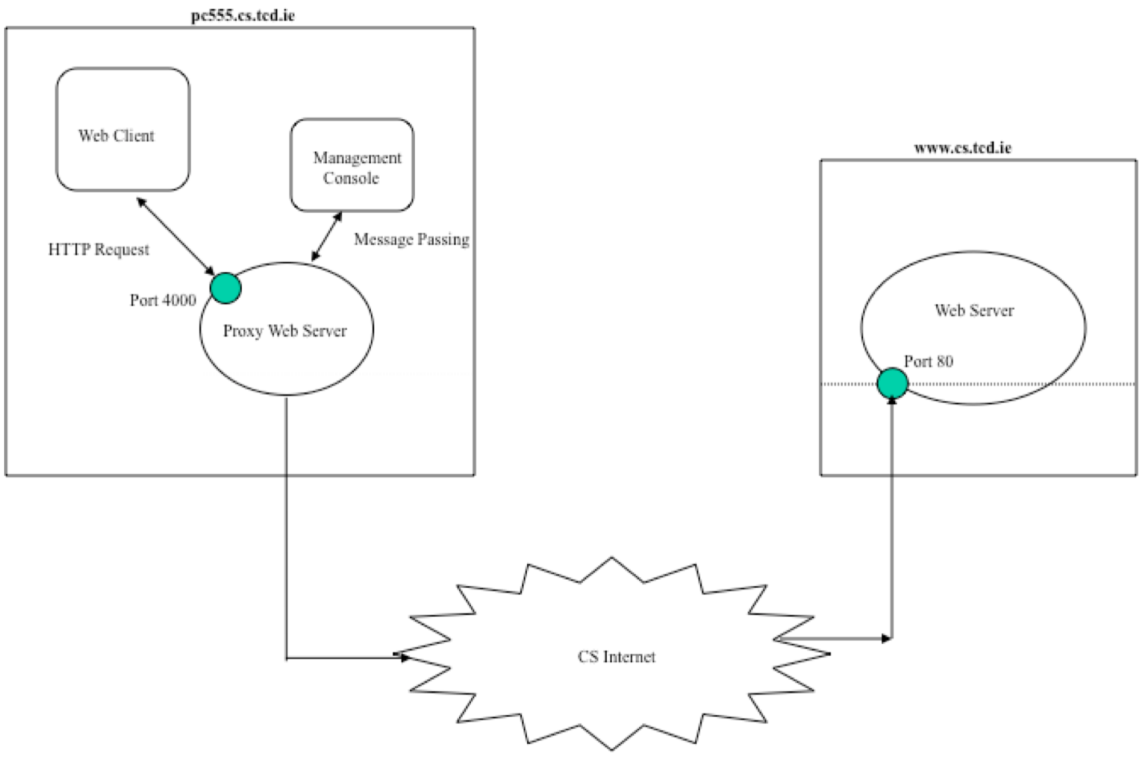
Course: CS3031 Advanced Telecommunications
Title: Assignment 1 - Web Proxy Server Documentation
Name: Leong Kai Ler
Student Number: 15334636
Date: February 26, 2019

Introduction:

In computer networks, a proxy server is a server that acts as the intermediary application connecting the clients and servers. It is typically used to receive information seeking requests from clients and direct them towards the target servers. The functionalities of proxy servers also encompasses processes such as comprehending and evaluating complex requests before simplifying them. Furthermore, the implementation of proxy servers also adds layers of structure and encapsulation to distributed systems in real world.

Purpose:

The purpose of this project is to design a web proxy sever that serves as a local server, in retrieving items from the Web on behalf of a Web client instead of the client fetching them directly. Such design enables the proxy server to control access flow through it and cache web pages.

