Network Latency Tool

Final Report

Submitted for the BSc in
Computer Science

May 2017

by

Robert Steven Waterhouse

Word Count: 4271

Table of Contents

1. Introduction

1.1 Report Content

This project's title is 'Network Latency Tool' and the goal of this project will be to create an application that a user could use to simulate latency and packet loss over a network. This report will explore the project's aims and objectives as well as evaluating the finished project at the end. The aims and objectives section elaborates on each goal the project wishes to achieve as well as giving some detail as to how the objectives will be measured when they have been completed. The background section details the context of the problem as well as a strong comparison to alternative solutions which already exist. The background also provides a good insight into some of the technologies used in the project as well as a comparison against other technologies which the project could have used.

The technical development section focusses on the design choices made throughout development of the software as well as giving insight as to how the program functions.

1.2 Report Overview

This project will explore the effects of manipulating a connection on a network. The project will strive to help achieve an understanding of how the degradation of a network can affect a connection's integrity. This will be accomplished by designing and implementing a system in which clients can communicate with each other, with an addition of a tool sitting in-between the connection, providing latency and packet loss at the user's request. An interactive demonstration will provide insight as to how the certain values of latency and packet loss can affect the data's reliability.

2. Aim and Objectives

The project's aim is to gain an insight into the effect network degradation has on a network connection / network reliant program. Some issues to highlight would be the result of packet loss / latency on a network connection and how it effects the reliability. These values can be scaled so the network's degradation can be seen in real time with varying results.

This will be achieved by completing the following objectives.

Objective 1: Design a client that with basic networking functionality
- This will be basic functionality for the project, for the whole system to function, there will have to be a client capable of sending and receiving packets to and from another client as well as receiving packets from a server.

Objective 2: Design a tool to connect to two clients simultaneously
- To develop further steps, A tool will need to be made to forward messages from one client to another.

Objective 3: Add latency manipulation to the network connection.
- This would be the first objective to allow the main aspect of the tool to function, the system should allow for a range of values to be added to the latency of the connection.
- This goal will be measured by testing whether latency has been added to the connection.

Objective 4: Add packet loss manipulation to the network connection.
- This would be a follow-up objective to the latency. The system should allow for a percentage packet loss value to be added to the connection to cause instability.
- This goal will be measured by testing whether packet loss has been added to the connection.

Objective 5: Add data logging to allow further analysis of the data post-testing.
- This allows me to export the data once a test has been completed, to allow for extra analysis even once the program has been shut down. This will be done by exporting the data to a file.

Objective 6: Build an interactive demo to show the network's performance in real time
- A demo will show how the network connection is currently performing. The demo will be able to show to the user how the current degradation settings are currently effecting the connection.
- Testing this demo involves having completed steps 1-4 to give the demo some real data and to then test whether the results are as expected.

3. Background

The basics of this project include using a networked connection between two devices. The two devices will be able to communicate to each other independently and but will also be able to connect through another application, the tool. The tool will artificially add packet loss and network latency between the two clients/devices. The effect of manipulating the connection will be measured by logs of incoming data as well as an interactive demo which will display how to degradation would affect a program is a real-life situation.

3.1 Problem Context

A problem that programmers around the world have is designing and manufacturing systems that handle of a lot of data being sent and received from clients. However, not all client's network connections are always the same, some may be more stable whereas others may be faster but unstable. It is hard to test how the connection would vary between certain devices across the world with varying ping, packet loss and connection stability. Therefore, a tool that could simulate these environments would be beneficial to these programmers as it would allow them to change their system or take precautions depending on how the client's connection performs when using their applications.

Verizon [2] states that "90ms [Latency] or less for transatlantic round trips between London and New York." And "99.5 percent or greater for regional round trips within Europe and North America.". Therefore, it is vital that this project can simulate a connection such at this in order to effectively achieve the task of simulating a real-world network.

3.2 Comparison of Technologies

The main technology used in this project is the TCP standard for networking which is used to create and maintain a networked connection between two devices. TCP is a connection oriented protocol meaning that it is essential for two devices to be able to communicate with one another.

Another protocol that could have been used would be UDP. UDP is a connectionless protocol, this means that one device could send a mass of packets to another without the receiving device needing to respond. UDP packets are not sent in an order, this is particularly useful in game programming as the application can send masses of updates to the server without having to wait for the packets to be acknowledged.

The reason TCP was chosen as the preferred communications protocol in this project was since TCP is a reliable connection, coupled with the fact that TCP orders each packet means programming in this protocol is a lot less hassle-free.

When choosing which language, the project should be written in, a few could have each been beneficial in their own way. C# was the language which was decided upon due to its extensive libraries supporting TCP programming.

The following table describes the benefits of each language when deciding which to use:

Language	Complexity	Libraries / Performance	Suitability
C#	Object-orientated. Very standard language in terms of syntax. Debugging should be straight forward. Programming time should be average.	Extensive library support which supports TCP connections built-in. C# performs perfectly well for most applications.	Overall C# has all the solutions required to make this project work efficiently.
C++	Object-orientated. The complexity of coding with C++ is quite troublesome. Overall programming time is extended compared to C#, meaning less time is spent coding the same thing.	C++ has good library support for TCP connections built-in. C++ is one of the fastest languages in terms of processing time.	C++ is not suitable for this project mainly due to its programming time. C++'s benefits from its processing time does not outweigh the cost of extra programming time.
Java	Object-orientated. Java is very like C# in terms of complexity, it's syntax is almost identical.	Java has sufficient libraries for TCP connections built-in.	Java is a suitable language.

Figure 1. Table comparing language pros and cons

3.3 Alternative Solutions

Alternative solutions for this project's problem would include a program called Traffic Shaper XP [1]. This program allows you to easily limit the network connection of a program such as Internet Explorer by capping its network throughput. It achieves this by queueing all the data being received by the application and releasing it at a chosen interval. This method is simulating a network which has a weaker network bandwidth by forcing all data to go through a buffer/queue.

Another alternative solution would be a program called TMNetSim [3]. *"TMnetsim is used to simulate a wide-area network for a single protocol. It is used to create test situations that simulate real-world situations in a reproducible way. TMnetsim works with TCP based socket protocols."* This solution is a well-rounded alternative. It allows for great flexibility when choosing exactly which ports you want to be working with. With it's user interface, the program allows the user to easily control the options of the connection, such as packet loss, latency, latency jitter etc. However sometimes getting the correct configuration working would take a while, other times it wouldn't load at all.

One solution for Linux based operating systems would be to use a program called “netem” [4]. *“netem provides Network Emulation functionality for testing protocols by emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering.”* This solution offers excellent control with the use of its command line interface. However, this solution does not perform well in visualization of the network’s degradation, all feedback of the program’s execution is in log form.

