

Prepare UDP File Packets

1. Get the size required to store all the packets locally and allocate the required resources.
2. Read each individual portion of the file into its own individual allocated section of memory.
3. Place all portions into a useful vector for easy access and freeing.

Send Next Packet

1. While there's still packets in the queue
 - a. Extract the next packet from the queue and send it to the client.
 - b. Add additional information to the data statistics
 - i. Update packets sent and average packet send time
 - ii. Update total data sent / total transmission time
 - c. If there's still more packets, prepare the resources for sending again and restart this while loop.
2. Since sending is finished, clean up all the UDP resources.

Clean up UDP resources

1. Deallocate all used resources.
2. Disconnect the client from the Server.

UDP Server Initialize

1. Refuse to change behavior if this application is currently running as a Client.
2. Perform the WSASStartup to import the winsock dll library and start the networking service.
3. Create a designated UDP socket to use.
4. Set the requested port (GUI will grab from user input) and bind the Socket
5. Prepare the socket and server information for all windows procedure interactions.
6. Listen for all incoming UDP interactions.

Read Incoming Packet

1. There is a client that connected and the client is attempting to send packets.
2. Prepare a huge reading buffer (only once) that is the size of 65,535 bytes. This is the biggest piece of data that is allowed for UDP purposes.
3. Read in the UDP packet that is sent into the huge reading buffer.
4. Copy the received data into a separate writing buffer and reset the reading buffer.
5. Add additional information to the data statistics
 - a. Update packets received and average packet receive time
 - b. Update total data received / total transmission time
6. If there isn't another packet waiting to be read in the queue
 - a. Write buffer to File and empty the writing buffer.
7. Else there is another packet waiting to be read in the queue.
8. Prepare the Server to read again. Repeat this function.

Write Buffer to File

1. Whenever there isn't a datagram being processed, check to see if there is information in the writing buffer for new data.
2. All new data in the writing buffer will be written onto the file for saving.
3. Clear the new data from the writing buffer and return to the standard windows procedure.

TCP Socket Operations

Requires the TCP protocol to be set (comes default on initial startup and the user to select either a client or a server. Note that a client does not properly work if there is no available server and will throw an error message.

TCP Socket

1. All TCP operations get directed to here.
2. IF the application is directed as a client...
 - a. Perform the TCP Client duties.
3. Else the application is directed as a server....
 - a. Perform the TCP Server duties.

TCP Client Initialize

1. Refuse to change behavior if this application is currently running as a Server.
2. Perform the WSASStartup to import the winsock dll library and start the networking service.
3. Create a designated TCP socket to use.
4. Grab the server's information (grabbed by the GUI) and bind that information to the socket we made.
 - a. If we can't connect to the server, **abort** all operations and deallocate all used resources.
5. Connect to the server using the TCP socket opened. This automatically binds the socket.
6. Prepare the socket and server information for all windows procedure interactions.

Wait for File Send

1. Wait for the User to select the file they wish to send.
2. If there is an attempt to send a file without a selected file, politely ignore the request.
3. Once the file is selected, gather the file's information.
4. The user will click the Send menu option which will provoke the entire TCP sending process.
5. Prepare the TCP File Packets and Send the Packets.
6. After all the Packets are sent, close the connection to the server.

Prepare TCP File Packets

1. Get the size required to store all the packets locally and allocate the required resources.
2. Read each individual portion of the file into its own individual allocated section of memory.
3. Place all portions into a useful vector for easy access and freeing.

Send TCP Packet

1. While there's still packets in the queue....
 - a. Extract the next packet from the queue and send it to the client.
 - b. Add additional information to the data statistics
 - i. Update packets sent and average packet send time
 - ii. Update total data sent / total transmission time
 - c. Wait for a response from the client indicating that the packet was received.
 - d. Add additional information to the data statistics
 - i. Update packets received and average packet receive time
 - ii. Update total data received / total transmission time
 - e. If there's still more packets...
 - i. Prepare the resources for sending again and restart this while loop.
 - f. Else there's no more packets...
 - i. Send a Farewell Message to the Server and break out of this while loop.
2. Since sending is finished, clean up all the TCP resources and close the connection to the server.