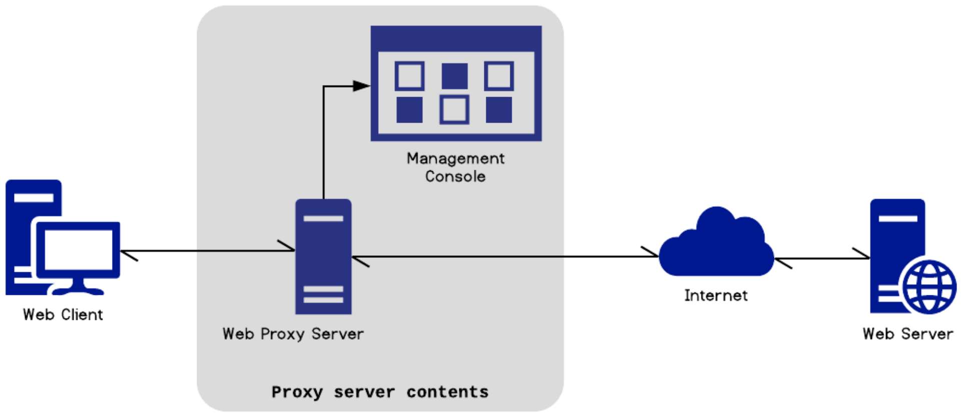


The functionalities of the web proxy server should include the following aspects:

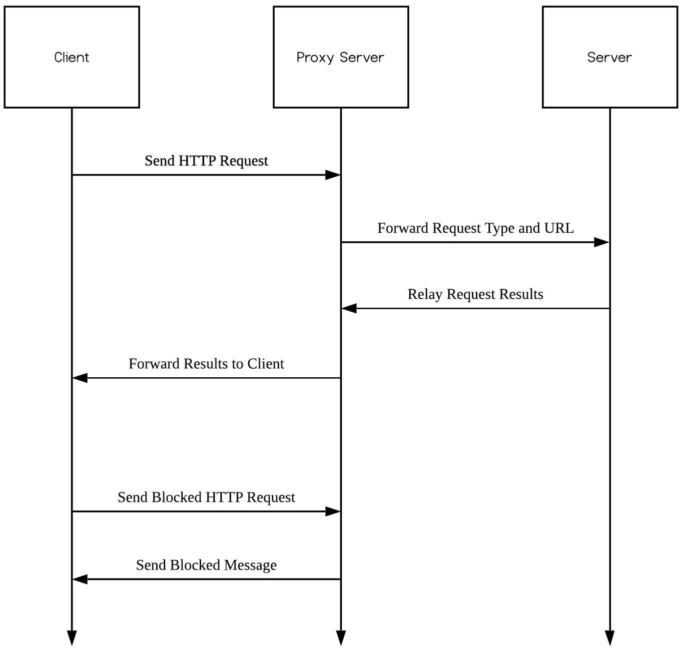
- Respond to HTTP & HTTPS requests, and should display each request on a management console. It should forward the request to the Web server and relay the response to the browser.
- Handle websocket connections.
- Dynamically block selected URLs via the management console.
- Efficiently cache requests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

Design:

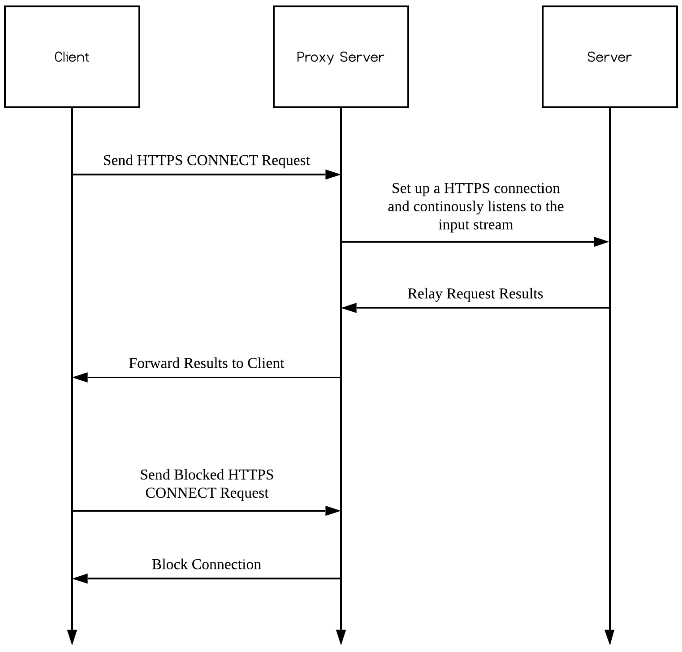
As mentioned above the intended proxy server is divided into mainly two components: the web proxy server and a management console. This program is written using Java programming language.



