Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33032-202122 Advanced Computer Networks Assignment #1: Web-Proxy-server

Saumya Bahuguna, 21344349

March 6, 2022

Contents

1 Introduction

The problem description for this assignment required us to create a Web Proxy server that acts as an intermediary between the client and the Internet. The server fetches the information requested by the client without revealing any client address details.
As the assignment, we were provided with a task to implement a proxy server where the proxy server was required to perform certain tasks i.e;

- 1: Responding to HTTP & HTTPS requests and to display each request on a management console. It forwards the request to the Web server and relay the response to the browser.
- 2: Handle Web socket connections.
- 3: Dynamically block selected URLs via the management console.
- 4: Efficiently cache HTTP 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.

I have implemented all the features in my execution of the server. In this report, I will be describing HTTP and HTTPS. I will also cover some details about web caching, finally moving to the explanation of my implementation I will then end my report with a summary and reflection

2 Overall Design

In the First two Sections I will define my Understanding of **HTTP,HTTPS & Web Caching**

2.1 HTTP & HTTPS

Hypertext Transfer Protocol
Is a protocol used to fetch resources such as HTML documents and jpg, png e.t.c files, It is the base of data transmission on the web and is a client-server protocol which means requests are initiated by a client(usually a web browser) in a two-way communication. The web page that is constructed is a collection or a reconstruction of different types of data i.e videos, scripts, images e.t.c.
As mentioned it is a client-server protocol; requests are made by a client(or a proxy in our case) to the web server which then returns all the data which is then handled.

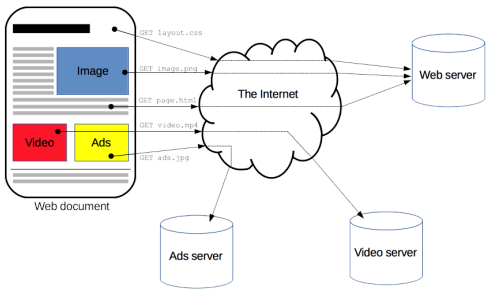

Figure 1: The figure above demonstrates a general idea of how HTTP requests the data and various files that are provided by the web server.

Hypertext Transfer Protocol Secure : is an encrypted version of HTTP. it uses Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to encrypt and secure the communication between two devices(Client and a server). The major difference between HTTP and HTTPS is, HTTPs is encrpted so teh response from the web browser is not visible to any potential attached unlike HTTP where the request is easily accessible.

2.2 Web Caching

A Web Cache is also known as a proxy server, it satisfies the HTTP requests on behalf of the main server, maintains all the previously requested objects from the web server, and keeps a copy, this saves a lot of bandwidth while requesting the same page. the cache server usually stores the Date of the last requested page and keeps a conditional if on any future requests that it makes to check if the page was updated after the date that it has stored the file on.

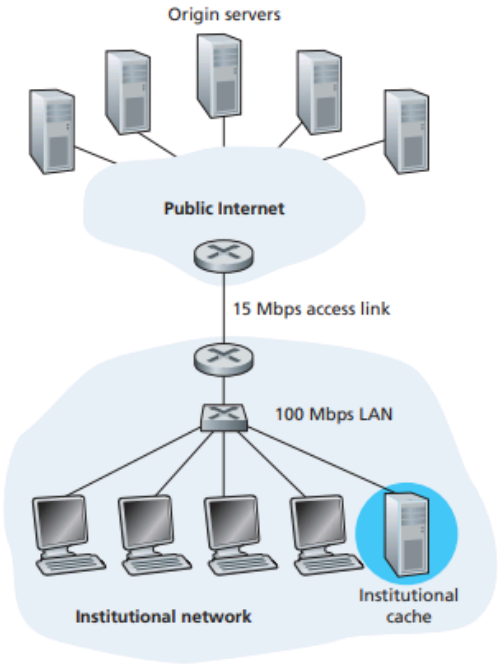

Figure 2.13 ♦ Adding a cache to the institutional network

Figure 2: The figure shows a bottle neck between a insituions network and the internet where the Cache server stores all the HTTP request responses which will nake the loading of the webpages much faster if they are not constantly updated

3 Implementation

This section presents the implementation details of the proxy server I have used Three classes in total to run the proxy server the Proxy class, the DriverThread class and the Clienttoservertunnel class I will be talking about them further in my report.

3.1 Proxy

This is the main class that calls all the other class this class handles all the user inputs(through the terminal), manages the thread and stores all the blocked and cached url to be specific this class handles all the functions which require only one instance.
It starts the server on port ”50000” and lsitens to all the requests from the browser and creates instances of DriverThread to manage it.

```
        public static void main(String[] args) {
//Creates an instance of Proxy and starts listening for requests from
the browser
                Proxy server = new Proxy(50000);
                server.listen();
        }
        public Proxy(int port) {
        //...
try {
                        Listen = new ServerSocket(port);
                System.out.println("Waiting for client on port " +
                Listen.getLocalPort());
                        running = true;
        } catch (SocketException e) {
        System.out.println("Socket Exception when connecting to client");
                } catch (SocketTimeoutException e) {
        System.out.println("Timeout occured while connecting to client");
        } catch (IOException e) {
        System.out.println("IO exception when connecting to client");
                }
        }

        // start a new instance of thread for every new
        connection that the browser requests
        public void listen() {
                while (running) {
                try {
                Socket socket = Listen.accept();
                Thread thread = new Thread(new DriverThread(socket));
                                threads.add(thread);
                                thread.start();
        } catch (SocketException e) {
                System.out.println("Server closed");
                        } catch (IOException e) {
System.out.println("Error creating new Thread from ServerSocket.");
                        }
                }
        }
```