Send Farewell Message

1. Deallocate all used resources.
2. Disconnect the client from the Server.

TCP Server Initialize

1. Refuse to change behavior if this application is currently running as a Server.
2. Perform the WSStartup to import the winsock dll library and start the networking service.
3. Create a designated TCP socket to use.
4. Grab the server's information (grabbed by the GUI) and bind that information to the socket we made.
 - a. If we can't connect to the server, **abort** all operations and deallocate all used resources.

5. Begin listening to the connected socket for incoming client requests.
6. Prepare the socket and server information for all windows procedure interactions.

Wait For File Details

1. When a client has connected, grab their internet information.
2. Wait for any incoming data from that client.
3. Initial data has been received, and the client wants to send a file. Grab the file details included in the message and Prepare Resources for the data to be received.

Prepare Resources

1. Allocate memory for all the incoming data to be received.
2. Get the user's file for information to be saved into.
3. Once finished, begin reading all the packets and updating the statistics of the information sent and received.

Read Incoming Packet

1. While the last packet hasn't been sent...
 - a. Prepare the reading buffer for receiving.
 - b. Read in the TCP packet that is sent into the reading buffer.
 - c. Add additional information to the data statistics
 - i. Update packets received and average packet receive time
 - ii. Update total data received / total transmission time
 - d. If the received packet isn't a farewell message...
 - i. Write the received data's contents into the file.
 - ii. Respond to the client with an acknowledgement of the packet.
 - iii. Restart reading in the while loop.
 - e. Else the received packet is the farewell message...
 - i. Respond to the client with a Farewell Message as well.
 - ii. Break out of this while loop.
2. At this point, the client will disconnect on its own accord.
3. The server will handle the client's disconnection.

Send Farewell Message

1. Send a message indicating that the Server knows of the disconnection by echoing the same message to the client.
2. Deallocate all reading resources for the purposes of the file transmission.

Set Output Text

Changes the text for the output text box to the specified text in the parameter. Requires the output text field to have been previously created.

1. Get the output child.
2. Clear the output text.
3. Insert the new specified text.

Append Output Text

Appends the output text with the specified text in the parameter. Requires the output text field to have been previously created.

1. Set text cursor to the end of the previously inserted text.
2. Insert the new text into the output.

Technical Report

Summary

Regardless of the protocol, it is highly recommended to have smaller data packets. The sending and receiving speeds are superior. One major benefit of these smaller packets is that it allows the server and the client to manage the data efficiently. The TCP protocol is great when it is required that the data being sent needs to be maintained along the way. However this approach sacrifices speed of transmission because of the other party's requirement to send an acknowledgement. When you require fast speeds and don't necessarily require all of the data to be in the right spot 100% of the time, UDP is your next best bet. UDP is all about speed of delivery since it lacks any acknowledgement from the other party.

Introduction

TCP and UDP have long withstood the test of time to become the standard internet protocol of the modern era. Many other protocols such as Gopher, AppleTalk and many others have been used but no longer see use outside of their hobbyist domains. In this section, the technical report, I created a program to test the limits of TCP and UDP protocols and to test each of their speed and reliability.

In order to prove the validity of my tests, I will be showing off their file size, their similarity (only when sent once), and keeping of all the data sent and received within the program itself.

Test Case Scenarios and Results

Test Type: TCP

TEST NUMBER	SCENARIO	EXPECTATION	RESULT	PASS/FAIL
1	Transmission of a 74MB file size using the TCP/IP protocol. It will transfer 1 times at a packet size of 1kB.	File transfer will be a success and the resulting file will share no differences. Also all the packets sent will be received.	Successful file transfer with no binary differences. All the packets sent received.	PASS
2	Transmission of a 74MB file using the TCP/IP protocol. It will transfer 1 times at a packet size of 4kB.	File transfer will be a success and the resulting file will have no differences and will be of the same size. Also all the packets sent will be received.	Successful file transfer. The file is 10 times the size of the original. All the packets sent received.	PASS
3	Transmission of a 74MB file using the TCP/IP protocol. It will be sent 10times at a packet size of 4kB.	File transfer will be a success and the resulting file will have no differences other than the file size being 10 times the size as the original. Also all the packets sent will be received.	Successful file transfer, file is 10 times bigger than the original file. All the packets sent received.	PASS