The management console is responsible in displaying the traffic on input and output of data streams through the proxy server, while the web proxy server listens to the web clients by creating threads and respond to their requests accordingly and setting up HTTP and HTTPS connections to retrieved information seek by web clients. It then relays the results of the request back to them respectively. A thread pool is created to access active threads each containing a request and execute them by creating sockets to establish connections between local server with web servers. In essence, the process of handling a HTTP request can be simplified as below:



On the other hand, the process of handling a HTTPS request would look slightly different as displayed in the following image:


Client
Proxy Server
Server
Send HTTPS CONNECT Request
Set up a HTTPS connection and continously listens to the input stream
Relay Request Results
Forward Results to Client
Send Blocked HTTPS CONNECT Request
Block Connection

Furthermore, it should be able to dynamically block websites that are stored in a blocked list. In other words, every time before setting up any connections, the proxy server has to confirm if the host names of the provided websites has been blacklisted. If the host name of the website is present in the blocked list, it should prevent the request from being executed and display a forbidden message as a warning signal to the web client. Otherwise, the web proxy can deal with the request normally.

In order to improve operating efficiencies, a cache is to be set up to store data of previously visited websites. These data can range from normal text files, images in the form of png, jpg, jpeg or gif, html pages and etc. To achieve this, a hash map is set up to check for existence of a requested page in a cache set up before setting up a connection to deal with it. A timer has also been set up to display the difference in duration of handling GET request with and without the assistance of cache.

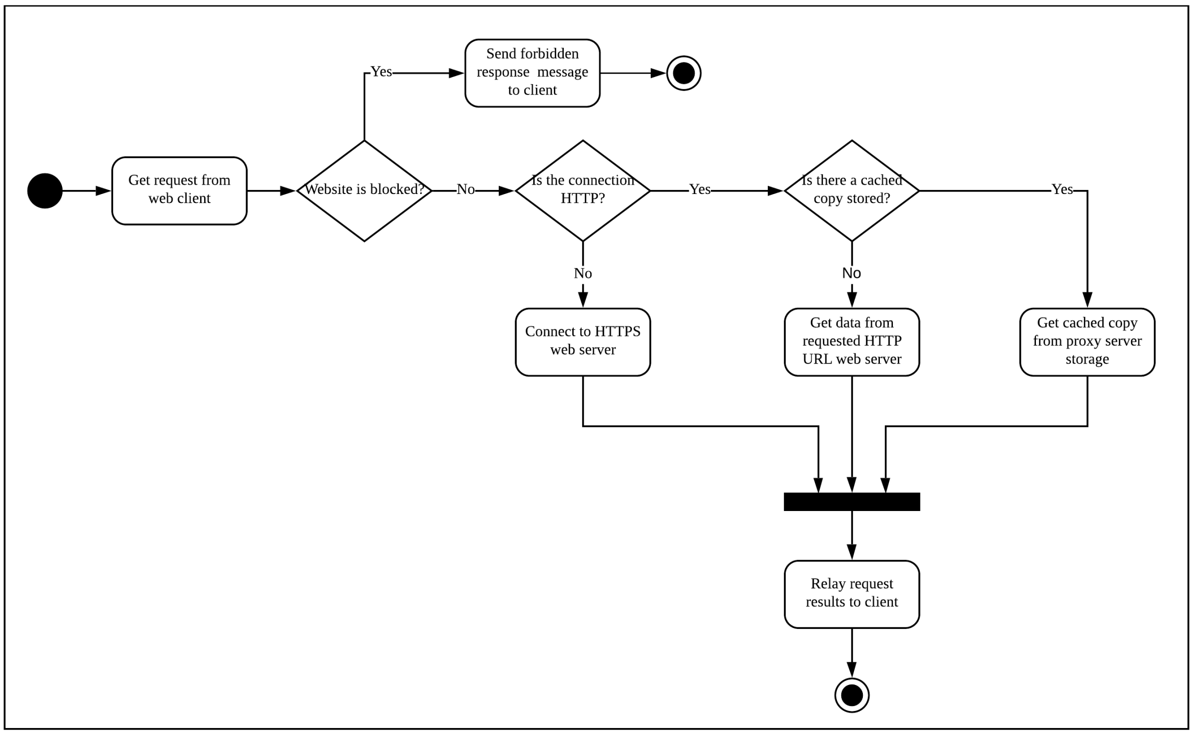
Additionally, every time when the proxy server is closed, the blocked website list and cached list has to be saved so that they are preserved the next time the proxy server is active. This can be done through serialization. Serialization is a data storage method that translates data structures or object state into a format by converting them into a stream of bytes so that they can be stored, transmitted and even reconstructed later. This allows us to preserve the state of the object when we recreate or de-serialize it the next time the proxy server is turned on.