3.4 Useful Tools

A useful tool in development of these types of applications would be “TCPView” [5]. This programs show all active TCP and UDP endpoints currently connected to the system. This allows the user to see details such as the current connection state, total / sent received packets / bytes, local / remote addresses / ports and protocols used. Data such as this would be useful to include in the final product. This would also help as a diagnostic tool when the program is running. Including this data into any externally saved file will also be useful as you would allow the data to gain some context of the connection that took place, rather than it being a simple log of what happened.

4. Technical Development

The section will outline some of the development strategies and decisions made throughout the design and implementation of the project.

4.1 Network Communications

At the start of the project, when designing how the program should be written, there was two main options to consider regarding network communications. The first option would be to use sockets [6] and the second option would be to use C#'s built-in TCP client [7] class.

4.1.1 Sockets

Sockets are a good method to use for the implementing networked communication. Sockets give the programmer great flexibility and allow programmers to control most aspects of the connection in which they need. The downside to sockets are that they can be complicated to develop, these complications can be detrimental to the speed of developing an application.

4.1.2 TCP Client

The TCP Client is included in one of C#'s standard libraries. It gives the programmer a much higher-level access to the networking tools. This therefore allows the programmer to set up a networked-program without needed to worry about some of the minor details that comes along with developing with sockets. Because of this, a programmer can actively maintain and develop a networked-application with greater ease and efficiency.

During development of the applications, it was found that the default TCP Client class did not have an accessible member property of "Available". A workaround was found that involved creating a custom version of the class and writing my own get method.

```
namespace Client
{
    /// <summary>
    /// Wrapper around TcpListener that exposes the Active property
    /// </summary>
    public class TcpListenerEx : TcpListener
    {
        /// <summary> Initializes a new instance of the T:System.Net.Sockets.TcpListener class with the specified local endpoint ...
        public TcpListenerEx(IPEndPoint localEP) ...

        /// <summary> Initializes a new instance of the T:System.Net.Sockets.TcpListener class that listens for incoming connect ...
        public TcpListenerEx(IPAddress localaddr, int port) ...

        public new bool Active
        {
            get { return base.Active; }
        }
    }
}
```

Figure 2. Custom TCP Listener Class

4.1.3 Conclusion

After attempting to create both programs and finding that sockets was a lot more complex, it was decided to use the TCP Client class to handle the network. This was the best choice as it allows for easier and more streamlined development of the code, as well as providing a simpler code base that allows for easy debugging.

4.2 System Design

The system was designed such that a user can use two clients independently. This allowed them to connect and send data to each other whilst also allowing each of them to connect to any other client/application. A tool was also designed that allows for two incoming connections.

Once all 3 applications are connected (See figure below), any data that one client sends will be passed on through the tool. The tool will then hold that message for a given amount of time (Latency) and will also attempt to discard the message depending on the packet loss rate in which the user provides.

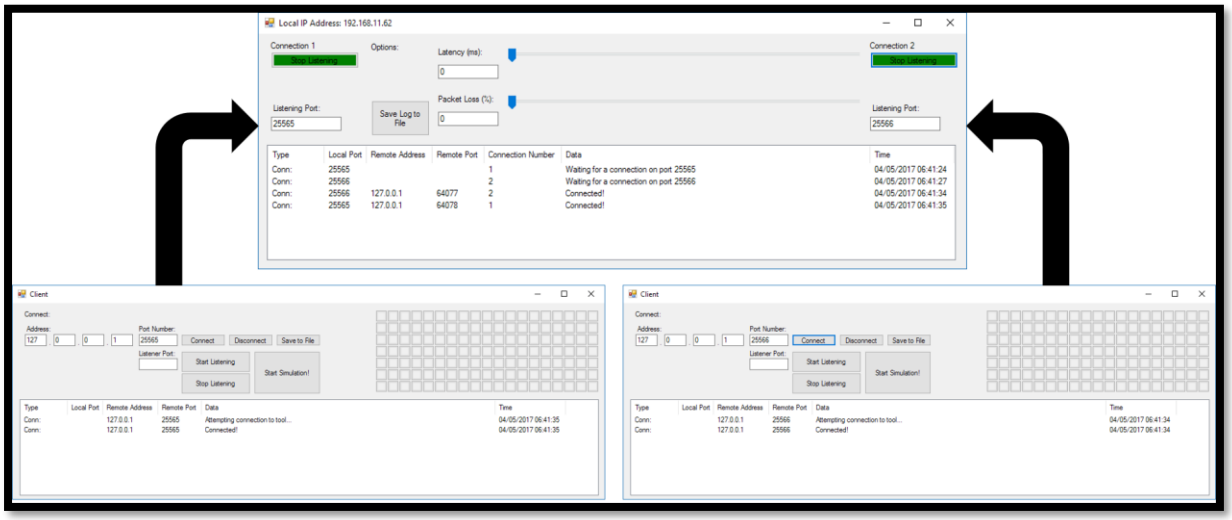


Figure 3. Example network setup

4.2.1 Class Design

The design of the software's classes is setup with manageability in mind. There are several tasks that have been split up and given to a class separately. General roles of classes include the logging of the data, the managing of the connection and custom button classes. Allowing the classes to perform more of a general task allowed programming the software to be straightforward, while this did leave some classes with a fair amount of functions it was overall a positive decision allowing for easier programming.

The connection class was similar for both applications, client and tool. It would manage initiating a connection with another application, listening for a connection from another application, sending data, reading incoming messages, closing connections and stopping listeners.

The client also had a logger class which handles importing data from the log window in the application and saves it to a file.