Listing 1: The class starts a listener class of runnable on port 50000 which constantly *listens* for requests adn upon receiving a request adds the thread to an array list and starts a new thread

As per the requirement this class also has the management console which is constantly listening for user input and according the value added by the user takes proper action.

```
                cache = new HashMap<>();
                block = new HashMap<>();
                threads = new ArrayList<>();

                // Spin off a seperate thread to handle the management console
                managementConsole = new Thread(this);
                managementConsole.start();
```

Listing 2: The snippet shows the data structures used to store and handle cached and blocked urls and how the management console is started once on the thread so that it can run paralley with other requests *listens* for requests

There are other functions which upon initialization of the constructor either get all the cached files and blocked websites from the files on the hard drive or create a new file based on if the file exist or not. The other important function in this class is the close server method this when called closes all the threads and saves the cached and blocked data structure in individual files.

3.2 DriverThread

This is the class which handles all the requests from the browsers (HTTP and HTTPS) The code in Listing3 shows how the HTTP requests are managed by the class after extracting out of the GET request the content is checked if it is a Image or an URL by calling a fuction of the DriverThread class which returns true if image, if true it uses javas ImageIO library to communicate with the server and then send it to the client. If an Endpoint were to receive a packet it would run the onReceipt method. This method prints the received string to the terminal and then sends its response back to the original sender using a method called send().

```
if (checkifImage(fileExtension)) {
                URL remoteURL = new URL(requestUrl);
                BufferedImage image = ImageIO.read(remoteURL);

 if (image != null) {

  ImageIO.write(image, fileExtension.substring(1), fileToCache);
                String line = getResponse(200, false);
                clientWriter.write(line);
                clientWriter.flush();
  ImageIO.write(image, fileExtension.substring(1),
  browserSocket.getOutputStream());

                        } else {
                        String error = getResponse(404, false);
                        clientWriter.write(error);
                        clientWriter.flush();
                                return;
                        }
```

Listing 3: The snippet above shows how the the class handles if the content is an Image

the Listing4 shows how tif the request is not an image then Some metadata is set for the connection, and a reader is setup from the connection. The HTTP content is read from the server, and a 200 "OK" message is sent back to establish that the transfer was successful. The HTTP content is then written to the client (the browser) to interpret and then displayed to the user.

```
else {
                                URL remoteURL = new URL(requestUrl);
        remoteURL.openConnection();
ServerCon.setRequestProperty("Content-Type","application/x-www-form-
urlencoded");
ServerCon.setRequestProperty("Content-Language", "en-US");
ServerCon.setUseCaches(false);
                        ServerCon.setDoOutput(true);
                BufferedReader toServer = new BufferedReader(
                new InputStreamReader(ServerCon.getInputStream()));
                                String line = getResponse(200, false);
```

```
                                clientWriter.write(line);
                                while ((line = toServer.readLine()) != null) {
                                        clientWriter.write(line);
                                        if (caching) {
                                                cacheWriter.write(line);
                                        }
                                }
                                clientWriter.flush();
                                if (toServer != null) {
                                        toServer.close();
                                }
                        }
```

Listing 4: the snippet shows the connectiuon made with the server if the request is URL/Text

As the class handles the HTTP requests it checks if the cache for the requested page exits or not based on the response it passes the url to the code shows in Listing 5 which is a step taken by the program before the code in Listing3 and Listing4, which gets the name of the url by removing the illegal file names from the url and converting it into a filename that can be saved as a file and then stores the file name with the file path on the HashMap file path is the path of the folder on my hard drive named as cached which stores all the cached files.

```
private void Cachenotstored(String requestUrl) {
                try {

        int fileExtensionIndex = requestUrl.lastIndexOf(".");
        String fileExtension;
fileExtension = requestUrl.substring(fileExtensionIndex
,requestUrl.length());
String fileName = requestUrl.substring(0, fileExtensionIndex);
if(fileName.indexOf('.')!=fileName.lastIndexOf('.')
&&fileName.indexOf('.
')!=-1 &&fileName.lastIndexOf('.')!=-1 ){
        fileName=fileName.substring(fileName.indexOf('.'),
        fileName.lastIndexOf('.'));
                        }
                        else{
                fileName = fileName.substring(7,fileName.indexOf("."));
                }
                while(fileName.contains("/")||fileName.contains(".")){
                fileName = fileName.replace("/", "__");
                        fileName = fileName.replace(".", "_");
                        }

        if (fileExtension.contains("/")) {
                fileExtension = fileExtension.replace("/", "__");
                fileExtension = fileExtension.replace(".", "_");
                fileExtension += ".html";
                        }

                boolean caching = true;
                        File fileToCache = null;
                        BufferedWriter cacheWriter = null;
                        fileName = fileName + fileExtension;
                        try {
                                fileToCache = new File("cache/" + fileName);
```

```
                                if (!fileToCache.exists()) {
                                        fileToCache.createNewFile();
                                }

        cacheWriter = new BufferedWriter(new FileWriter(fileToCache));
                        } catch (IOException e) {
        System.out.println("Error trying to cache " + fileName);
                                caching = false;
                        e.printStackTrace();
                        } catch (NullPointerException e) {
        System.out.println("Null pointer opening file " + fileName);
                        }
```