4	Transmission of a 74MB file using the TCP/IP protocol. It will be sent 100 times at a packet size of 20kB.	File transfer will be a success and the resulting file will have no differences. Also all the packets sent will be received.	Error, ran out of memory space. The resulting file became 14.2GB before the program had to close.	FAIL
---	--	---	---	------

Test Type: UDP

TEST NUMBER	SCENARIO	EXPECTATION	RESULT	PASS/FAIL
1	Transmission of a 74MB file using the UDP protocol. It will be sent 1times at a packet size of 4kB.	Some packet loss will occur but the majority of the file will remain intact. The speed will be faster than its TCP counter-part	Very little packet lost occurred.	PASS
2	Transmission of a 74MB file using the UDP protocol. It will be sent 1 times at a packet size of 20kB.	Some packet loss will occur but the majority of the file will remain intact. The speed will be faster than its TCP counter-part	The majority of the file is still intact. For the security of an intact datagram, we trade for speed.	PASS
3	Transmission of a 74MB file using the UDP protocol. It will be sent 10 times at a packet size of 60kB.	Some packet loss will occur but the majority of the file will remain intact. The speed will be faster than its TCP counter-part	Surprisingly, there was absolutely no packet loss.	PASS
4	Transmission of a 74MB file using the UDP protocol. It will be sent 200 times at a packet size of 60kB.	There will be a minor packet loss and the speed will increase compared to the smaller UDP packets.	The Server received 99% of the total data sent by the Client.	PASS

Conclusion

After a lot of experimenting, I can safely conclude that TCP is by far the most reliable and protocol to be used. In all of the cases, TCP managed to maintain a one-to-one ratio with its client in regards to the quantity of packets to be transmitted reliably. For all of the TCP connections (except for the 4th test which crashed), the data was transmitted in a reliable fashion with no error. You can see from TCP figures, (figures 2.2, 3.1 and 3.2) that the data is sent and received in the similar amount of packets.

UDP is a whole another story where even from the comfort of a local area connection, I experienced a drop in sent UDP packets. I'd imagine that if I were to stretch out and send from a farther location, I would experience a greater loss in packets. However, for local area connections and nearby connections, I can safely conclude that if the data does not require to be sent perfectly and in order, UDP is the way to go.

