ADVANCED TELECOMS CS 3031 Assignment 1 2019

FEBRUARY 26TH

Prapti Setty
Student Number: 16336726

Implementing a Web Proxy Server

A Web proxy is a local server, which fetches items from the Web on behalf of a Web client instead of the client fetching them directly. This allows for caching of pages and access control.

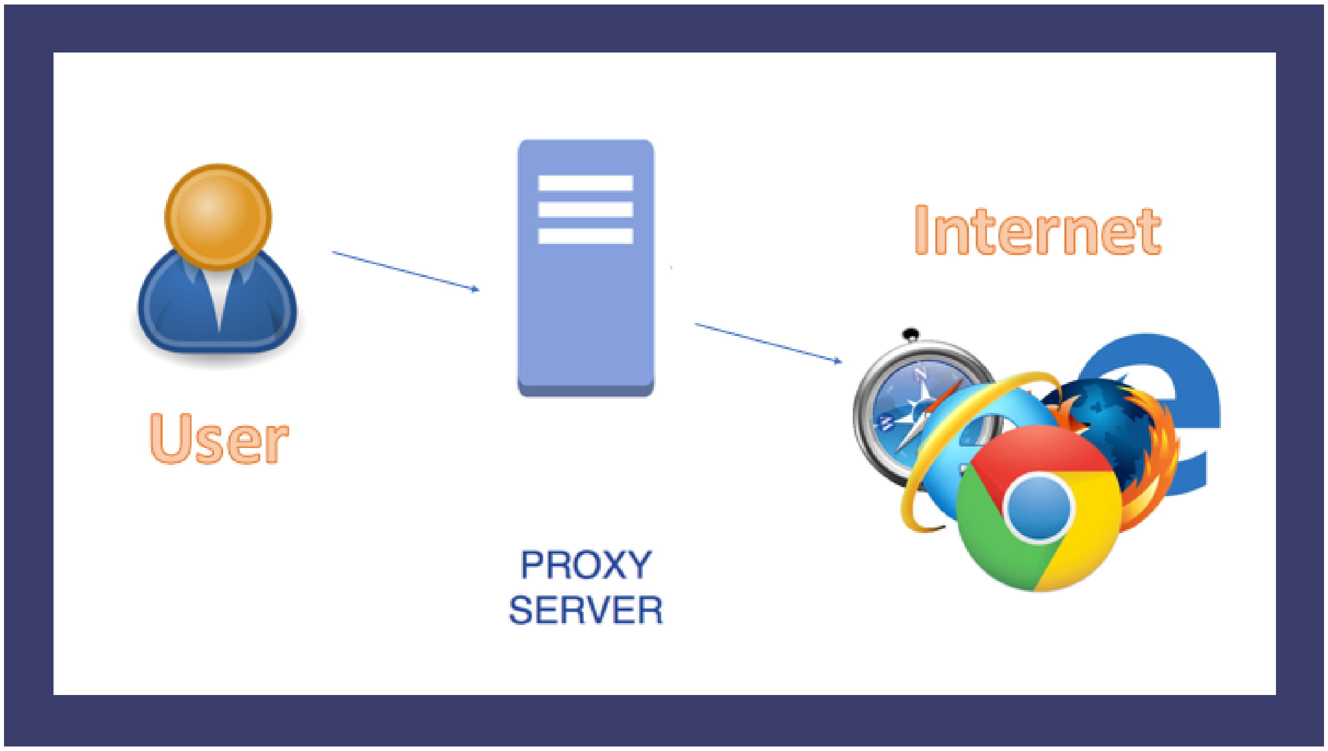
The program should be able to:

1. Respond to HTTP & HTTPS requests and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.
2. Handle WebSocket connections.
3. Dynamically block selected URLs via the management console.
4. Efficiently cache requests locally and thus save bandwidth. You must gather timing and bandwidth data to prove the efficiency of your proxy.
5. Handle multiple requests simultaneously by implementing a threaded server.

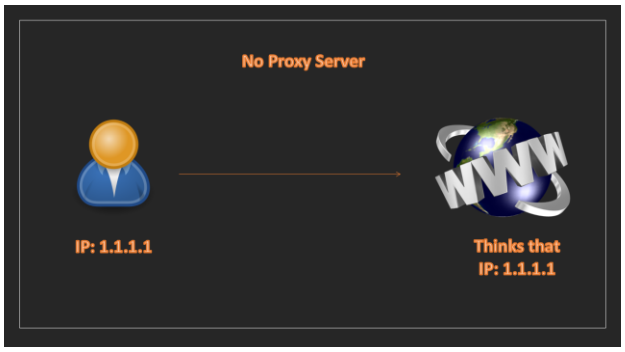
You should provide a high-level description of the protocol design and implementation. A listing of the code should also be provided along with meaningful comments. You are required to submit a single PDF file containing the documentation and code using the Turnitin system.