Lastly, to enable web socket connections, the proxy server should be able to set up a special thread to handle HTTPS connections. To facilitate real-time data transfer to-and-from server due to interactions, this thread has to be able to take in continuous input stream of data from the web server socket, thus constructing a full-duplex communication between client and web server with the proxy server in between.

Proxy Server:

Flow chart:

Overall, the flow of the proxy server operations as designed above can be visualized in a flow chart with countable states connected by several decision boxes:



Send forbidden response message to client
Yes
Get request from web client
Website is blocked?
No
Is the connection HTTP?
Yes
Is there a cached copy stored?
Yes
No
No
Connect to HTTPS web server
Get data from requested HTTP URL web server
Get cached copy from proxy server storage
Relay request results to client

In general, when a proxy server receives a request from web client, it will run through up to a maximum of 3 decision boxes to decide on its approach for the request based on its needs. But they will all in the end converge into one state, which is the proxy server relaying the results of the request to the web client, even if the message to be conveyed is an indication that the website requested is forbidden by the administrator.

Implementation:

Setting up

This program make extensive use of TCP sockets provided in Java.net.sockets library to handle connections. The proxy web server is implemented using server sockets that is capable of accepting multiple socket connections from clients and handles them simultaneously,thus allowing multi-threading. When the web proxy server socket receive a connection request from a client socket, it creates a thread to handle their request and the thread is then put into a thread pool for concurrent management during the process. Every time when the proxy web server is turned on, it will load back the cached data and blocked websites into their respective data structures, as a method of continuing its from previous state.

To take in input request from and output messages as response to web clients and web servers, the proxy server make use of buffered readers and writers. Buffered Reader allows the web server to read lines of request and response from both sides and extract important information for further processing. On the other hand after the further processing, it will output the result of it using Buffered Writer, which is a versatile application in dealing with sending response code to sockets and writing data retrieved from socket output streams on a file.

Sending HTTP Request

There two situations that can happened when dealing with HTTP request, that is if there is a cached copy available for the requested URL site or not. To deal with situation where there are no cached copies available typically during compulsory cache misses, the proxy server first creates a file using the url host name as the file name and the url extension as the file type extension. Then, depending on the type of requested data, whether if is an image or a text based page, the proxy server will set up http url connection to forward the GET request to the site and attempt to extract its data and relay it to the web client. At the same time, it will also try to save a copy of the visited site using a separate buffer writer or ImageIO if it is an image to store a copy of the result data into the file created earlier. The web cache is implemented using HashMap with the url as key and the file as its corresponding values, due to its resemblance to a normal cache, where the url act as the memory address and the file act as the data stored at that particular address. Suppose if the request was a failure, then a response code of 404 NOT FOUND will be transmitted to the client, otherwise a response code of HTTP 200 will be transmitted instead. HTTPUrlConnection is being prioritized over using sockets here due to its versatility to extract even the last modified date of a web page that can be used to implement the proper functionality of a web cache.

Now, since there is a copy of the visited site preserved in the proxy server already, when the same site is being visited again, all that's needed to do is to find the file name of the saved copy in the cache using the url of the requested site as key to access the cache HashMap data structure. If there is hit, the contents of the file will be extracted using buffered reader and transmitted to client using buffered writer. This further reduces the time needed to transmit data as there's no need to set up a connection for retrieving data.