Listing 5: The code shows the Cachenotstored function of the class which stores the cache of the HTTP website in the hard drive

Listing6 shows how the program handles a cached file that exists on the system, it is the similar method to how the page is loaded from the server when the file is not cached but it is much faster as the file is stored locally. the method to load and get the cached file can be seen of Listing 8

```
private void storedcache(File storedcacheFile) {
                try {
String fileExtension =storedcacheFile.getName().substring
(storedcacheFile.getName().lastIndexOf('.'));
        if (checkifImage(fileExtension)) {
        BufferedImage image = ImageIO.read(storedcacheFile);
        if (image == null) {
        System.out.println("Image " + storedcacheFile.getName() + " was null");
        String response = getResponse(404, false);
                clientWriter.write(response);
                clientWriter.flush();
                        } else {
                String response = getResponse(200, false);
                clientWriter.write(response);
                clientWriter.flush();
        ImageIO.write(image, fileExtension.substring(1),
        browserSocket.getOutputStream());
                        }
                } else {
BufferedReader storedcacheFileBufferedReader = new BufferedReader(
new InputStreamReader FileInputStream(storedcacheFile)));
                String response = getResponse(200, false);
                clientWriter.write(response);
                clientWriter.flush();
                String line;
while ((line = storedcacheFileBufferedReader.readLine()) != null) {
clientWriter.write(line);
                                }
                        clientWriter.flush();

                                if (storedcacheFileBufferedReader != null) {
                                        storedcacheFileBufferedReader.close();
                                }
                        }
                        if (clientWriter != null) {
                                clientWriter.close();
                        }
```

```
                } catch (IOException e) {
  System.out.println("Error sending storedcache file to client");
                }
        }
```

Listing 6: The code shows the storedCache function of the class which gets the previously stored HTTP file from the directory

```
        public static File getCachedPage(String url) {
                return cache.get(url);
        }
        public static void addCachedPage(String urlString, File fileToCache) {
                cache.put(urlString, fileToCache);
        }
        public static void  removeBlocked(String urlString) {
                block .remove(urlString);
                 {
                        try {
                          Files.deleteIfExists(
Paths.get("D:\\study\\eclipse Project\\PRoxyServer\\blockedSites.txt"));
                        }
                        catch (NoSuchFileException e) {
                            System.out.println(
                                "No such file/directory exists");
                        }
                        catch (DirectoryNotEmptyException e) {
                            System.out.println("Directory is not empty.");
                        }
                        catch (IOException e) {
                            System.out.println("Invalid permissions.");
                        }

                        System.out.println("Deletion successful.");
                    }
                }

        public static boolean isBlocked(String url) {
                for (String key : block.keySet()) {
                        if (url.contains(key)) {
                                return true;
                        }
                }
                return false;
        }
```

Listing 7: The code shows the functions that are used to get and store the blocked, cached function there is also a removeBlocked which removes the file form a hasmap and then deletes the text file which then completely removes a webpage from being blocked

As the Text in HTTP requests is handled in text(non-encrypted) as mentioned in Section 2 therefore it's easy to parse and divide the urls based on Get Requests images and htmls', unlike HTTP, HTTPS is end to end encrypted so the only Connect request is identified and the port and address can be extracted other than that the connection between the client and server needs to be an end to end connection as from the

proxy class we are already passing a socket to the client and have been reading and writing to that therefore we need not create the socket from the server to client but we need to create a socket form the client to the server and read and write objects on that socket as well, to make it work parallel a thread is also required to be created.

```
// Handle a HTTPS CONNECT request
        private void handleHTTPSRequest(String requestUrl) {
                String url = requestUrl.substring(7);
                String pieces[] = url.split(":");
                url = pieces[0];
                int serverPort = Integer.valueOf(pieces[1]);

                try {

                        for (int i = 0; i < 5; i++) {
                                clientReader.readLine();
                        }

                        InetAddress serverAddress = InetAddress.getByName(url);
Socket serverSocket = new Socket(serverAddress, serverPort);
                        serverSocket.setSoTimeout(5000);

                        String line = getResponse(200, true);
                        clientWriter.write(line);
                        clientWriter.flush();
                        //Create a Buffered Writer between proxy and remote

                BufferedWriter serverWriter = new BufferedWriter(new
                OutputStreamWriter(serverSocket.getOutputStream()));
                        // Create Buffered Reader from proxy and remote
                        BufferedReader serverReader = new BufferedReader(new
                        InputStreamReader(serverSocket.getInputStream()));
// Create a new thread to listen to client and transmit to server
        Clienttoservertunnel clientToServer = new Clienttoservertunnel(
                                        browserSocket.getInputStream(),
                                        serverSocket.getOutputStream());
                        clientToServerThread = new Thread(clientToServer);
                        clientToServerThread.start();

                        // Listen to remote server and relay to client
                        try {
                                byte[] buffer = new byte[4096];
                                int read;
                                do {
        read = serverSocket.getInputStream().read(buffer);
        if (read > 0) {
        browserSocket.getOutputStream().write(buffer, 0, read);
                if (serverSocket.getInputStream().available() < 1) {
                        browserSocket.getOutputStream().flush();
                        }
                }
                } while (read >= 0);
        } catch (SocketTimeoutException e) {
                System.out.println("Socket timeout during HTTPs connection");
        } catch (IOException e) {
                System.out.println("Error handling HTTPs connection");
```

```
        }
                // Close the resources
        closeResources(serverSocket, serverReader,
                                serverWriter, clientWriter);

                } catch (SocketTimeoutException e) {
                        String line = getResponse(504, false);
                        try {
                                clientWriter.write(line);
                                clientWriter.flush();
                        } catch (IOException x) {
                        }
                } catch (Exception e) {
                        System.out.println("Error on HTTPS " + requestUrl);
                }
        }
```