4.2.2 Client UI Design

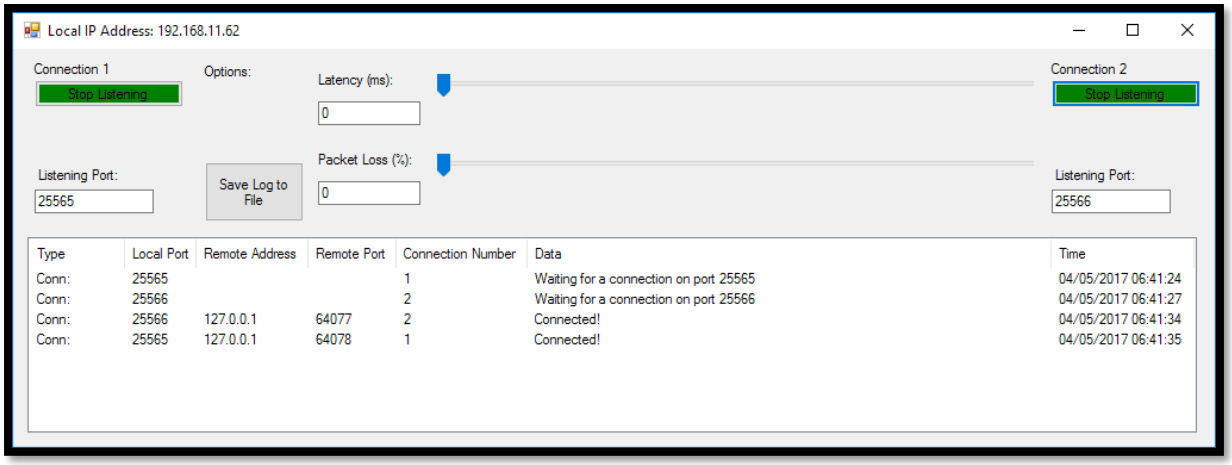


Figure 4. The client

The design of the client is very simple, designed such that each section is self-explanatory and intuitive. The client is split into 3 main sections.

The first thing the user should see is the form controls at the top left of the UI, this is important to make sure the user knows where to start. This section has all the tools the user needs to set up the connection. Input into the application is made as simple as possible with input fields for the IP address and port as text boxes. Other buttons include allowing the user to connect/disconnect, start/stop listening for incoming connections and a start simulation button.

The second section is the log view. It was important when designing this stage to make sure that any piece of data that would possibly come through would be easily shown without being cut off. It was also important to make sure there was enough logging data to make any messages meaningful. This required adding sections such as Remote Address and Remote Port, these give the user some context to the data that is currently coming through the connection.

The third section consists of a series of custom buttons arranged in a grid format. These buttons handle the simulation aspect of the project. When a button is hovered over with a mouse, the button will change colour to black initially then will dim incrementally over several seconds before fading back to default. This gives the buttons a delay effect which can be used to measure the extent of the network's degradation. The start simulation button is used in conjunction with the button-grid to generate a random wave pattern, moving from left to right, allowing the user to automate the button hover process.

4.2.3 Tool UI Design

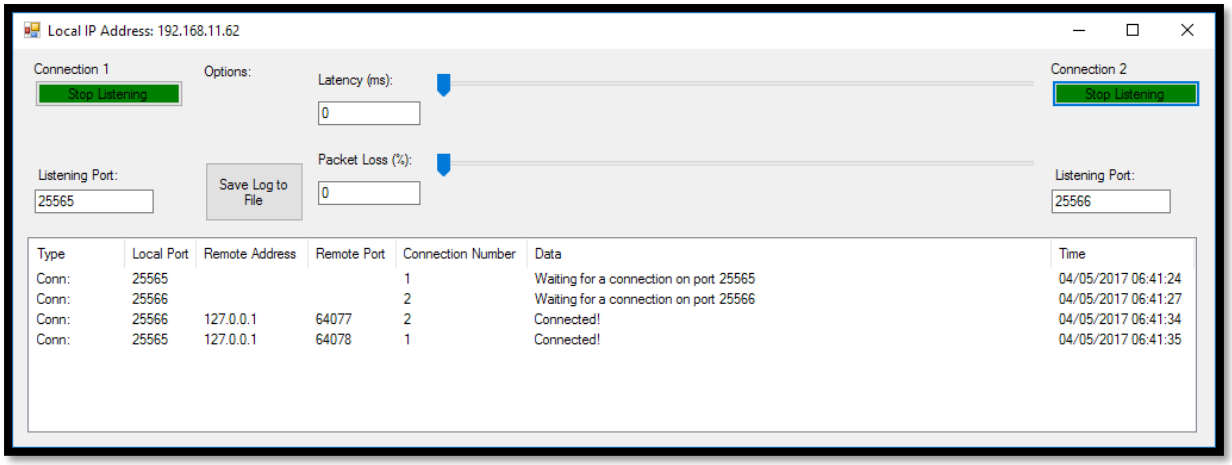


Figure 5. The tool

The tool's design is very like that of the client's. It was important to make them look similar and behave similarly so that users can feel comfortable using any part of the software.

The tool has been designed in 3 main sections. The first section is where the user controls the incoming connections to the tool, there is a listening toggle button and a port number. The user is easily able to distinguish whether the tool is currently listening or not by the colour of button. Green is listening, red is not. It is important to have these visual aids as it allows the user to see what is going on at a glance.

The second section is the options section. Here the user will be able to control and manipulate the connection between the two clients. Trackbars are used to allow the user to easily change the value of the latency and packet loss values while keeping the values within a predefined range. The ranges are included to make sure the software can handle the values correctly. The range for latency is 0 seconds to 10 seconds. The range for packet loss is 0% to 100%.

The first implementation of latency came in the form of a queue. Originally the program would only send a message after the latency time had passed. Reviewing the code found this to be the wrong implementation of latency as some packets had to wait a long time before being sent. The code was then changed to add delay to each packet rather than a queue. This then resolved the issue

Finally, the last section of the tool's design is the log view. This is very like that of the client's however there is an extra "Connection Number" column. This extra column is useful as it allows the user to see where the data is flowing from and where to.

4.2.4 Data Design

When designing how the clients and tool will handle incoming data, A coded or indexed design was decided. If any incoming packets started with the string "0xx" then it was decided these packets would be used for commands sent between each connection. For example, "0xxClientClosing" would be recognized as a command sent from a client. The tool would notice this and perform actions based on the data's value. If the message didn't start with "0xx" then it was assumed that the data being sent was not a command issued to client or tool, therefore the message was sent on to the other connection after adding the necessary packet loss and latency.

For the demonstration to work, the data being sent between clients needed to be able to be translated back into meaningful data. Therefore, it was decided that any data being sent across that was involved with the button-grid demo needed to have its own prefix. “1xx” was the designation used for these messages.

An example packet from the demo is “1xx,8,4”. After splitting the data into several strings using the ‘,’ separator the receiving client can then determine to activate the button at the given co-ordinates. In this case, the 8th button along and 4th button down would change colour to black and then proceed to fade out to default after a few seconds.

5. Evaluation

5.1 Objectives Evaluations

***"Objective 1**: Design a client that with basic networking functionality*

- *This will be basic functionality for the project, for the whole system to function, there will have to be a client capable of sending and receiving packets to and from another client as well as receiving packets from a server."*

Objective 1 was completed to a good standard. The client can connect to another client and the tool, it can also send and receive data. The client meets all of the objective needs.