Sending HTTPS Request

Sending a HTTPS request can be tricky. This is because HTTPS connections make use of secure sockets (SSL) making data transfer between clients and servers encrypted. To overcome this problem, the proxy server needs to know how to deal with the encrypted data. The approach used in this context is is called HTTP Tunneling method, which allow us to send encrypted data over public network. This is how it works. When the proxy server receives a CONNECT Request, it attempts to extract the destination url from the inputs and created a socket connection to the remote web server. A response code of 200 is sent to client as confirmation of the connection being established. Now all there's left id to create a thread that allows the continuous transmissions of data between client and server through the proxy server simultaneously, thus enabling web socket connections application. However, since the data input and output stream are encrypted, it will be difficult to cache the data.

Administrator:

The proxy web server is set up in a way that only the administrator have the authority to manage its operation in real time. For example, the administrator has a higher privilege over the web clients, allowing him or her to block and unblock web pages. This allows him to have control access over data flowing in and out of the system, which is crucial to ensure the security of the proxy server. Moreover, the administrator can also shut down the web proxy server whenever he deems appropriate and thus terminating the request of the web clients to be served.

Blocking

As mentioned, the administrator has the authority to block websites. This can be done by entering the URL of websites where the proxy server will convert them into host names and store them into a hash set. One reason that hash set is chosen as a data structure in this context is because that it prevents duplicates host names from being stored in the list, thus no extra memory space is being wasted on duplicate copies of blocked host names.

Supposed a blocked website is being requested by a web client, instead of establishing a connection like the proxy server normally do, it will instead generate a forbidden access alert and output a denied access message to the web client, indicating that the requested website is blacklisted by the administrator due to security purposes. The forbidden response code would be 403.

Proxy Web Server shut down:

As mentioned above that the proxy server maintains copies of previously cached data and blocked websites, so it is crucial that during the shut down of the proxy server socket, these files are being saved somehow beforehand. The initiative in this situation is by serializing them. This allow us to convert the cache hash map and blocked list hash set into stream of bytes that can be easily loaded back the next time the proxy server is turned on again. The files are then stored in a $.ser$ file and thus the state of the proxy server is somehow preserved for future use. Upon shutting down, all the threads in the thread pool will be terminated even though they have not finished executing and the server socket is also closed to avoid unmonitored and unwanted influx of data.

Code:

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;
import java.util.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.net.URL;

public class ProxyServer implements Runnable{
        //initialise variables
        static HashSet<String> BlockedList;
        static HashMap<String, File> CachedList;
        static ArrayList<Thread> ActiveThreadsList;
        private volatile boolean active = true;
        private ServerSocket ProxySocket;

        // constructor for proxy server
        public ProxyServer(int portNumber, int maxQueue){
                //initialise data structures
                BlockedList = new HashSet<String>();
                ActiveThreadsList = new ArrayList<Thread>();
                CachedList = new HashMap<String,File>();

                //start server thread and socket afterwards
                Thread serverthread = new Thread(this);
                serverthread.start();
                try {
                        ProxySocket = new ServerSocket(portNumber
                           , maxQueue);
                        loadList();
                        active = true;
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                // set up data structure

        }

        // not needed currently only for improvement purposes
        class Wrapper{
                File file;
                long date;

                public Wrapper(File file, long date){
                        this.file = file;
                        this.date = date;
                }

                public File getFile(){
                        return this.file;
                }

                public long getDate(){
                        return this.date;
                }
        }

        // set up proxy server load back previous stored cached
           list and blocked ones as well
        private void loadList(){
                try{
                        //File file = new File("CachedURL.ser");
                        File file = new File("CachedURL.ser");
                        if(!file.exists()){
                                System.out.println("Error:␣
                                   CachedURL.ser␣does␣not␣exist")
                                   ;
                                file.createNewFile();
                        }

                        else{
                                FileInputStream fis = new
                                   FileInputStream(file);
                                ObjectInputStream ois = new
                                   ObjectInputStream(fis);
                                CachedList = (HashMap<String,File
                                   >)ois.readObject();
                                ois.close();
                                fis.close();
                        }