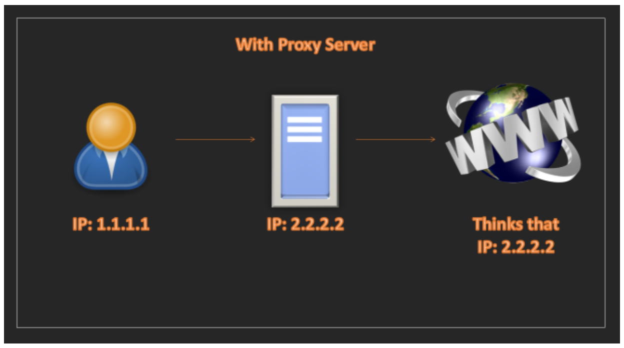
How I implemented this project:
I began by understanding exactly what a proxy server is and what it does. I created some diagrams for myself to help myself with the process of understanding and also as a means to help me create/get an idea of what system architecture for the project should look like.

On a very high level:



A proxy server is basically a computer on the internet with its own IP address that your computer knows. When you send a web request, your request goes to the proxy server first. The proxy server then makes your web request on your behalf, collects the response from the web server, and forwards you the web page data so you can see the page in your browser.



VS


As we can see from the comparison diagram above, a Proxy Server intercepts requests made to the internet and can change the IP of the user.

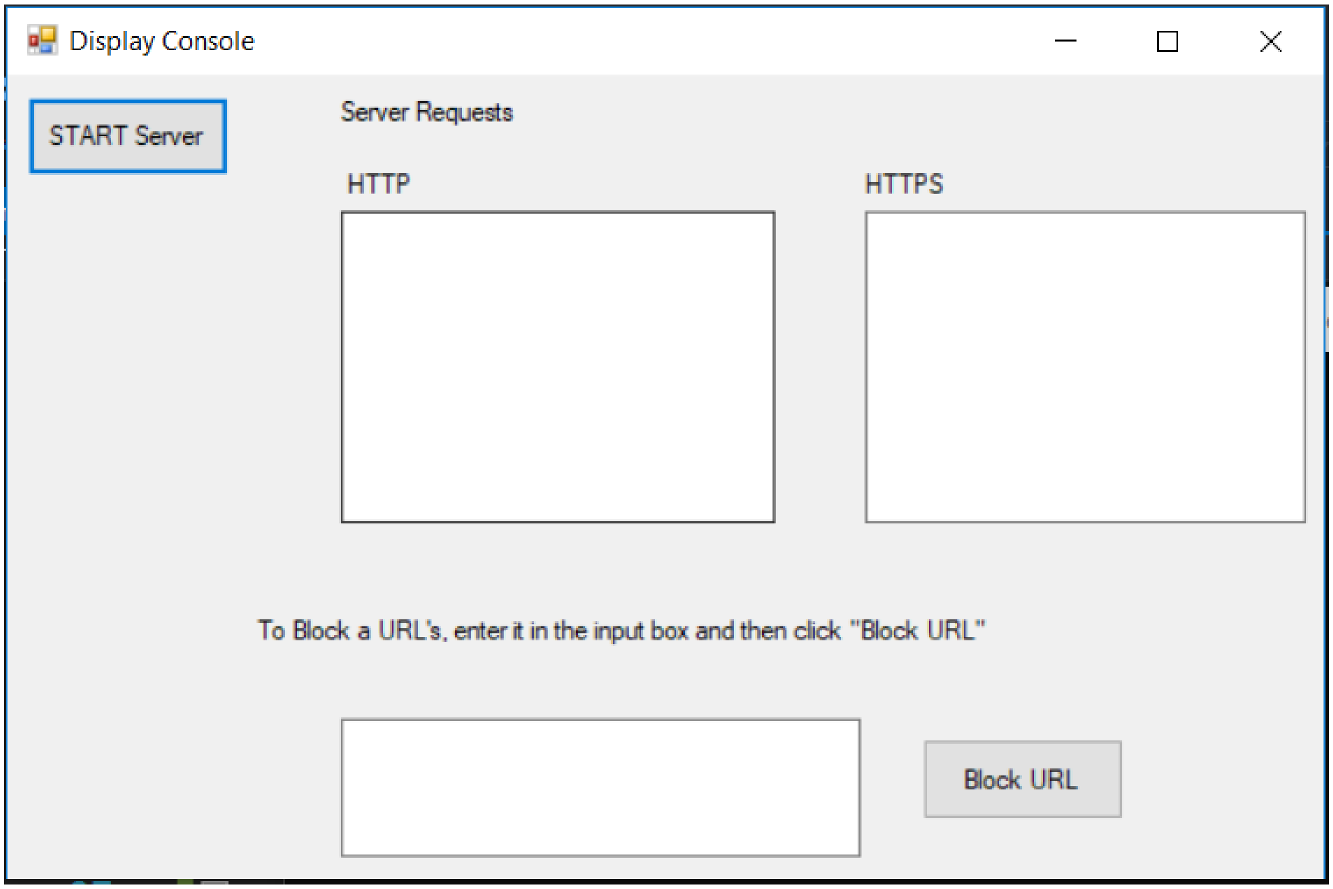
For the purpose of this project we had to create a very simple server that intercepts at HTTP and HTTPS requests.
We need to first understand the difference between these.

HTTP stands for HyperText Transfer Protocol, the S in HTTPS stands for secure.

The webpage is made secure by requiring a connection established message sent back. The webpage is encrypted in HTTPS.