Listing 8: The code shows the handleHttpsrequest function which sets up the url creates a server port and then creates a reader and writer object and then a thread is created to run parallel Lisitng9 shows the clienttoservertunnel class

```
class Clienttoservertunnel implements Runnable {

                InputStream client;
                OutputStream server;

public Clienttoservertunnel(InputStream
       ClientInput, OutputStream serverout) {
                        this.client = ClientInput;
                        this.server = serverout;
                }

                @Override
                public void run() {
                        try {
                                byte[] buffer = new byte[4096];
                                int read;
                                do {
                                        read = client.read(buffer);
                                        if (read > 0) {
                                                server.write(buffer, 0, read);
                                                if (client.available() < 1) {
                                                        server.flush();
                                                }
                                        }
                                } while (read >= 0);
                        } catch (SocketTimeoutException e) {
                        } catch (IOException e) {
        System.out.println("Proxy to client HTTPS read timed out");
                        }
                }
```

Listing 9: The code shows the Clienttoservertunnel class which creates a server side socket for the Https end to end communication shows the clienttoservertunnel class

4 Summary

This report showed the components that I used to create my implementation of the proxy server all the task mentioned for the assignment where achieved with a few more tasks in the end to conclude witht he help of 3 diffrent classes I was succesfully able to create a proxy server which was able to do things like Handeling the HTTP requests and cache the pages, succesfully able to block and unblock a webpage, adn after the server is shutdown it still stores all the cached and blocked websites on the local storage

5 Reflection

To look back to reflect what benefits this assignment had, I can say that while doing and researching about the assignment I learned a lot about Internet and how different types of requests that are Handled in the Application layer, I got appropriate understanding of Computer Networks

6 references

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview Computer Networking a top-Down Approach Kurose,Ross