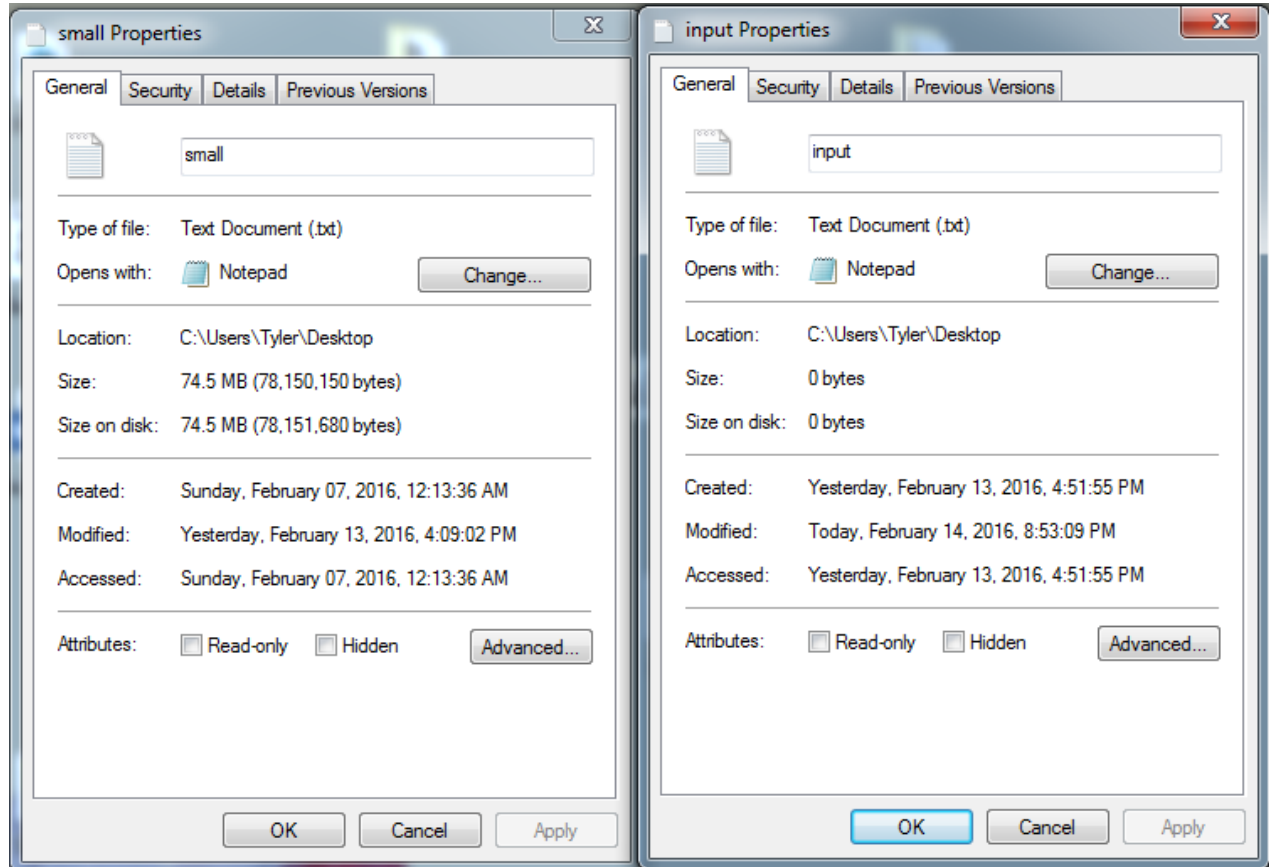
Another area I experimented in was speed of packets. When sending smaller UDP packets, regardless of the quantity of the packets, there was a huge increase in speed. Each packet took 0.02ms to send to another client. On the other side of the coin, reading UDP packets became a bit of a burden when the size and frequency of the packet began to increase. At a size of 60k packet size, it is no longer feasible to use UDP packets as the receiving time diminishes the gains in sending data.

TCP performed the worse in terms of speed for the simple fact that each sent packet must be acknowledged before sending the next packet. It's tedious work but ensures a stable packet.

To wrap it all up, when you have large file sizes, I recommend that you use TCP packets in addition to breaking down the data yourself into smaller chunks for more speed. When you are in need of fast packets but are capable of handling a missing packet or two, you should try for UDP. ALSO PLEASE NOTE, YOU CANNOT SEND A UDP PACKET LARGER THAN 65K! The datagram will be simply refused by the internet in its entirety.

Appendix

Figure 1.1: Properties of both the test files



The input file was constantly emptied in preparation for the next experiment.