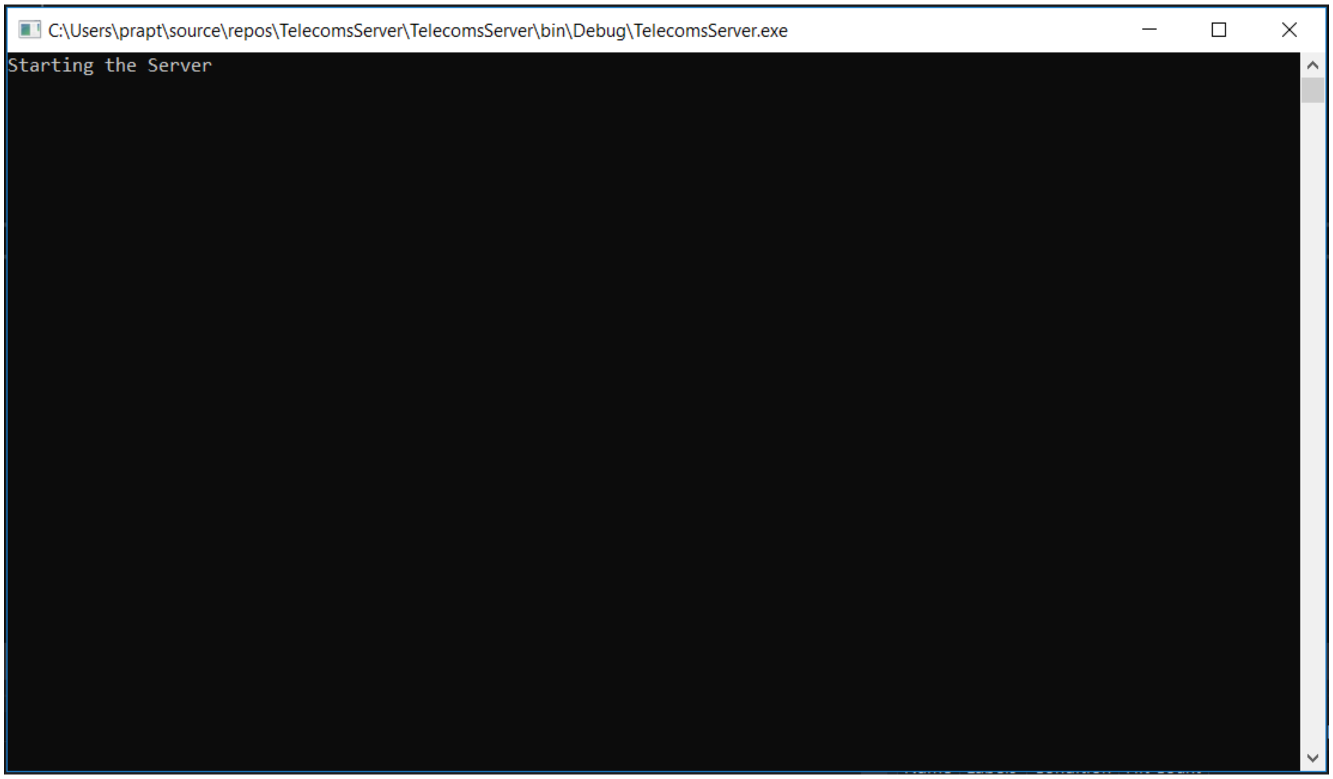
After understanding this I went on to examine the requirements and began implementing them:

When the project is Run, we expect a Display console to open with two buttons (Start Server and Block URL) and three text boxes (one to display HTTP connections, one to display HTTPS connections and one to allow us to block URLs).
The Block URL box is the only one that takes input. The other two can only display outputs.
This display console is a windows form, created in C# itself.