7 Appendix

```
package ProxyServer;
import java.io.*;
import java.net.*;
import java.nio.file.DirectoryNotEmptyException;
import java.nio.file.Files;
import java.nio.file.NoSuchFileException;
import java.nio.file.Paths;
import java.util.*;


/*This class is the implementation of runnable
and works as the engine of my
implementation i.e will be waiting for the requests
 *and connection listening o n a port(5000)
 it will call the Driverthread class
 which implements threads and creates multiple
 *Driver threads to handle multiple requests separately.
 This class manages the user inputs as well as blocked
 and cached Web sites When closed it keeps track of the blocked
 and cached web sites by saving the files on a local hard
 drive which will then be extracted if required

 * */
public class Proxy implements Runnable {

        public static void main(String[] args) {
// Create an instance of Proxy and starts listening for requests form the
browser
                Proxy server = new Proxy(50000);
                server.listen();
        }
/*  DataStructures and variable required to run the programme*/
```

```
        private static HashMap<String, File> cache;
        private static HashMap<String, String> block;
        private static ArrayList<Thread> threads;
        private ServerSocket Listen;
        private volatile boolean running = true;
        private Thread managementConsole;
/*
 * Methods to return values contained in the datastructures
 *
 * */

        public static File getCachedPage(String url) {
                return cache.get(url);
        }
        public static void addCachedPage(String urlString, File fileToCache) {
                cache.put(urlString, fileToCache);
        }
        public static void  removeBlocked(String urlString) {
                block .remove(urlString);
                 {
                        try {
                            Files.deleteIfExists(
Paths.get("D:\\study\\eclipse Project\\PRoxyServer
blockedSites.txt"));
                        }
                        catch (NoSuchFileException e) {
                            System.out.println(
                                "No such file/directory exists");
                        }
                        catch (DirectoryNotEmptyException e) {
                            System.out.println("Directory is not empty.");
                        }
                        catch (IOException e) {
                            System.out.println("Invalid permissions.");
                        }

                        System.out.println("Deletion successful.");
                    }
                }

        public static boolean isBlocked(String url) {
                for (String key : block.keySet()) {
                        if (url.contains(key)) {
                                return true;
                        }
                }
                return false;
        }
        /*
         * Constructor for the Proxy.
         */
        public Proxy(int port) {
                // initialise the data structures and client port number
                cache = new HashMap<>();
                block = new HashMap<>();
```

```
                threads = new ArrayList<>();

                // Spin off a seperate thread to handle the management console
                managementConsole = new Thread(this);
                managementConsole.start();
                //Initalize the maps of cached site and blocked sites
                initializeCachedSites();
                initializeBlockedSites();
                // Start listening to the browser
                try {
                        Listen = new ServerSocket(port);
        System.out.println("Waiting for client on port " +
        Listen.getLocalPort());
        running = true;
                } catch (SocketException e) {
System.out.println("Socket Exception when connecting to client");
                } catch (SocketTimeoutException e) {
System.out.println("Timeout occured while connecting to client");
        } catch (IOException e) {
System.out.println("IO exception when connecting to client");
                }
        }


//start a new instance of thread for every new connection that the browser
requests
        public void listen() {
                while (running) {
                        try {
Socket socket = Listen.accept();
Thread thread = new Thread(new DriverThread(socket));
threads.add(thread);
thread.start();
} catch (SocketException e) {
System.out.println("Server closed");
} catch (IOException e) {
System.out.println("Error creating new Thread from ServerSocket.");
                        }
                }
        }

//Initalize the data structure for the
//cached sites by reading from the cache
//file. If a cache file does not exist, create one
        @SuppressWarnings("unchecked")
        private void initializeCachedSites() {
                try {
                        File cachedSites = new File("cachedSites.txt");
                        if (!cachedSites.exists()) {
                                System.out.println("Creating new cache file");
                                cachedSites.createNewFile();
                        } else {

                FileInputStream cachedFileStream = new
                FileInputStream(cachedSites);
```

```
        ObjectInputStream cachedObjectStream = new
        ObjectInputStream(cachedFileStream);
        cache = (HashMap<String, File>) cachedObjectStream.readObject();
                cachedFileStream.close();
                cachedObjectStream.close();
                        }
                } catch (IOException e) {
        System.out.println("Error loading previously
        cached sites file");
        } catch (ClassNotFoundException e) {
        System.out.println("Class not found loading in previously cached
                                sites file");
                }
        }

//Initalize the data structure for the blocked sites by reading from the
// blocked file. If a blocked file does not exist, create one
        private void initializeBlockedSites() {
                try {
        File blockedSitesTxtFile = new File("blockedSites.txt");
        if (!blockedSitesTxtFile.exists()) {
                                System.out.println("No blocked files");
                                blockedSitesTxtFile.createNewFile();
                        } else {
                                FileInputStream blockedFileStream = new
                                FileInputStream(blockedSitesTxtFile);
                                ObjectInputStream blockedObjectStream = new
                                ObjectInputStream(blockedFileStream);
                                block = (HashMap<String, String>)
                                blockedObjectStream.readObject();
                                blockedFileStream.close();
                                blockedObjectStream.close();
                        }
                } catch (IOException e) {
System.out.println("Error loading
previously Blocked sites file");
                } catch (ClassNotFoundException e) {
        System.out.println("Class not found
        loading in preivously Blocked sites file");
                }
        }

// Close the server. Write back to the cached and blocked files. Join the
// threads.
        private void closeServer() {
                System.out.println("Closing server");
                running = false;
                try {
                FileOutputStream cachedFileStream = new
                FileOutputStream("cachedSites.txt");
                ObjectOutputStream cachedObjectStream = new
                ObjectOutputStream(cachedFileStream);

                cachedObjectStream.writeObject(cache);
```

```
                cachedObjectStream.close();
                cachedFileStream.close();
                System.out.println("Cached sites written");

                FileOutputStream blockedFileStream = new
                FileOutputStream("blockedSites.txt");
                ObjectOutputStream blockedObjectStream = new
                ObjectOutputStream(blockedFileStream);
                blockedObjectStream.writeObject(block);
                blockedObjectStream.close();
                blockedFileStream.close();
                System.out.println("Blocked site list saved");
                try {
                for (Thread thread : threads) {
                if (thread.isAlive()) {
                                        thread.join();
                                }
                        }
                        } catch (InterruptedException e) {
        System.out.println("Interrupted exception when closing
                server.");
                        }

        } catch (IOException e) {
        System.out.println("Error saving cache/blocked sites");
                }
        try {
        System.out.println("Terminating connection");
        Listen.close();
                } catch (Exception e) {
        System.out.println("Exception closing proxy's server socket");
        e.printStackTrace();
                }
        }

        // The functionality of the management console.
        Watch System.in and look out for
        // commands. If a defined command is
not entered, assume that the input is a URL
        // to be blocked and block it.
        @Override
        public void run() {
                Scanner terminalScanner = new Scanner(System.in);
                String userInput;
                while (running) {
        System.out.println("Please enter a command.
        Enter 9 to see the list of commands.");
                        userInput = terminalScanner.next();

                        switch (userInput) {
                case "1":
        System.out.println("\nCurrently Blocked Sites");
                for (String key : block.keySet()) {
        System.out.println(key);
                                }
```

```
        System.out.println();
                                        break;
        case "2":
        System.out.println("\nCurrently␣Cached␣Sites");
        for (String key : cache.keySet()) {
                                                System.out.println(key);
                                        }
                                        System.out.println();
                                        break;
                                case "3":
                                        running = false;
                                        closeServer();
                                        break;

                                case "4":
         System.out.println("Enter␣the␣site␣to␣be␣REMOVED␣from
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣BLOCKED␣");
                                String r= terminalScanner.next();
                                 removeBlocked(r);
                                                break;
                                case "9":
        System.out.println("Enter␣1␣to␣view␣the␣list␣of␣blocked
␣␣␣␣␣␣␣␣URLs.");
        System.out.println("Enter␣2␣to␣view␣the␣list␣of␣caches
␣␣␣␣␣␣␣␣webpages");
        System.out.println("Enter␣3␣to␣close␣the␣proxy␣server.");
        System.out.println("Enter␣4␣to␣UNBLOCK␣a␣website");
        System.out.println("Enter␣any␣url␣to␣BLOCK" );
        System.out.println("Enter␣9␣to␣see␣the␣list␣of␣possible
␣␣␣␣␣␣␣␣commands");

        break;
       default:

block.put(userInput.toLowerCase(), userInput.toLowerCase());
System.out.println("\n" + userInput + "␣blocked␣successfully
␣␣␣␣␣␣␣␣\n");


                                        break;

                                }
                        }
                        terminalScanner.close();
        }
}
```