***Objective 2**: Design a tool to connect to two clients simultaneously*

- *To develop further steps, A tool will need to be made to forward messages from one client to another.*

Objective 2 was completed to good standard. The tool created is able to handle two incoming connections and it is able to forward messages from one client to another.

***Objective 3**: Add latency manipulation to the network connection.*

- *This would be the first objective to allow the main aspect of the tool to function, the system should allow for a range of values to be added to the latency of the connection.*
- *This goal will be measured by testing whether latency has been added to the connection.*

Objective 3 was completed to good standard, it is one of the major objectives in which the project is focused on. The tool is required to add latency after it receives a packet from one client, instead of passing on the data straight away, the tool instead waits a certain amount of time depending on what the user selects.

***Objective 4**: Add packet loss manipulation to the network connection.*

- *This would be a follow-up objective to the latency. The system should allow for a percentage packet loss value to be added to the connection to cause instability.*
- *This goal will be measured by testing whether packet loss has been added to the connection.*

Objective 4 was completed to a good standard. Packet loss's implementation is very like latency's. After receiving a message from one client, the tool then generates a random number between 0 and 100, if the random value is less than the user's chosen packet loss value, then the message is not sent.

***Objective 5**: Add data logging to allow further analysis of the data post-testing.*

- *This allows me to export the data once a test has been completed, to allow for extra analysis even once the program has been shut down. This will be done by exporting the data to a file.*

Objective 5 was completed to a good standard. Each application has a logger class that keeps track of all messages that go through the log window. Once the user decides to click save, the logger then prompts the user to choose a file name and location, the program then proceeds to save all data to the file. The file also has column headers and file information to inform the user what each column of data represents as well as when the test was performed. The data logging has been implemented to a good standard, however it could be improved upon by adding filters to the data collection as well as adding contextual information such as current latency value and current packet loss value.

***Objective 6**: Build an interactive demo to show the network's performance in real time*

- *A demo will show how the network connection is currently performing. The demo will be able to show to the user how the current degradation settings are currently effecting the connection.*
- *Testing this demo involves having completed steps 1-4 to give the demo some real data and to then test whether the results are as expected.*

Objective 6 was completed with great success. The method used for demonstrating the network's current connection integrity was a grid of custom buttons. These buttons work very well to effectively communicate how the tool's current packet loss and latency values affect the connection between the clients. Another solution that was tried was to implement a simple game of Pong! This idea was scrapped soon into development of the demo as it was deemed to be too complex for the scope of the project. Too much time was spent on developing the game rather than developing the application itself, therefore it made more sense to make a mini display like demo in the form of a grid of custom buttons. Initially this method of demonstration didn't seem to work very well. However, with adjustments to button size, row length and column length, the button grid ended up working very effectively.

5.2 Objectives Overview

Overall the project has achieved the goals it set out to meet. While the application could have been developed further any future developments would be considered quite complex, for example features such as connection interception and packet sniffing would have been great potential additions to the project. However, developing these features was deemed too complex and would have stretched the time scale of the project significantly, potentially causing some other features to be neglected.

In terms of future improvement, the software could be enhanced by adding features previously described such as packet sniffing, connection intercepting. Other enhancements could be made in the current software. The user interface in this application was very basic and could be improved on to be made more intuitive. The logging of data could also be improved with some export to graph functions, allowing the user to more easily review how the tool's settings affected the connection between the two clients.

5.3 Real World Uses

The software at its current stage is a great tool for developers all around the world to use. There is no complex set up procedures to be made in order to get the software working, therefore, the software is available to be used by anyone. The tool itself can sit in-between any two applications or devices and accurately manipulate the connection. This would be great for IT services technicians that have problems with certain devices or applications where the networks being used are performing as predicted. Giving the user more

debugging steps allows them to find the problem and attempt a solution more quickly and easily.