Figure 2.1: Test 1 for TCP final results

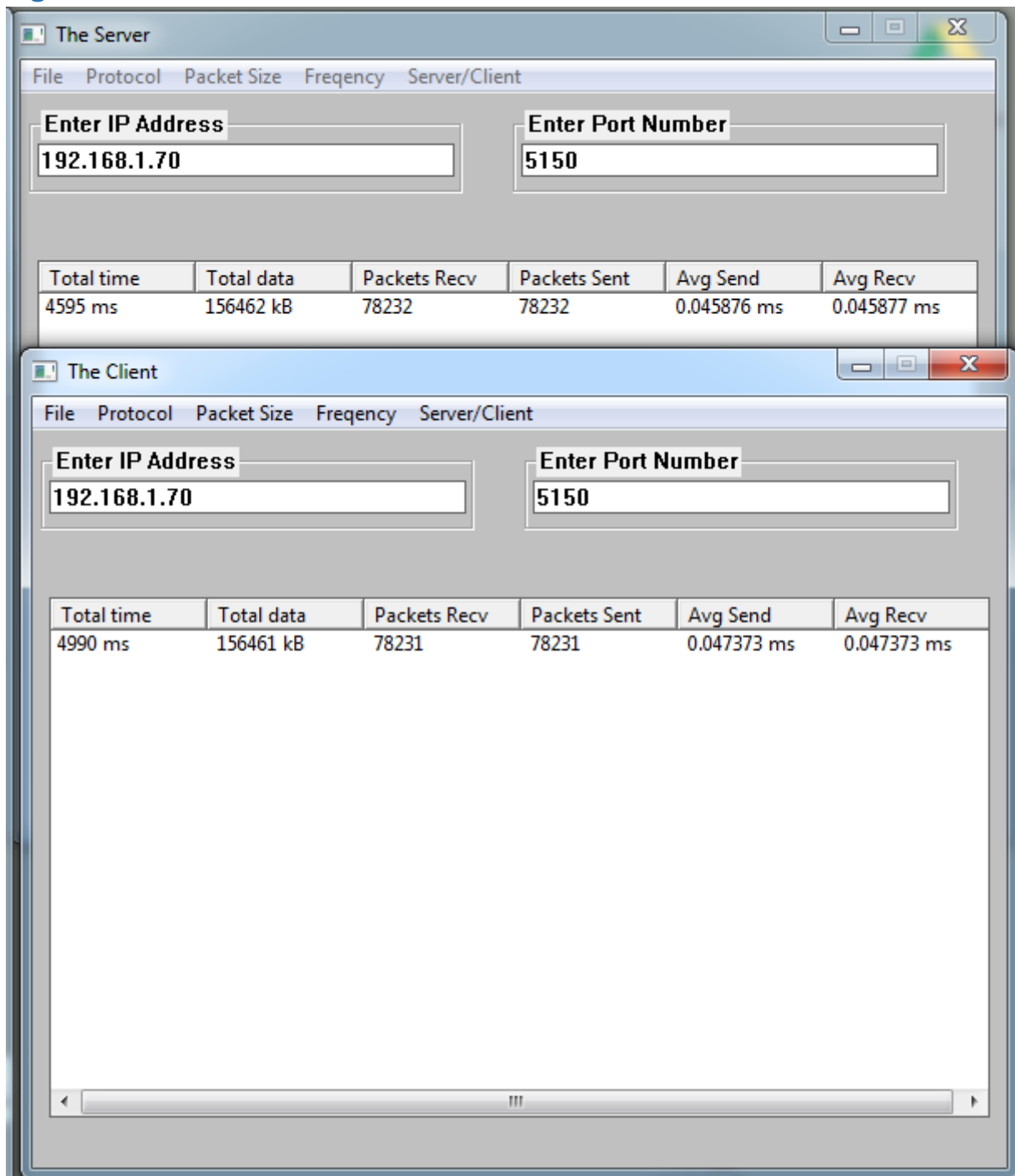
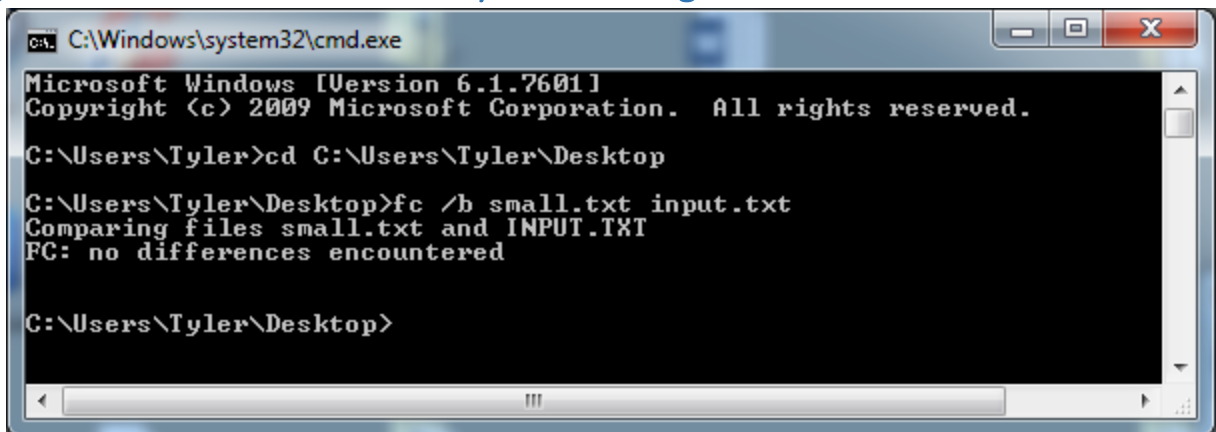