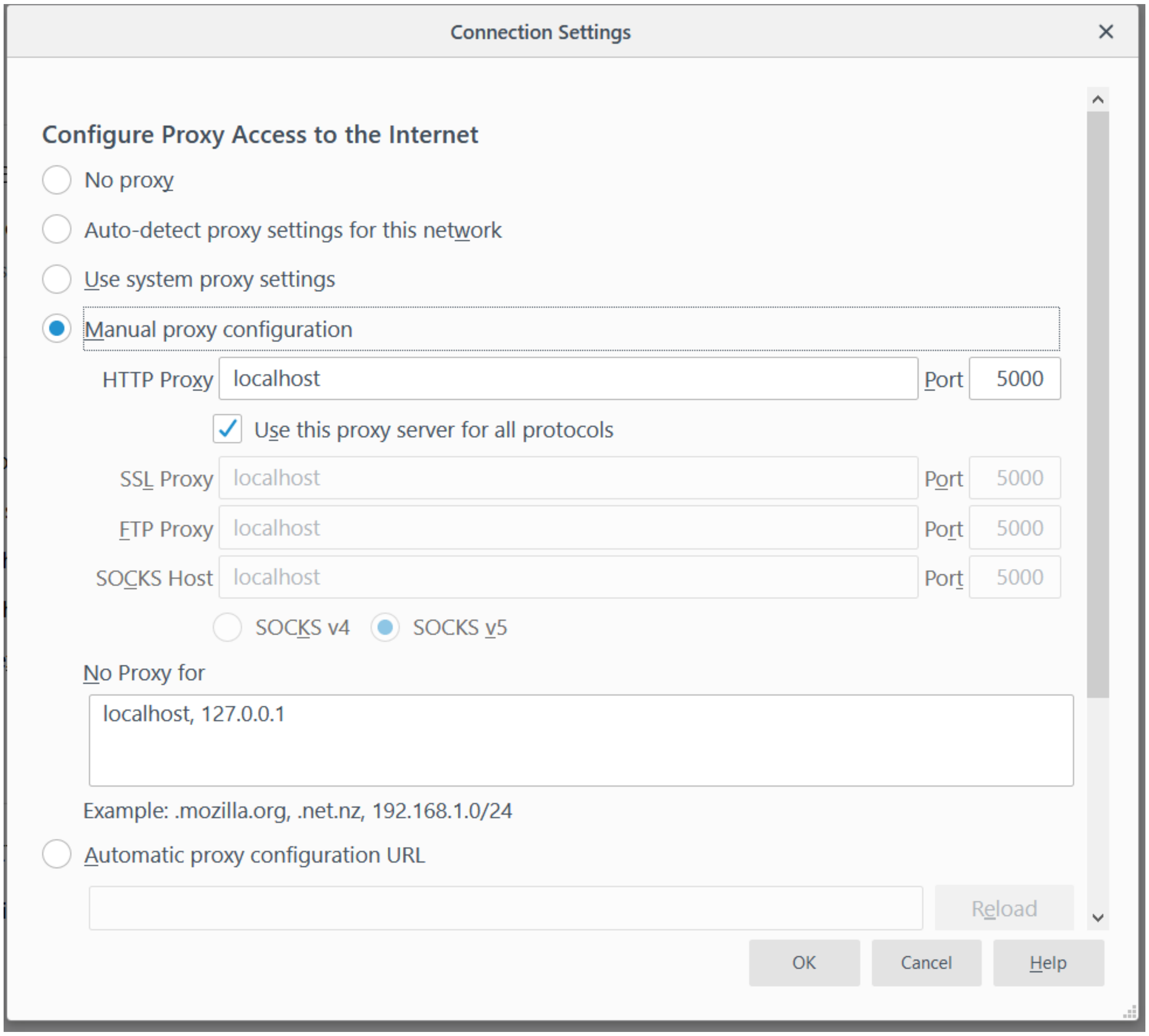
To Block a URL's, enter it in the input box and then click "Block URL"

To start our server, we must click the “START Server” option. Once we click “START Server” button, a new console log should open as shown below.
This is running in a separate thread, thus allowing us to access both consoles at the same time, i.e. type into the Block URL text box and actually block URLs.
This new console window logs all the hosts and responses, as opposed to short URL and Host connections to show what is happening.


C:\Users\prapt\source\repos\TelecomsServer\TelecomsServer\bin\Debug\TelecomsServer.exe

```
Starting the Server
```

These are the settings I chose to set for my browser to be able to complete these:


Connection Settings

Configure Proxy Access to the Internet

( ) No proxy

( ) Auto-detect proxy settings for this network

( ) Use system proxy settings

(●) Manual proxy configuration

HTTP Proxy: localhost Port: 5000

✓ Use this proxy server for all protocols

SSL Proxy: localhost Port: 5000

FTP Proxy: localhost Port: 5000

SOCKS Host: localhost Port: 5000

( ) SOCKS v4 (●) SOCKS v5

No Proxy for: localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

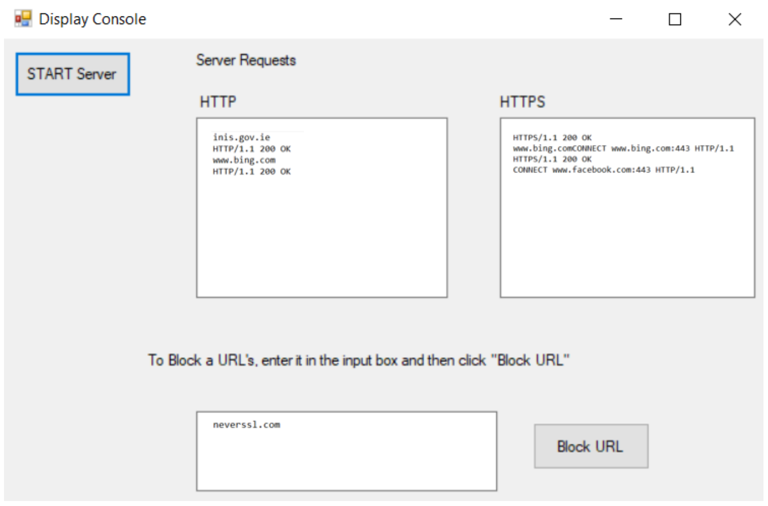
( ) Automatic proxy configuration URL

HTTP requests come in over Port 80 and redirect to Port 5000.
HTTPS requests come in over Port 443 and redirect to Port 5000.

To start,

Respond to HTTP & HTTPS requests and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.