                        File file2 = new File("BlockedURL.ser");

                        if(!file2.exists()){
                                System.out.println("Error:␣
                                   BlockedURL.ser␣does␣not␣exist"
                                   );
                                file2.createNewFile();
                        }

                        else{
                                FileInputStream fis2 = new
                                   FileInputStream(file2);
```

```
            ObjectInputStream ois2 = new
              ObjectInputStream(fis2);
            BlockedList = (HashSet<String>)
              ois2.readObject();
            ois2.close();
            fis2.close();
        }

    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException io){
        io.printStackTrace();
    } catch (ClassNotFoundException cnf){
        cnf.printStackTrace();
    }
}
/*
 * might not need this method depending on the client
 */
public static void blockURl(String URL){
    BlockedList.add(URL);
    System.out.println(”Blocked␣URL␣List:”);
    Iterator<String> bList = BlockedList.iterator();
    while(bList.hasNext()){
        System.out.println(bList.next());
    }
}

// handle proper caching when new get request done
public static void caching(String URL, File file, long
  date){
    CachedList.put(URL, file);
}

public static boolean isBlocked(String url){
    String host=””;
    try {
        host = new URL(url).getHost();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    return BlockedList.contains(host);
}

// get cache webpage
public static File getCachedPage(String urlLink){
```

```
                System.out.println("Checking cache copy for " +
                    urlLink);
                URL url;
                long date = 0;
                try {
                        url = new URL(urlLink);
                        HttpURLConnection httpCon = (
                            HttpURLConnection) url.openConnection
                            ();

                    date = httpCon.getLastModified();
                } catch (MalformedURLException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

            if (date == 0){
                System.out.println("No last-modified information.
                    ");
                return CachedList.get(urlLink);
            }
            else{
              System.out.println("Last-Modified: " + new Date(
                  date));
                  return CachedList.get(urlLink);
            }
        }

        public static boolean hasCachedCopy(String URL){
                return CachedList.get(URL)!=null;
        }

        //caching webpages
        public static void cachePage(String URL, File file){
                CachedList.put(URL, file);
        }

        //close server
        private void closeProxyServer(){
                active = false;

                // save all blocked sites and cache sites to txt.
                    file
                try {
                        System.out.println("Saving cached URL
                            sites to CachedURL.ser ....");
                        FileOutputStream cachedFile = new
                            FileOutputStream("CachedURL.ser",false
                            );
                        ObjectOutputStream oos = new
                            ObjectOutputStream(cachedFile);
            oos.writeObject(CachedList);
            oos.close();
            cachedFile.close();
            System.out.println("Saving␣cached␣URL␣sites␣to␣
                CachedURL.ser␣completed.");


            System.out.println("Saving␣blocked␣URL␣sites␣to␣
                BlockedURL.ser␣....");
                        FileOutputStream blockedFile = new
                            FileOutputStream("BlockedURL.ser",
                            false);
                        ObjectOutputStream oos2 = new
                            ObjectOutputStream(blockedFile);
            oos2.writeObject(BlockedList);
            oos2.close();
            blockedFile.close();
            System.out.println("Saving␣blocked␣URL␣sites␣to␣
                BlockedURL.ser␣completed.");
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException io){
                        io.printStackTrace();
                }
                closeThreads();
                try {
                        // close server socket
                        ProxySocket.close();
                } catch (IOException e) {
                        System.out.println("Error:␣Server␣
                            Terminating␣Failed");
                        e.printStackTrace();
                }
                System.out.println("Proxy␣Server␣Connection␣
                    Terminated␣Successfully.");
        }

        private void closeThreads(){
                // close all remaining active threads
                for(Thread clientThread : ActiveThreadsList){
                        if(clientThread.isAlive()){
```

```
                                        System.out.println("Closing␣
                                           thread" + clientThread.getId()
                                           );
                                        try {
                                                clientThread.join();
                                        } catch (InterruptedException e)
                                           {
                                                // TODO Auto-generated
                                                   catch block
                                                e.printStackTrace();
                                        }
                                }
                        }
                }