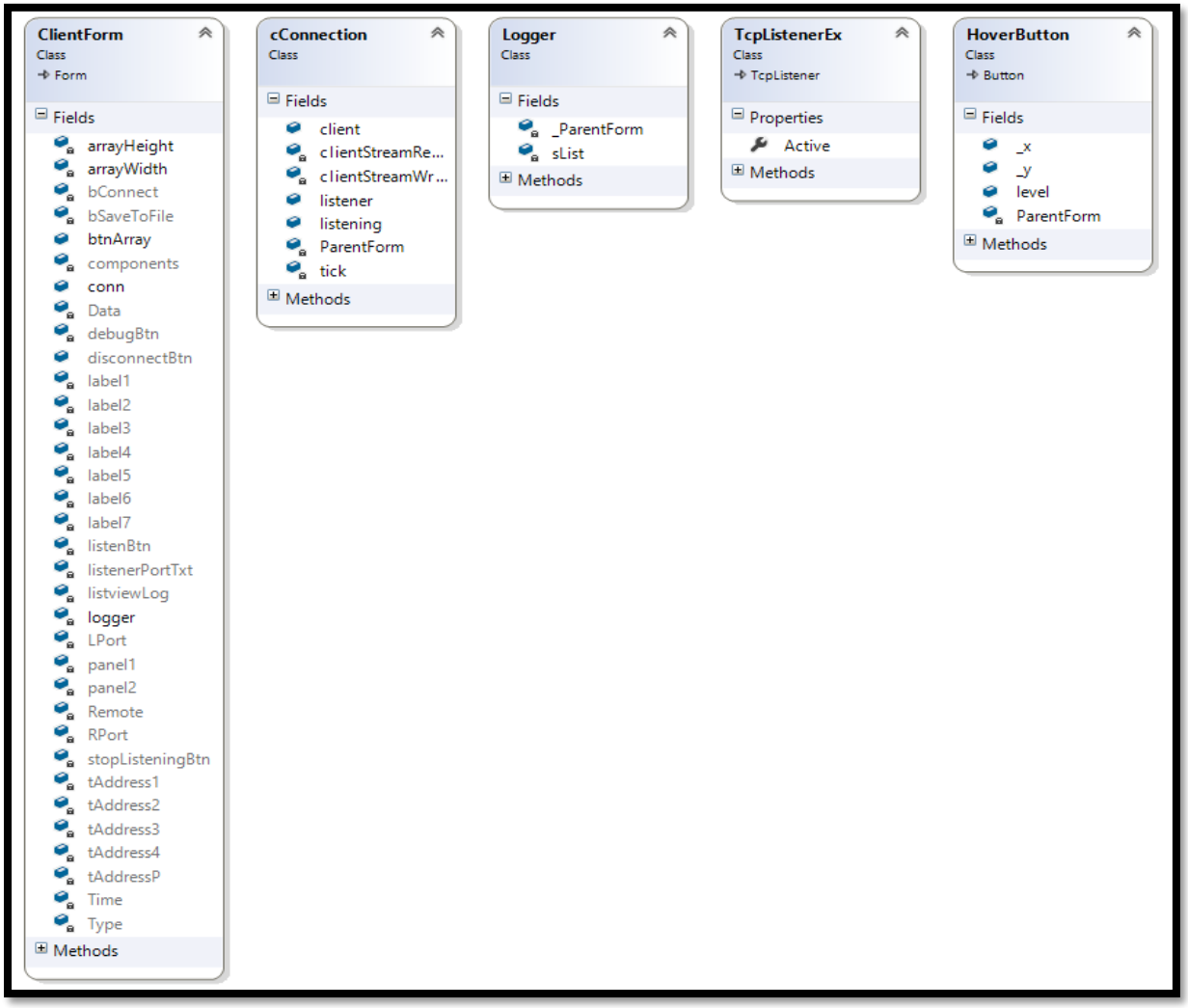
6. Conclusion

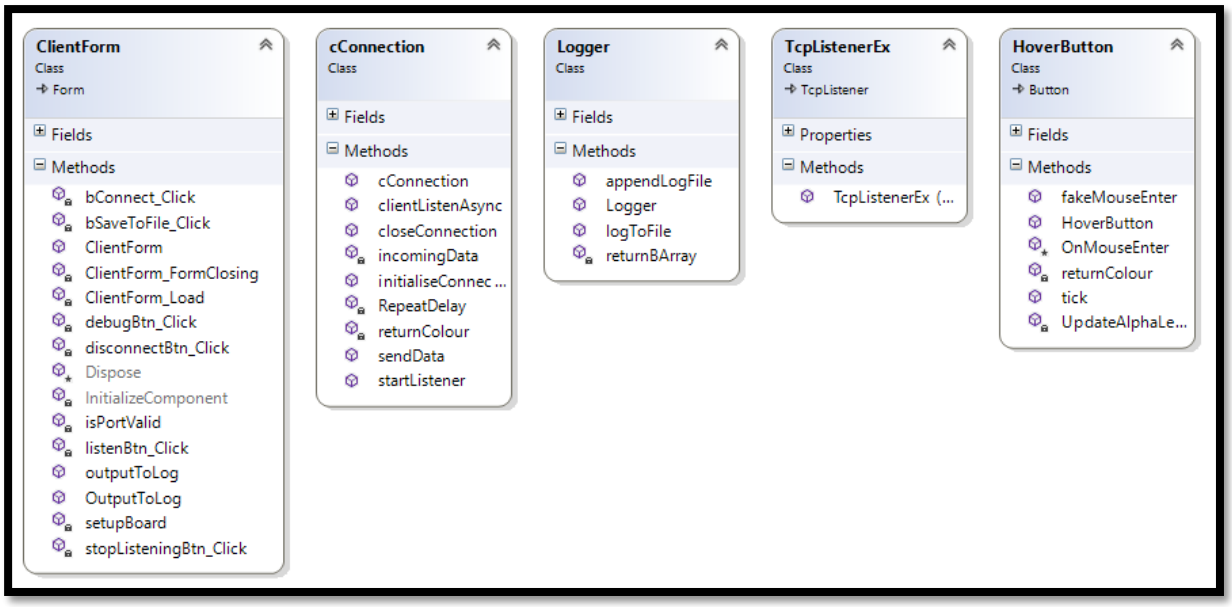
The project can be considered as complete considering that all the goals have been successfully implemented, while there are still some improvements to be made the project is complete. The software has delivered on all the original goals that it set out to achieve with no currently known bugs.

For potential end users, this software provides a great tool to be able to simulate networked connections from anywhere around the world. Simulating a connection from London to New York would be straight forward for the user to achieve, a latency of 90ms and packet loss of 1% would accurately represent real-world speeds.

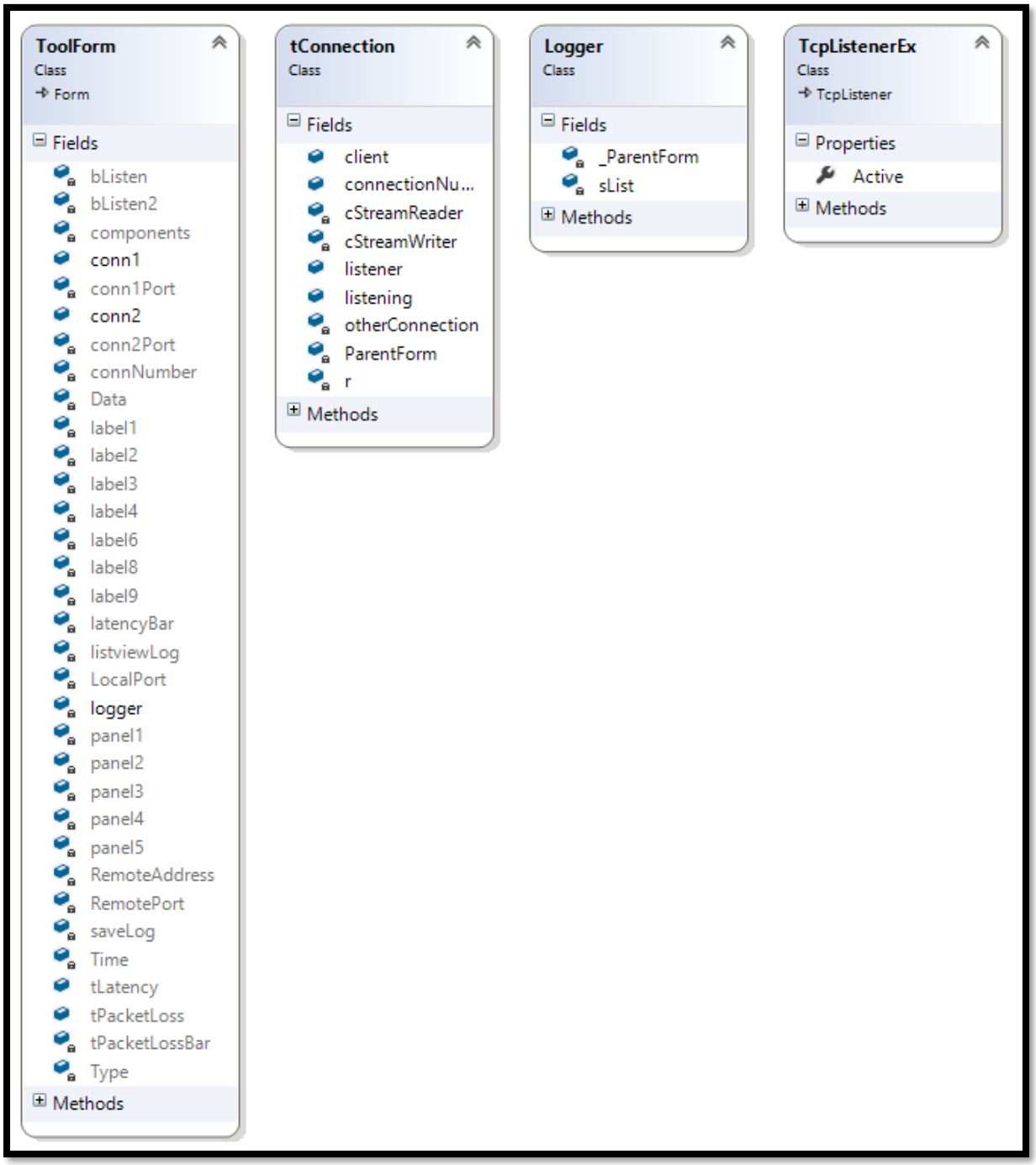
Overall, all the algorithms used are reliable and don’t need to be changed or improved. The greatest improvements in the software could be made through improving the user interface design and potentially adding further features to the software.