Figure 2.2: Test 1 for TCP Binary check using windows file checker



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Tyler>cd C:\Users\Tyler\Desktop

C:\Users\Tyler\Desktop>fc /b small.txt input.txt
Comparing files small.txt and INPUT.TXT
FC: no differences encountered

C:\Users\Tyler\Desktop>
```

The test is verified using the filechecker associated with windows. The switch '/b' checks both files for a binary equivalence. Windows has a file checker program which can be accessed using:

Fc /b [filepath1] [filepath2]

Figure 3.1: Test 2 for TCP [4k packet size] [Sent once]

The Client

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
1748 ms	156356 kB	19545	19545	0.070504 ms	0.070508 ms

The Server

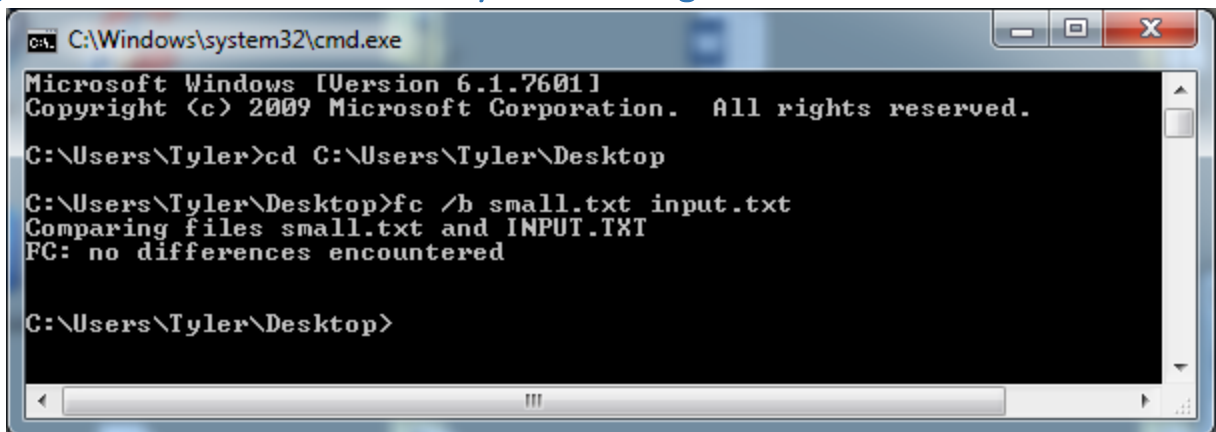
File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
1859 ms	156360 kB	19546	19546	0.076742 ms	0.076746 ms

Figure 3.2: Test 2 for TCP Binary check using windows file checker



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Tyler>cd C:\Users\Tyler\Desktop

C:\Users\Tyler\Desktop>fc /b small.txt input.txt
Comparing files small.txt and INPUT.TXT
FC: no differences encountered

C:\Users\Tyler\Desktop>
```


Figure 4.1: Test 3 for TCP [4k packet size] [Sent 10 times]

The Client

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
17472 ms	1563452 kB	195432	195432	0.071467 ms	0.071468 ms

The Server

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
18789 ms	1563456 kB	195433	195433	0.078267 ms	0.078268 ms

Figure 4.2: Test 3 for TCP [File size verification]