When we start making requests, the Display console should start logging the results as follows:



```
inis.gov.ie
HTTP/1.1 200 OK
www.bing.com
HTTP/1.1 200 OK
```

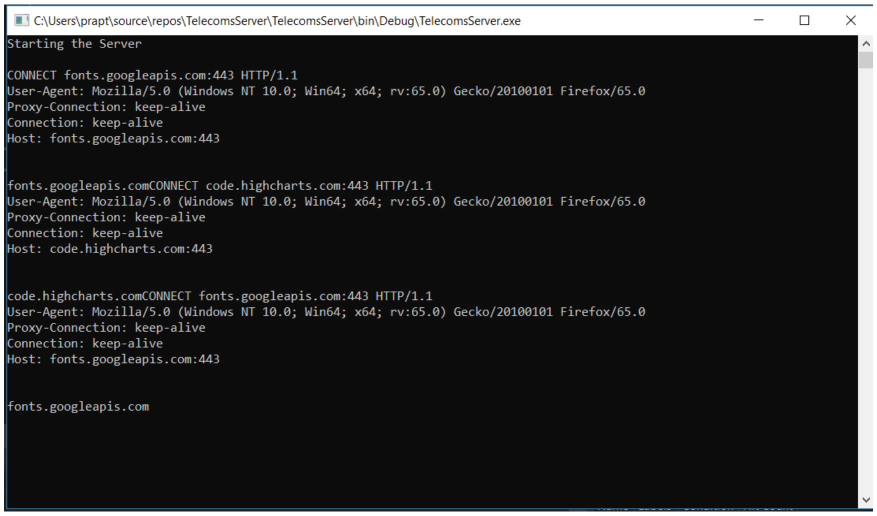
```
HTTPS/1.1 200 OK
www.bing.comCONNECT www.bing.com:443 HTTP/1.1
HTTPS/1.1 200 OK
CONNECT www.facebook.com:443 HTTP/1.1
```

To Block a URL's, enter it in the input box and then click "Block URL"

```
neverssl.com
```

As seen above, we can type into the bottom text box and block URLs. These URL's will never be loaded and after a certain amount of time this will timeout, and deny the connection. Once we click "Block URL" the url we entered gets added to an array of URLs, that stays blocked.

Simultaneously, the second console window will start logging all the information as shown below:


C:\Users\prapt\source\repos\TelecomsServer\TelecomsServer\bin\Debug\TelecomsServer.exe

```
Starting the Server

CONNECT fonts.googleapis.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: fonts.googleapis.com:443


fonts.googleapis.comCONNECT code.highcharts.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: code.highcharts.com:443


code.highcharts.comCONNECT fonts.googleapis.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0) Gecko/20100101 Firefox/65.0
Proxy-Connection: keep-alive
Connection: keep-alive
Host: fonts.googleapis.com:443


fonts.googleapis.com
```

I log all the information in a second window, so I can check if a page loads incorrectly, where it goes wrong and why it goes wrong. It is also a much more detailed logging as opposed to the quick logging on the management console. This mainly helps for HTTP since HTTPS webpages are encrypted and will often show up as random symbols that we cannot really understand.

We start by creating the listeners that are “listening” for any requests made. We set it to Port 5000, as we chose that for our settings in Firefox. (Firefox will only accept items coming in over port 5000). This is our first TCP Client.
If no requests are being made, we can tell the thread to pause/sleep for some amount of time. Once requests start getting made, we parameterize a new thread with a new session. This allows my server to meet the fifth implementation condition:

Handle multiple requests simultaneously by implementing a threaded server.

The program can now load multiple pages/tabs in the browser at the same amount time.

We must check for HTTP or HTTPS and we call on the function IsHttpOrHttps(string[] request) for this. If there is a Connect in the request, we know it is HTTP, else it is HTTPS.

Next we check if it is blocked by calling on a function that iterates through the array that stores all the blocked URLs and returns false if not blocked. If we get a true, we do not need to load the page. As mentioned previously, we can type into the bottom text box of the Display console and block URLs. These URL’s will never be loaded and after a certain amount of time this will timeout, and deny the connection. Once we click “Block URL” the url we entered gets added to an array of URLs, that stays blocked.
Now we can

Dynamically block selected URLs via the management console.

HTTP:
Once we know it is not blocked and a http request, we can create the second TCP Client at Port 80, the HTTP Port.

In HTTP, we need to forward the http request to the server, get the response, forward the response just got to the client and this displays it.

To forward requests, I have created a function called Send Message and to get the message, I call on my function Read Message.

HTTPS:
Once we know it is not blocked and a https request, we can create the second TCP Client at Port 443, the HTTPS Port.