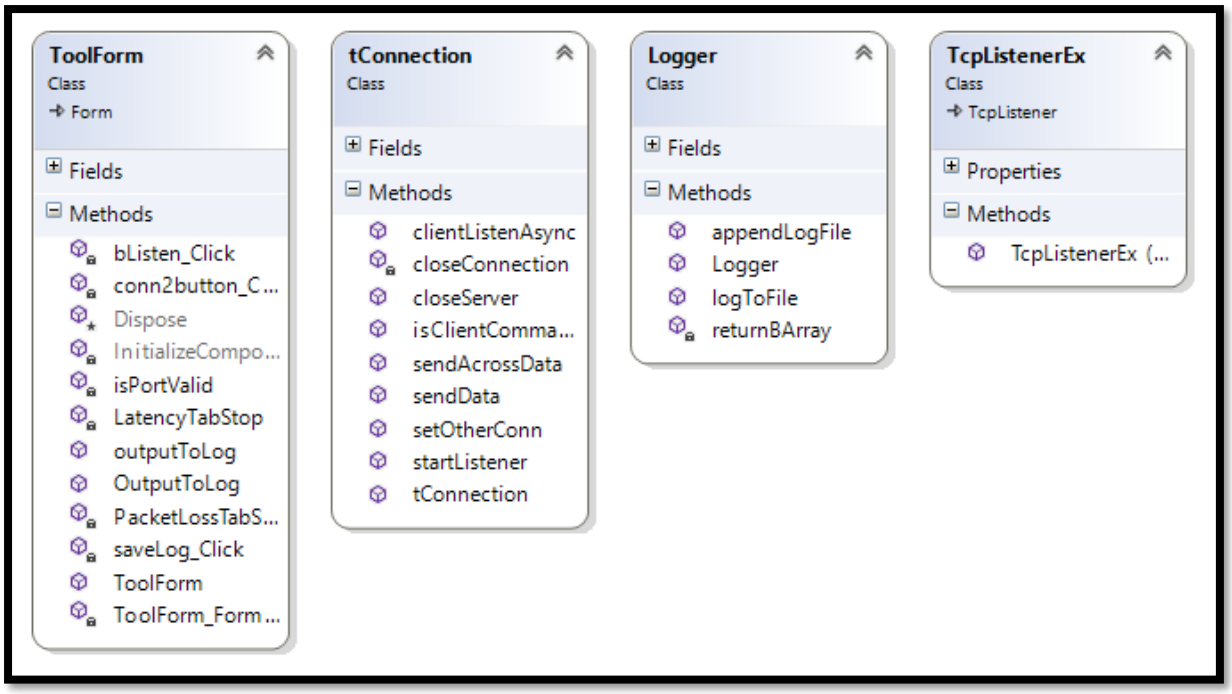
Appendix A: Client Class Diagrams




ClientForm
Class
Form
Fields
Methods
bConnect_Click
bSaveToFile_Click
ClientForm
ClientForm_FormClosing
ClientForm_Load
debugBtn_Click
disconnectBtn_Click
Dispose
InitializeComponent
isPortValid
listenBtn_Click
outputToLog
OutputToLog
setupBoard
stopListeningBtn_Click
cConnection
Class
Fields
Methods
cConnection
clientListenAsync
closeConnection
incomingData
initialiseConnec ...
RepeatDelay
returnColour
sendData
startListener
Logger
Class
Fields
Methods
appendLogFile
Logger
logToFile
returnBArray
TcpListenerEx
Class
TcpListener
Properties
Methods
TcpListenerEx (...
HoverButton
Class
Button
Fields
Methods
fakeMouseEnter
HoverButton
OnMouseEnter
returnColour
tick
UpdateAlphaLe...

Appendix B: Tool Class Diagrams





Appendix C: Code

Client – “cConnection.cs”:
Code used to sort incoming packets:

```
void incomingData(string data)
{
    switch (data)
    {
        case "0xxToolClosing":
        case "0xxToolStopListening":
        case "0xxClientClosing":
            closeConnection();
            break;
        default:
            break;
    }

    if (data.StartsWith("1xx"))
    {
        string[] _data = data.Split(',');
        int[] _vals = new int[2];
        for (int i = 0; i < 2; i++)
        {
            int.TryParse(_data[i + 1], out _vals[i]);
        }
        ParentForm.btnArray[_vals[0], _vals[1]].BackColor = returnColour(6);
        ParentForm.btnArray[_vals[0], _vals[1]].level = 6;
        ParentForm.btnArray[_vals[0], _vals[1]].tick();
    }
}
```

Code used to send data from one client to another:

```
public void sendData(string msg)
{
    try
    {
        if (client.Connected)
        {
            NetworkStream nStream = client.GetStream();
            clientStreamReader = new StreamReader(nStream);
            clientStreamWriter = new StreamWriter(nStream);

            clientStreamWriter.WriteLine(msg);

            string address = ((IPEndPoint)client.Client.RemoteEndPoint).Address.ToString();
            string port = ((IPEndPoint)client.Client.RemoteEndPoint).Port.ToString();
            ParentForm.OutputToLog("Sent:", "", address, port, msg);
            clientStreamWriter.Flush();
            Application.DoEvents();
        }
        else{
            if(tick == 0)
            {
                ParentForm.outputToLog("Error:", "Could not send data, Client not connected.");
                RepeatDelay();
            }
        }
    }
    catch
    {
        ParentForm.outputToLog("Error:", "Could not connect to server");
        return;
    }
}
```