Listing 10: Proxy.java

```
package ProxyServer;

import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.*;
import java.net.*;
import java.util.Random;
```

```
public class DriverThread implements Runnable {

        /*
         * Variables and constants  that are used in the code
         */
        private Socket browserSocket;
        private BufferedReader clientReader;
        private BufferedWriter clientWriter;
        private Thread clientToServerThread;
    public Random a = new Random();
        /*
         * Constants for this class. To identify connection types
     and file types.
         */
        private static final String CONNECT = "CONNECT";
        private static final String JPG = ".jpg";
        private static final String JPEG = ".jpeg";
        private static final String PNG = ".png";
        private static final String GIF = ".gif";

        /*
         * Constructor for this thread. Initialize the local variables.
         */
        public DriverThread(Socket browserSocket) {
                this.browserSocket = browserSocket;
                try {
                        this.browserSocket.setSoTimeout(2000);
                        clientReader = new BufferedReader(new
            InputStreamReader(browserSocket.
            getInputStream()));
                        clientWriter = new BufferedWriter(new
            OutputStreamWriter(browserSocket.
            getOutputStream()));
                } catch (IOException e) {
            System.out.println("Error initializing new thread.");
                }
        }

        /*
        This Function check if the url that was requested by
    the client is blocked or not if
    it is do no t continue otherwise segregate the request
    based on http and https
         */
        @Override
        public void run() {
                String requestString, requestType, requestUrl;
                try {
                        requestString = clientReader.readLine();
                        String[] splitRequest = splitRequest(requestString);
                        requestType = splitRequest[0];
                        requestUrl = splitRequest[1];
                } catch (IOException e) {
                        System.out.println("Error reading request from
```

```
␣␣␣␣␣␣␣␣␣␣␣␣client.");
                        return;
                }

                if (Proxy.isBlocked(requestUrl)) {
                        System.out.println("Blocked␣site␣" + requestUrl +
            "␣requested.");
                        handleblocked();
                        return;
                }

                switch (requestType) {
                        case CONNECT:
                                System.out.println("HTTPS␣request␣for␣:␣"
                + requestUrl);
                                handleHTTPSRequest(requestUrl);
                                break;
                        default:
                                File file = Proxy.getCachedPage(requestUrl);
                                if (file == null) {
System.out.println("HTTP␣request␣for␣:␣" + requestUrl +
".␣No␣storedcache
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣page␣found.");
                                        Cachenotstored(requestUrl);
                                } else {
System.out.println("HTTP␣request␣for␣:␣" + requestUrl +
".␣storedcache
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣page␣found.");
                                        storedcache(file);
                                }
                                break;
                }
        }

        // Parse a CONNECT or GET request and return
    an array containing the URL and
        // port number
        private String[] splitRequest(String requestString) {
                String requestType, requestUrl;
                int requestSeparatorIndex;
                requestSeparatorIndex = requestString.indexOf('␣');
                requestType = requestString.substring(0,
        requestSeparatorIndex);
                requestUrl = requestString.substring
        (requestSeparatorIndex + 1);
                requestUrl = requestUrl.substring(0,
        requestUrl.indexOf('␣'));
                if (!requestUrl.substring(0, 4).equals("http")) {
                        requestUrl = "http://" + requestUrl;
                }
                return new String[] { requestType, requestUrl };
        }

        // Fetch a page from the cache for the client
        private void storedcache(File storedcacheFile) {
```

```
                try {
                        String fileExtension = storedcacheFile.
            getName().substring(
            storedcacheFile.getName().lastIndexOf('.'));
                        if (checkifImage(fileExtension)) {
                                BufferedImage image = ImageIO.read
                    (storedcacheFile);
                                if (image == null) {
                                        System.out.println("Image " +
                        storedcacheFile.getName() + " was null");
                                        String response =
                        getResponse(404, false);
                                        clientWriter.write(response);
                                        clientWriter.flush();
                                } else {
                                        String response =
                        getResponse(200, false);
                                        clientWriter.write(response);
                                        clientWriter.flush();

                    ImageIO.write(image, fileExtension.substring(1),
                    browserSocket.getOutputStream());
                                }
                        } else {
                                BufferedReader storedcacheFileBufferedReader =
                    new BufferedReader(
                                                new InputStreamReader(new
                            FileInputStream(storedcacheFile)));
                                String response = getResponse(200, false);
                                clientWriter.write(response);
                                clientWriter.flush();
                                String line;
                                while ((line = storedcacheFileBufferedReader.
                    readLine()) != null) {
                                        clientWriter.write(line);
                                }
                                clientWriter.flush();

                                if (storedcacheFileBufferedReader != null) {