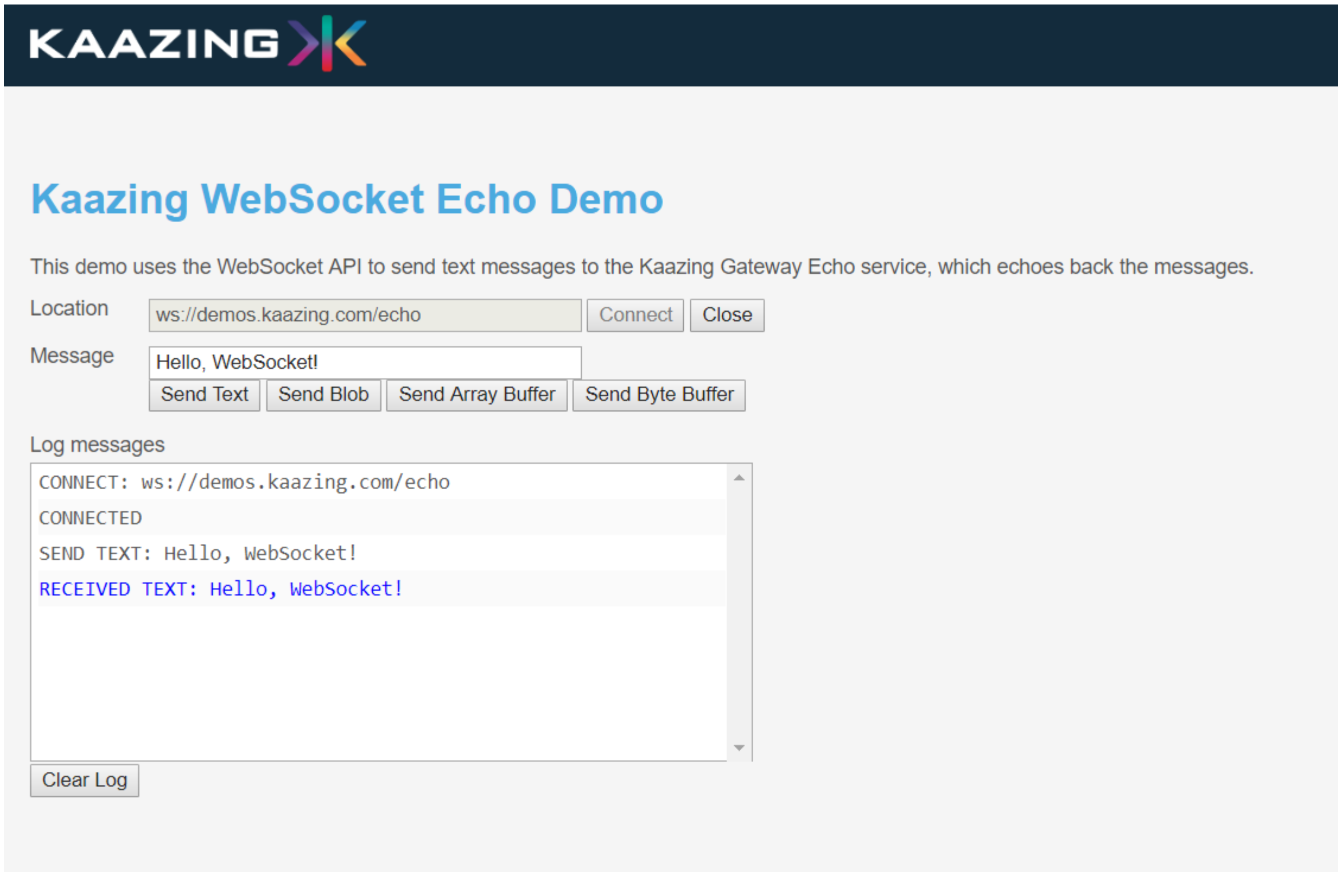
In HTTPS, we need to establish a connection by using a handshake method, and for this we send on a "HTTP/1.0 200 Connection established" message.
We can then begin forwarding the https request to the server, get the response, forward the response just got to the client and vice versa. It can send messages both ways. We do this on a loop, continually as information is sent back and forth. As mentioned previously, this is encrypted so we cannot log or cache this.

To forward requests, I have created a function called SendMessage and to get the message, I call on my function ReadMessage.

SendMessage is of type void as it does not need to return any information. Just needs to be able to send on a message. It writes to the webpage open the information it is getting in using the stream.Write function.

ReadMessage returns a byte array with the information we are looking for. I use an array to be able to chop off any excess white space that is not needed and copy information over to the array. This information is concatenated to the previous info stored in tempArray. This is needed as it is on a while loop that executes till the end of the information.

Once we have implemented all of the above, we can also test for web socket connections.
To do this I used this website:
http://demos.kaazing.com/echo/index.html?fbclid=IwAR2yLYbgNjz2hNxp8HQmzrxj_ZkqG7FYqrFHOAUqGC658vUPOefN5vOqKf0



Kaazing WebSocket Echo Demo

This demo uses the WebSocket API to send text messages to the Kaazing Gateway Echo service, which echoes back the messages.

Location: ws://demos.kaazing.com/echo

Message: Hello, WebSocket!

Log messages

```
CONNECT: ws://demos.kaazing.com/echo
CONNECTED
SEND TEXT: Hello, WebSocket!
RECEIVED TEXT: Hello, WebSocket!
```

As we can see, we can connect web sockets and send messages back and forth.

Now we have completed the following tasks of this assignment:

1. Respond to HTTP & HTTPS requests and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.
2. Handle WebSocket connections.
3. Dynamically block selected URLs via the management console.
4. Handle multiple requests simultaneously by implementing a threaded server.