Code used to decide which colour value should be used:

```
private Color returnColour(int level)
{
    switch (level)
    {
        case 0:
            return SystemColors.Control;
        case 1:
            return Color.FromArgb(255 / 5, 0, 0, 0);
        case 2:
            return Color.FromArgb(255 / 4, 0, 0, 0);
        case 3:
            return Color.FromArgb(255 / 3, 0, 0, 0);
        case 4:
            return Color.FromArgb(255 / 2, 0, 0, 0);
        case 5:
            return Color.FromArgb(255 / 1, 0, 0, 0);
        default:
            return SystemColors.Control;
    }
}
```

Waiting asynchronously for a connection:

```
//Asynchronously wait for a connection
if (!client.Connected)
{
    ParentForm.OutputToLog("Conn:", localPort.ToString(), "", "", "Waiting for a connection on port " + localPort.ToString());
    client = await listener.AcceptTcpClientAsync();
    string[] parts = client.Client.RemoteEndPoint.ToString().Split(':');
    remoteAddr = parts[0];
    remotePort = parts[1];
    ParentForm.OutputToLog("Conn:", localPort.ToString(), remoteAddr, remotePort, "Connected!");

    NetworkStream nStream = client.GetStream();

    clientStreamWriter = new StreamWriter(nStream);
    clientStreamReader = new StreamReader(nStream);
    ParentForm.disconnectBtn.Enabled = true;
}
```

Waiting asynchronously for incoming messages:

```
try
{
    string incoming = await clientStreamReader.ReadLineAsync();
    string temp = incoming;

    string[] parts = client.Client.RemoteEndPoint.ToString().Split(':');
    remoteAddr = parts[0];
    remotePort = parts[1];

    if (temp != null)
    {
        ParentForm.OutputToLog("Received:", localPort.ToString(), remoteAddr, remotePort, temp);
        incomingData(temp);
    }
    else
    {
        //bad state
    }
}
catch (Exception ex)
{
    Console.WriteLine("Error: " + ex);
    //Tool is hanging after client disconnects
}
```

Connect to another client:

```
public void initialiseConnection(IPAddress address, int port)
{
    //Initialise a new client
    client = new TcpClient();

    ParentForm.OutputToLog("Conn:", "", address.ToString(), port.ToString(), "Attempting connection to tool...");
    try
    {
        client.Connect(address, port);
        ParentForm.OutputToLog("Conn:", "", address.ToString(), port.ToString(), "Connected!");
        ParentForm.disconnectBtn.Enabled = true;
        NetworkStream nStream = client.GetStream();
        clientStreamWriter = new StreamWriter(nStream);
        clientStreamReader = new StreamReader(nStream);
        listening = true;
        clientListenAsync(port);
    }
    catch
    {
        ParentForm.OutputToLog("Conn:", "", address.ToString(), port.ToString(), "Could not connect to tool, Server is probably not listening");
        return;
    }
}
```

Wait for incoming connections:

```
public void startListener(int port)
{
    //default port is 25565 as per the form design
    listener = new TcpListenerEx(IPAddress.Any, port);
    listener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.KeepAlive, true);
    listener.Start();
    client = new TcpClient();
    listening = true;
    clientListenAsync(port);
}
```

Client – “HoverButton.cs” – Most of the code relevant to the button

```
public class HoverButton : Button
{
    public int level = 0;
    ClientForm ParentForm;
    public int _x;
    public int _y;
    /// <summary>
    ///
    ///
    /// </summary>
    /// <param name="size">Width of button</param>
    /// <param name="x">co-ords x</param>
    /// <param name="y">co-ords y</param>
    /// <param name="p">Parent form</param>
    /// <param name="a">Array width value of this button</param>
    /// <param name="b">Array height value of this button</param>
    public HoverButton(int size, int x, int y, ClientForm p, int a, int b)
    {
        ParentForm = p;
        this.Width = size;
        this.Height = size;
        this.Location = new Point(x, y);
        base.BackColor = returnColour();
        this._x = a;
        this._y = b;
    }

    public void fakeMouseEnter()
    {
        base.BackColor = Color.FromArgb(255, 0, 0, 0);
        level = 6;
        ParentForm.conn.sendData("1xx" + "," + _x + "," + _y);
        tick();
    }

    protected override void OnMouseEnter(EventArgs e)
    {
        base.OnMouseEnter(e);
        base.BackColor = Color.FromArgb(255, 0, 0, 0);
        level = 6;
        ParentForm.conn.sendData("1xx" + "," + _x + "," + _y);
        tick();
    }
```

Client – “Logger.cs”

```
class Logger
{
    List<string> sList = new List<string>();
    ClientForm _ParentForm;

    public Logger(ClientForm form)
    {
        _ParentForm = form;
        //File Header Text
        string entryText = "Test started at: " + DateTime.Now.ToString("yyyy/MM/dd HH:mm");
        string template = "Type, Local Port, Remote Address, Remote Port, Data, Time";
        sList.Add(entryText);
        sList.Add(template);
    }
    public void appendLogFile(string msg)...

    /// <summary> Save the file and output all log data to it
    public void logToFile()
    {
        //Configure defaults for the file to be saved
        Stream myStream;
        SaveFileDialog saveBox = new SaveFileDialog();
        saveBox.InitialDirectory = Directory.GetCurrentDirectory();
        saveBox.FileName = "Latency-Tool-Log--" + DateTime.Now.ToString("yyyy-MM-dd--HH.mm");
        saveBox.Filter = "All files (*.*)|*.*|txt files (*.txt)|*.txt";
        saveBox.FilterIndex = 2;
        saveBox.RestoreDirectory = true;

        //Open the save file dialog and output the log data to it
        if (saveBox.ShowDialog() == DialogResult.OK){
            if ((myStream = saveBox.OpenFile()) != null){
                Byte[] data = returnBArray();
                myStream.Write(data, 0, data.Length);
                myStream.Close();
            }
        }
        //Ask the user whether they want to open the file (Yes / No) Popup
        string message = "Do you want to open the file?";
        string caption = "Open file?";
        MessageBoxButtons buttons = MessageBoxButtons.YesNo;
        DialogResult result = MessageBox.Show(message, caption, buttons);

        if (result == System.Windows.Forms.DialogResult.Yes){
            System.Diagnostics.Process.Start(saveBox.FileName);
        }
    }

    /// <summary> Returns a byte array of the current log list
    Byte[] returnBArray()...
}
```