                                        storedcacheFileBufferedReader.close();
                                }
                        }
                        if (clientWriter != null) {
                                clientWriter.close();
                        }
                } catch (IOException e) {
                        System.out.println("Error sending
                ␣␣␣␣␣␣␣␣␣␣␣␣storedcache file to client");
                }
        }

        // handles a not cached HTTP url and adds the
    value and the path to the HashMap and saves the
    files on the hard drive in the folder made for cached files.
        private void Cachenotstored(String requestUrl) {
```

```
                try {

                        int fileExtensionIndex =
            requestUrl.lastIndexOf(".");
                        String fileExtension;
                        fileExtension = requestUrl.substring
            (fileExten
            sionIndex, requestUrl.length());
                        String fileName =
            requestUrl.substring(0, fileEx
            tensionIndex);
                if(fileName.indexOf('.')!=
           fileName.lastIndexOf('.')&
           &fileName.indexOf('.')!=-1
               &&fileName.lastIndexOf('.')!=-1 ){
                                fileName=fileName.substring
                (fileName.indexOf('.'),
                fileName.lastIndexOf('.'));
                        }
                        else{
fileName = fileName.substring(7,fileName.indexOf("."));
                        }
                        while(fileName.contains("/")||fileName
            .contains(".")){
                        fileName = fileName.replace("/", "__");
                        fileName = fileName.replace(".", "_");
                        }

                        if (fileExtension.contains("/")) {
                                fileExtension =
                    fileExtension.replace("/", "__");
                                fileExtension =
                    fileExtension.replace(".", "_");
                                fileExtension += ".html";
                        }

                        boolean caching = true;
                        File fileToCache = null;
                        BufferedWriter cacheWriter = null;
                        fileName = fileName + fileExtension;
                        try {
                                fileToCache =
                    new File("cache/" + fileName);
                                if (!fileToCache.exists()) {
                                        fileToCache.createNewFile();
                                }

                                cacheWriter = new BufferedWriter(new
                    FileWriter(fileToCache));
                        } catch (IOException e) {
                                System.out.println("Error trying to cache " +
                    fileName);
                                caching = false;
                                e.printStackTrace();
                        } catch (NullPointerException e) {
```

```
                                System.out.println("Null␣pointer␣openin
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣g␣file␣" + fileName);
                        }
                        if (checkifImage(fileExtension)) {
                                URL remoteURL = new URL(requestUrl);
                                BufferedImage image = ImageIO.read(
                remoteURL);

                                if (image != null) {
        ImageIO.write(image, fileExtension.substring(1),
    fileToCache);
                                        String line = getResponse(200, false);
                                        clientWriter.write(line);
                                        clientWriter.flush();
                                        ImageIO.write(image, fileExtension.
                    substring(1),
                    browserSocket.getOutputStream());

                                } else {
                                        String error = getResponse(404, false);
                                        clientWriter.write(error);
                                        clientWriter.flush();
                                        return;
                                }
                        } else {
URL remoteURL = new URL(requestUrl);
        HttpURLConnection ServerCon =
    (HttpURLConnection)
     remoteURL.openConnection();
ServerCon.setRequestProperty("Content-Type",
"application/x-www-form-
␣␣␣␣␣␣␣␣␣␣␣urlencoded");
ServerCon.setRequestProperty("Content-Language",
"en-US");
ServerCon.setUseCaches(false);
ServerCon.setDoOutput(true);
BufferedReader toServer = new BufferedReader(
new InputStreamReader(ServerCon.
getInputStream()));
                                String line = getResponse(200, false);
                                clientWriter.write(line);
                                while ((line = toServer.readLine()) != null) {
                                        clientWriter.write(line);
                                        if (caching) {
        cacheWriter.write(line);
                                        }
                                }
                                clientWriter.flush();
                                if (toServer != null) {
                                        toServer.close();
                                }
                        }

                        if (caching) {
                                cacheWriter.flush();
```

```
                                Proxy.addCachedPage(requestUrl, fileToCache);
                        }
                        if (cacheWriter != null) {
                                cacheWriter.close();
                        }
                        if (clientWriter != null) {
                                clientWriter.close();
                        }
                } catch (Exception e) {
    System.out.println("Error sending non stored
     cache page to client");
                }
        }

        // Handle a HTTPS CONNECT request
        private void handleHTTPSRequest(String requestUrl) {
                String url = requestUrl.substring(7);
                String pieces[] = url.split(":");
                url = pieces[0];
                int serverPort = Integer.valueOf(pieces[1]);

                try {

                        for (int i = 0; i < 5; i++) {
                                clientReader.readLine();
                        }

InetAddress serverAddress = InetAddress