All the code is attached below:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TelecomsServer
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault( false);
            Application.Run( new Form1());
        }
    }
}

```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.IO;
using System.Threading;
using System.Net.Sockets;

namespace TelecomsServer
{

    //creating the form/console window that displays all our information
    //the window contains a start server button and a block url button,
    //an input text box that can be used to block urls,
    //two log windows that display http and https requests
    public partial class Form1 : Form
    {
        public static string theHttpHosts = null;
        public static string theHttpsHosts = null;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Thread startProg = new Thread(() => ProxyServer.MainProg());
            startProg.Start();
        }

        private void button2_Click_1(object sender, EventArgs e)
        {
            string block = textBox2.Text;
            ProxyServer.addBlockURL(block);
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            textBox1.Text = theHttpHosts;
        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {
            textBox3.Text = theHttpsHosts;
        }

        public void changeHttps()
        {
            textBox1.Text = theHttpsHosts;
        }
    }

    public class ProxyServer
    {
        private TcpListener theListener;
        private bool runServer;
        public static List<string> blockedURL = new List<string>();

        //let the user know that the proxy server has been started
        public static void MainProg()
        {
            System.Console.WriteLine("Starting the Server");

            ProxyServer theServer = new ProxyServer();
            theServer.Start();
        }

        //the function called on when we block URLs
        public static void addBlockURL(string theURL)
        {
            blockedURL.Add(theURL);
        }
        //creating the start method that starts the listener
        public void Start()
        {
            this.theListener = new TcpListener(IPAddress.Any, 5000);
            this.theListener.Start();
            //ensure that we can run the server
            this.runServer = true;
            //execute these actions when our server is running
            while (this.runServer)
            {
                if (!theListener.Pending())
                {
                    //pause the thread for a small period
                    Thread.Sleep(200);
                    continue;
                }
                //create the tcp client for the listener
                TcpClient client = theListener.AcceptTcpClient();
                //create threads for the client session
                Thread session = new Thread(new ParameterizedThreadStart
                  (ClientSession));
                session.Start(client);
            }
        }

        public void ClientSession(object client)
        {
            TcpClient clientTCP = (TcpClient)client;
            NetworkStream clientStream = clientTCP.GetStream();
            //create a buffer
            byte[] buffer = null;
            //continually perform this loop
            while (true)
            {
                buffer = null;
                if (clientStream.CanRead && clientStream != null)
                {
                    //save the request made in the buffer (as ascii)
                    buffer = NetworkManager.ReadMessage(clientStream);
                }
                else
                {
                    continue;
                }
                //get the string val of this and display to console
                string request = Encoding.ASCII.GetString(buffer);
                Console.WriteLine(request);
                string[] splitRequest = request.Split(new char[0]);
                string host = GetHostFromRequest(splitRequest);
                //if the host is blocked then we do not want to continue
                  with the request
                if (!checkBlocked(host))
                {
                    Console.Write(host);
                    string httpOrHttps = IsHttpOrHttps(splitRequest);
                    //perform the action based on http or https
                    if (httpOrHttps == "HTTPS")
                    {
                        executeHttps(splitRequest, buffer, clientTCP, host);
                        Form1.theHttpsHosts += "\n" + host;
                        //Form1.changeHttps();
                    }
                    if (host == string.Empty)
                    {
                        continue;
                    }
                    Form1.theHttpHosts += "\n" + host;
                    //https port is 80 so create the tcpCkent for this and
                   get network stream
                    TcpClient serverTCP = new TcpClient(host, 80);
                    NetworkStream serverStream = serverTCP.GetStream();
                    // Forward HTTP request to server
                    if (serverStream.CanWrite && serverStream != null)
                    {
                        NetworkManager.SendMessage(serverStream, buffer);
                    }
                    else
                    {
                        continue;
                    }
                    // Get HTTP response from server
                    if (serverStream.CanRead && serverStream != null)
                    {
                        buffer = NetworkManager.ReadMessage(serverStream);
                    }
                    else
                    {
                        continue;
                    }
                    string response = Encoding.ASCII.GetString(buffer);
                    Console.Write(response);

                    // Forward HTTP response to client
                    if (clientStream.CanWrite && clientStream != null)
                    {
                        NetworkManager.SendMessage(clientStream, buffer);
                    }
                    else
                    {
                        continue;
                    }
                }
                else
                {

                }
            }
        }

        //function to check if a url has been blocked.
        //called on before the request is processed
        public bool checkBlocked(string host)
        {
            for (int i = 0; i < blockedURL.Count(); i++)
            {
                if (blockedURL[i].Contains(host))
                {
                    return true;
                }
            }
            return false;
        }

        //method to exectue https
        //very similar to http but we must be aware of the encryption
        //connection needs to be established first and
        //we need to send a response for this connection
        public void executeHttps(string[] splitRequest, byte[] buffer,
            TcpClient client, string host)
        {
            NetworkStream clientStream = client.GetStream();
            if (host == string.Empty)
            {
                return;
            }
            //https come sin over port 443
            //create the tcp client for this and find the network stream for ↩
               it