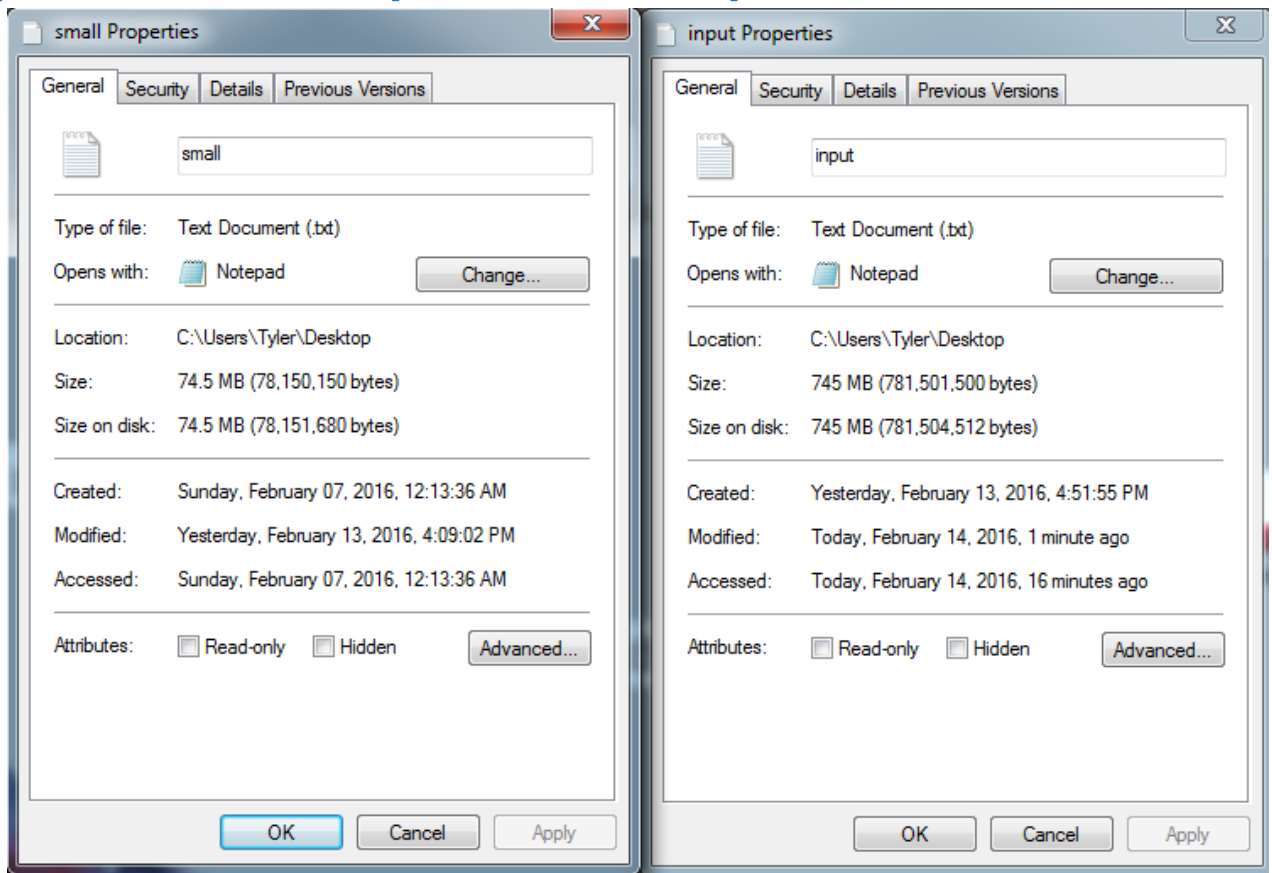


Figure 5.1: Test 4 for TCP [20k packet size] [Sent once]

No figure available.