.getByName(url);
Socket serverSocket = new Socket
(serverAddress, serverPort);
                        serverSocket.setSoTimeout(5000);

                        String line = getResponse(200, true);
                        clientWriter.write(line);
                        clientWriter.flush();
                        //Create a Buffered Writer between proxy and remote

        BufferedWriter serverWriter =
    new BufferedWriter(new
 OutputStreamWriter(serverSocket.
 getOutputStream()));
        // Create Buffered Reader from
    proxy and remote
 BufferedReader serverReader = new
 BufferedReader(new
            InputStreamReader(serverSocket.
            getInputStream()));
// Create a new thread to listen to client and transmit to server
        Clienttoservertunnel clientToServer = new
    Clienttoservertunnel(
        browserSocket.getInputStream(),
        serverSocket.getOutputStream());
                        clientToServerThread = new
            Thread(clientToServer);
```

```
                        clientToServerThread.start();

                        // Listen to remote server and relay to client
                        try {
                                byte[] buffer = new byte[4096];
                                int read;
                        do {
                read = serverSocket.getInputStream().read(buffer);
                if (read > 0) {
                browserSocket.getOutputStream().write(buffer, 0, read);
        if (serverSocket.getInputStream().available() < 1) {
                browserSocket.getOutputStream().flush();
                                                }
                                        }
                                } while (read >= 0);
                        } catch (SocketTimeoutException e) {
        System.out.println("Socket timeout during
    HTTPs connection");
} catch (IOException e) {
System.out.println("Error handling HTTPs
connection");
                        }

        // Close the resources
        closeResources(serverSocket, serverReader, serverWriter,
        clientWriter);

                } catch (SocketTimeoutException e) {
                        String line = getResponse(504, false);
                        try {
                                clientWriter.write(line);
                                clientWriter.flush();
                        } catch (IOException x) {
                        }
                } catch (Exception e) {
                        System.out.println("Error on HTTPS " + requestUrl);
                }
        }

        // Respond to the browser with a 403 Error Code
        private void handleblocked() {
                try {
        BufferedWriter bufferedWriter = new BufferedWriter(new
        OutputStreamWriter(browserSocket.getOutputStream()));
        String line = getResponse(403, false);
bufferedWriter.write(line);
        bufferedWriter.flush();
        } catch (IOException e) {
        System.out.println("Error writing to client when requested a
        blocked site");
                }
        }

        // Check whether the file extension is that of an image
        private boolean checkifImage(String fileExtension) {
```

```
                return fileExtension.contains(PNG) ||
                fileExtension.contains(JPG) ||
                fileExtension.contains(JPEG)
                                || fileExtension.contains(GIF);
        }

/*
 *
 * it gets the response string based on the error that was received
 *
 * */
        private String getResponse(int code,
    boolean connectionEstablished) {
                String response = "";
                switch (code) {
                        case 200:
                if (connectionEstablished) {
                response = "HTTP/1.0 200 Connection \
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣established\r\nProxy-Agent: ProxyServer/1.0\r\n\r\n";
                                } else {
                            response = "HTTP/1.0 200 OK\nProxy-agent
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ProxyServer/1.0\n\r\n";
                }
        break;
        case 403:
response = "HTTP/1.0 403 Access Forbidden \nUser-Agent:
ProxyServer/1.0\n\r\n";
                                break;
                            case 404:
response = "HTTP/1.0 404 NOT FOUND \nProxy-agent:
ProxyServer/1.0\n\r\n";
                                break;
case 504:
response = "HTTP/1.0 504 Timeout Occured after 10s\nUser-Agent:
ProxyServer/1.0\n\r\n";
                                break;
default:
                                break;
                }
                return response;
        }

        // Close the passed resources
private void closeResources(Socket serverSocket,
BufferedReader serverReader,BufferedWriter serverWriter,
                        BufferedWriter clientWriter) throws IOException {
                if (serverSocket != null) {
                        serverSocket.close();
                }
                if (serverReader != null) {
                        serverReader.close();
                }
                if (serverWriter != null) {
                        serverWriter.close();
                }
```

```
                if (clientWriter != null) {
                        clientWriter.close();
                }
        }
/*this class handles the tunneling between  client and the server
*and isstarted as thread to run parallel to
*server to client tunneling
 *
 *
 *
 *
 * */
        class Clienttoservertunnel implements Runnable {

                InputStream client;
                OutputStream server;

public Clienttoservertunnel(InputStream ClientInput,
OutputStream serverout) {
                        this.client = ClientInput;
                        this.server = serverout;
                }

                @Override
                public void run() {
                        try {
                                byte[] buffer = new byte[4096];
                                int read;
                                do {
                                        read = client.read(buffer);
        if (read > 0) {
                server.write(buffer, 0, read);
                if (client.available() < 1) {
                server.flush();
                                        }
                                } while (read >= 0);
                } catch (SocketTimeoutException e) {
                } catch (IOException e) {
System.out.println("Proxy to client HTTPS read timed out");
                }
        }
}
}
```

Listing 11: DriverThread.java & Clienttoservertunnel