Client – “ClientForm.cs” – Code to set up an array of HoverButtons on the form

```
void setupBoard()
{
    arrayWidth = 19;
    arrayHeight = 7;
    int size = 20;
    btnArray = new HoverButton[arrayWidth, arrayHeight];
    for (int i = 0; i < arrayWidth; i++)
    {
        for (int j = 0; j < arrayHeight; j++)
        {
            //618, 12
            int x = 618 + (size * i);
            int y = 12 + (size * j);
            btnArray[i, j] = new HoverButton(size, x, y, this, i, j);
            this.Controls.Add(btnArray[i, j]);
            btnArray[i, j].Show();
        }
    }
}
```

Code to automate the demonstration grid

```
private async void debugBtn_Click(object sender, EventArgs e)
{
    Random r = new Random();
    int counter = 0;
    while (true)
    {
        await Task.Delay(130);
        btnArray[counter, r.Next(0, 6)].fakeMouseEnter();
        counter++;
        if (counter >= arrayWidth)
            counter = 0;
    }
}
```

Code to check validity of values before attempting a connection

```
private void bConnect_Click(object sender, EventArgs e)
{
    //Try to parse the IPAddress from the text boxes on the form
    IPAddress address;
    bool resHost = IPAddress.TryParse(tAddress1.Text + "." + tAddress2.Text + "." + tAddress3.Text + "." + tAddress4.Text, out address);
    if (resHost == false)
    {
        outputToLog("Error:", "Unable to parse IP Address, Make sure it is a valid Address");
    }

    //Try to parse the port from the text box on the form
    int port;
    bool resPort = Int32.TryParse(tAddressP.Text, out port);
    if (resPort == false)
    {
        outputToLog("Error:", "Unable to parse port number, Make sure it is a valid port number");
    }

    //If either of the values did not parse, do not run the SendData function. Output to the log showing error
    if (resHost == false || resPort == false)
    {
        return;
    }
    else
    {
        if (!conn.client.Client.Connected)
        {
            conn.initialiseConnection(address, port);
        }
    }
}
```

Tool – “tConnection.cs”
Code to pass along data to another connection

```
public async void sendAcrossData(string msg)
{
    int time;
    int.TryParse(ParentForm.tLatency.Text, out time);
    await Task.Delay(time);

    int packetLoss;
    int.TryParse(ParentForm.tPacketLoss.Text, out packetLoss);
    if(packetLoss > 100)
    {
        packetLoss = 100;
    }
    int value = r.Next(0, 100);
    if(value < packetLoss)
    {
        //Don't send
        ParentForm.outputToLog("Packet Loss:", "Did not send message");
    }
    else
    {
        //Do send
        otherConnection.sendData(msg);
    }
}
```

Code to toggle the Listening state of a connection

```
private void bListen_Click(object sender, EventArgs e)
{
    bListen.Text = bListen.Text == "Listen" ? "Stop Listening" : "Listen";
    bListen.BackColor = bListen.BackColor == Color.Green ? Color.Red : Color.Green;
    if (bListen.BackColor == Color.Green)
    {
        bool portIsValid = isPortValid(conn1Port.Text);
        if (portIsValid == true)
        {
            int portValue;
            bool resPort = Int32.TryParse(conn1Port.Text, out portValue);
            conn1.startListener(portValue);
        }
    }
    else
    {
        conn1.sendData("0xxToolStopListening");
        conn1.closeServer();
    }
}
```

Appendix D: Original Task Plan

#	Task Name	Description	Duration (days)
1	Design program tree	This involves splitting the project into sub tasks with further sub tasks.	14
2	Write pseudocode	For each baseline sub task, Pseudocode should be written to understand what needs to be done.	14
3	Generate user interface	Design the look of the application with all of the buttons and labels needed for the program to run.	14
4	Implement client	Complete functionality to allow connections to be opened with other sources	21
5	Implement server	Complete functionality to allow the sending of lots of data to another source.	21
6	Interim report	Write the interim report deliverable	21
7	Implement latency tool	Manipulate outgoing/incoming connections by varying the latency	14
8	Implement packet loss tool	Manipulate outgoing/incoming connections by varying the packet loss.	14
9	Implement data capture	Allow each packet of data to be saved and allow the user to view the data	14
10	Implement data storage	Allow the user to save the data transmitted from a session to a file. This will include contextual data such as packet loss / latency over time.	14
11	Input validation	Test the program to ensure all possible values able to be input by the user work correctly as intended.	14
12	Data testing	Test the program with varying latency and packet loss values and record results of the connection.	14
13	Finalize report	Write the final report, discuss all aspects of creating the program, discuss results.	21

Appendix E: Original Time Plan

		University Calendar Weeks																	
#	**Task Name**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	Design program structure tree					D													D
2	Write pseudocode							D											
3	Generate user interface									D									
4	Implement client												D						
5	Implement server															D			
6	Interim report																		D

Continued..

		University Calendar Weeks														
#	**Task Name**	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
7	Implement latency tool		D													
8	Implement packet loss tool				D											
9	Implement data capture						D									
10	Implement data storage								D							
11	Input validation										D					
12	Data testing												D			
13	Finalize report															D

References

[1] *Bandwidthcontroller.com. (2017).* Traffic Shaper XP - Freeware Bandwidth Limiter. *[online] Available at: http://bandwidthcontroller.com/trafficShaperXp.html [Accessed 3 May 2017].*

[2] *Solutions, A. (2017). IP Latency Statistics. [online] Verizon Enterprise Solutions. Available at: http://www.verizonenterprise.com/about/network/latency/ [Accessed 4 May 2017].*

[3] *User, S. (2017).* TMNetSim: Quick and Easy Network Simulation Tool. *[online] Tmurgent.com. Available at: http://www.tmurgent.com/appv/index.php/en/87-tools/performancetools/177-tmnetsim-quick-and-easy-network-simulation-tool [Accessed 3 May 2017].*

[4] *Wiki.linuxfoundation.org. (2017).* Networking: netem [Linux Foundation Wiki]. *[online] Available at: https://wiki.linuxfoundation.org/networking/netem [Accessed 3 May 2017].*

[5] *Technet.microsoft.com. (2017).* TCPView for Windows. *[online] Available at: https://technet.microsoft.com/en-us/sysinternals/tcpview.aspx [Accessed 3 May 2017].*

[6] *Msdn.microsoft.com. (2017). Socket Class (System.Net.Sockets). [online] Available at: https://msdn.microsoft.com/en-us/library/system.net.sockets.socket(v=vs.110).aspx [Accessed 3 May 2017].*

[7] *Msdn.microsoft.com. (2017). TcpClient Class (System.Net.Sockets). [online] Available at: https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.110).aspx [Accessed 3 May 2017].*