            TcpClient serverTCPhttps = new TcpClient(host, 443);
            NetworkStream serverStream = serverTCPhttps.GetStream();
            // Forward connection establish request
            byte[] establishConnection = Encoding.ASCII.GetBytes("HTTP/1.0  ↩
              200 Connection established\r\n\r\n");
            NetworkManager.SendMessage(clientStream, establishConnection);
            //as it isencryped we just want to forward requests on from 443 ↩
              to 5000
            int emptyRead1 = 0;
            int emptyRead2 = 0;
            while(emptyRead1 < 100 && emptyRead2 < 100)
            {
                byte[] temp1 = NetworkManager.ReadMessage(clientStream);
                if (temp1.Length != 0)
                {
                    NetworkManager.SendMessage(serverStream, temp1);
                    emptyRead1 = 0;
                }
                else
                {
                    emptyRead1++;
                }
                temp1 = NetworkManager.ReadMessage(serverStream);
                if (temp1.Length != 0)
                {
                    NetworkManager.SendMessage(clientStream, temp1);
                    emptyRead2 = 0;
                }
                else
                {
                    emptyRead2++;
                }
            }
            //close both client and server comunication if empty reads
            serverTCPhttps.Close();
            client.Close();
        }

        //function to check if it is a http or https request
        //https sends connect first
        //https can send get, post, etc
        public string IsHttpOrHttps(string[] request)
        {
            for (int i = 0; i < request.Length; i++)
            {
                if (request[i] == "CONNECT")
                {
                    return "HTTPS";
                }
                else
                    return "HTTP";
            }
            return string.Empty;
        }

        //used by both http and https
        //use this to get the host from the block of the request sent on
        //string matches host and returns what comes after host
        public string GetHostFromRequest(string[] request)
        {
            for (int i = 0; i < request.Length; i++)
            {
                if (request[i] == "Host:")
                {
                    string[] checkHost = request[i + 1].Split(':');

                    if (checkHost.Length != 1)
                    {
                        return checkHost[0];
                    }
                    else
                    {
                        return request[i + 1];
                    }
                }
            }

            return string.Empty;
        }
    }

    public class NetworkManager
    {

        //the function to read the message that is sent on from the network ↩
          stream
        public static byte[] ReadMessage(NetworkStream stream)
        {
            byte[] receiveBuffer = new byte[8192];
            byte[] tempArray = new byte[0];
            byte[] returnBuffer = new byte[0];
            int receivedBytes = 0;
            stream.ReadTimeout = 3000;
            try
            {
                if (stream.CanRead && stream != null)
                {
                    while ((receivedBytes = stream.Read(receiveBuffer, 0, receiveBuffer.Length)) != 0)
                    {
                        returnBuffer = new byte[receivedBytes];
                        Array.Copy(receiveBuffer, 0, returnBuffer, 0, receivedBytes);
                        tempArray = tempArray.Concat(returnBuffer).ToArray();
                    }
                }
            }
            catch (IOException e)
            {

            }
            return tempArray;
        }

        //the function to send the message that is sent on from the network stream
        public static void SendMessage(NetworkStream stream, byte[] sendBuffer)
        {
            if (stream.CanWrite && stream != null)
            {
                stream.Write(sendBuffer, 0, sendBuffer.Length);
            }
        }
    }
}
```

```
namespace TelecomsServer
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
          disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.button2 = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.textBox3 = new System.Windows.Forms.TextBox();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(12, 12);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(122, 44);
            this.button1.TabIndex = 0;
            this.button1.Text = "START Server";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler
              (this.button1_Click);
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(202, 76);
            this.textBox1.Multiline = true;
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(262, 174);
            this.textBox1.TabIndex = 1;
            this.textBox1.TextChanged += new System.EventHandler
              (this.textBox1_TextChanged);
            //
            // button2
            //
            this.button2.Location = new System.Drawing.Point(556, 372);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(122, 46);
            this.button2.TabIndex = 2;
            this.button2.Text = "Block URL";
            this.button2.UseVisualStyleBackColor = true;
            this.button2.Click += new System.EventHandler
              (this.button2_Click_1);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(199, 12);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(114, 17);
            this.label1.TabIndex = 3;
            this.label1.Text = "Server Requests";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(148, 303);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(440, 17);
            this.label2.TabIndex = 4;
            this.label2.Text = "To Block a URL\'s, enter it in the input box
               and then click \"Block URL\"";
            //
            // textBox2
            //
            this.textBox2.Location = new System.Drawing.Point(202, 361);
            this.textBox2.Multiline = true;
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(315, 77);
            this.textBox2.TabIndex = 5;
            this.textBox2.TextChanged += new System.EventHandler
              (this.textBox2_TextChanged);
            //
            // label3
            //
            this.label3.AutoSize = true;
```

```csharp
            this.label3.Location = new System.Drawing.Point(202, 53);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(45, 17);
            this.label3.TabIndex = 6;
            this.label3.Text = "HTTP";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(518, 53);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(54, 17);
            this.label4.TabIndex = 7;
            this.label4.Text = "HTTPS";
            //
            // textBox3
            //
            this.textBox3.Location = new System.Drawing.Point(521, 76);
            this.textBox3.Multiline = true;
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(267, 174);
            this.textBox3.TabIndex = 8;
            this.textBox3.TextChanged += new System.EventHandler
              (this.textBox3_TextChanged);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Display Console";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.TextBox textBox3;
    }
}


```