Figure 6.1: Test 5 for UDP [4k packet size] [Sent once]

The Client

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
729 ms	78169 kB	0	19544	0.037300 ms	0.000000 ms

The Server

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
429 ms	77853 kB	19465	1	0.000000 ms	0.022040 ms

Figure 7.1: Test 6 for UDP [4k packet size] [Sent once]

The Client

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
199 ms	78154 kB	0	3909	0.050908 ms	0.000000 ms

The Server

File Protocol Packet Size Frequency Server/Client

Enter IP Address: 192.168.1.70

Enter Port Number: 5150

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
115 ms	77874 kB	3895	1	0.000000 ms	0.029525 ms

Figure 8.1: Test 7 for UDP [4k packet size] [Sent once]

The image shows two overlapping windows from a network testing application. The top window, titled 'The Server', has a menu bar with 'File', 'Protocol', 'Packet Size', 'Frequency', and 'Server/Client'. It contains two input fields: 'Enter IP Address' with the value '192.168.1.70' and 'Enter Port Number' with the value '5150'. Below these is a table with the following data:

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
46 ms	78151 kB	1304	1	0.000000 ms	0.035276 ms

The bottom window, titled 'The Client', has the same menu bar. It also has 'Enter IP Address' (192.168.1.70) and 'Enter Port Number' (5150) fields. Its table shows the following data:

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
90 ms	78151 kB	0	1304	0.069018 ms	0.000000 ms

Both windows have a scrollbar at the bottom.

Figure 9.1: Test 8 for UDP [60k packet size] [Send 200 times]

The image shows two windows from a network testing application. The top window, titled 'The Server', has a menu bar with 'File', 'Protocol', 'Packet Size', 'Frequency', and 'Server/Client'. It contains two input fields: 'Enter IP Address' with the value '192.168.1.70' and 'Enter Port Number' with the value '5150'. Below these is a table with the following data:

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
11159 ms	15557.358000 MB	259386	1	0.000000 ms	0.043021 ms

The bottom window, titled 'The Client', has a similar menu bar. It also has 'Enter IP Address' (192.168.1.70) and 'Enter Port Number' (5150) fields. Below them is a table with the following data:

Total time	Total data	Packets Recv	Packets Sent	Avg Send	Avg Recv
19323 ms	15630.200000 ...	0	260601	0.074148 ms	0.000000 ms

Both windows have a scroll bar at the bottom.

Figure 10.1: Final Conclusion

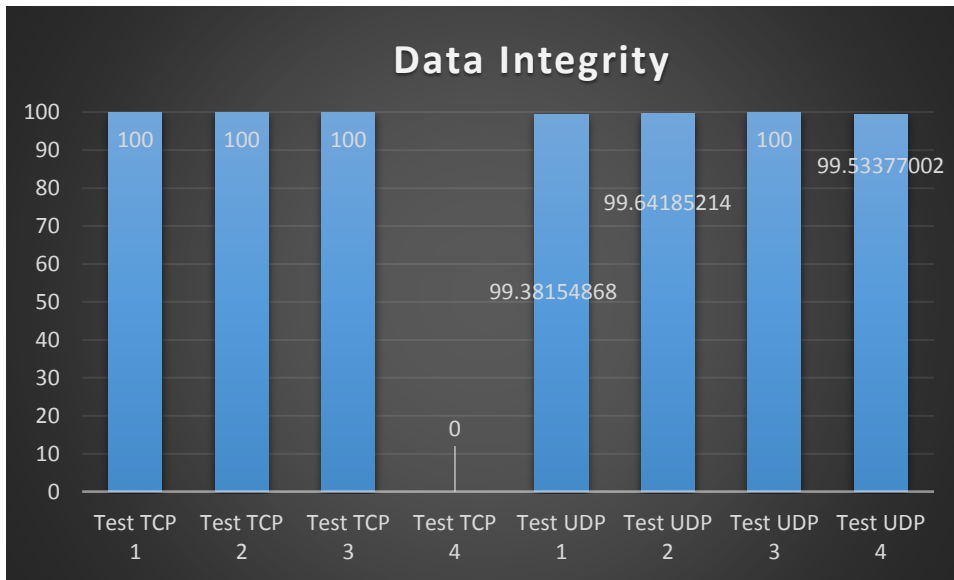


Figure 10.2: Average time per packet