                // create pools of threads and listens to them
                public void listen(){
                        while(active){
                                try {
                                        Socket client = ProxySocket.
                                           accept();
                                        System.out.println(client.getPort
                                           ());
                                        Thread serviceThread = new Thread
                                           (new ManagementConsole(client)
                                           );
                                        ActiveThreadsList.add(
                                           serviceThread);
                                        serviceThread.start();
                                } catch (SocketException e) {
                                        // Socket exception is triggered
                                           by management system to shut
                                           down the proxy
                                        System.out.println("Server␣closed
                                           ");
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }

                        }
                }

                // console for administrator to make changes
                @Override
                public void run() {
                        Scanner InputScanner = new Scanner(System.in);
                        String instruction;
```

```
        while(active){
                System.out.println("How␣can␣I␣help␣you?")
                  ;
                instruction = InputScanner.nextLine();
                //handle commands

                if(instruction.equalsIgnoreCase("BL")){
                        System.out.println("Blocked␣URL␣
                          List:");
                        Iterator<String> bList =
                          BlockedList.iterator();
                        while(bList.hasNext()){
                                System.out.println(bList.
                                  next());
                        }
                }

                else if(instruction.equalsIgnoreCase("
                  Quit")){
                        System.out.println("Closing␣Proxy
                          ␣Server......");
                        closeProxyServer();
                }

                else if(instruction.substring(0,5).
                  equalsIgnoreCase("Block")){
                        String host="";
                        try {
                                host = new URL(
                                  instruction.substring
                                  (6)).getHost();
                                System.out.println(host);
                        } catch (MalformedURLException e)
                          {
                                // TODO Auto-generated
                                  catch block
                                e.printStackTrace();
                        }
                        if(!host.isEmpty()){
                                BlockedList.add(host);
                                System.out.println("
                                  Blocking␣" + host);
                        }
                }

                // unblock site
                else if(instruction.substring(0,7).
                  equalsIgnoreCase("Unblock")){
```

```
                                        String host="";
                                        try {
                                                host = new URL(
                                                   instruction.substring
                                                   (8)).getHost();
                                                System.out.println(host);
                                        } catch (MalformedURLException e)
                                            {
                                                // TODO Auto-generated
                                                   catch block
                                                e.printStackTrace();
                                        }
                                        if(!host.isEmpty()){
                                                BlockedList.remove(host);
                                                System.out.println("
                                                   Blocking␣" + host);
                                        }
                                }
                        }
                        InputScanner.close();
                }


                public static void main(String[] args){
                        ProxyServer proxy = new ProxyServer(4000, 20);
                        proxy.listen();
                }

}
```

Weakness:

A flaw present in the system designed is that the cache implemented does not check if the copy supplied during HTTP request is the most updated version as present in the original web server. This causes potential wrong information to be supplied to web clients as the program only checks if the there is cached copy but not if it has been modified after the copy was first saved and stored in the data storage section.

Moreover, a cache should operate in a way such that it stores only either the most recently or frequently accessed items depending on the proxy server needs. this is mainly due to the limited memory space it has. Therefore, the implementation of cache using only a Hash Map is obviously not an appropriate choice here because there is no way to trace for data that haven't been accessed for a long time and seek to remove them in order to clear up space in the cache.

Improvements:

To solve the problems above, we can formulate alternatives that are specifically designed to tackle each of them. Firstly, by using the library class HttpURLConnection as provided in Java programming language, it allows us to extract various information such as host names, response code, certificates, and even last modified date of a web page. Utilizing this advantage, we can propose checks that set up a url connection to the requested website when a cached copy is requested. This allow us to compare the last modified date retrieved and the current modified date stored along with the files to ensure the validity of our copy. If the copy possessed is out of date, then the proxy server simply fetches a new one and display it to the client and as well as renewing its copy. That being said the file copy should now be stored in a wrapper class along with the last modified date for easier access.

Moreover, in order to properly simulate the operations of a cache, a Least Recently Used(LRU) queue class can be implemented to overcome the shortcomings of Hash Map in this context. A queue can be set to a limit size to prevent excess data from being stored on the proxy server, which could be detrimental to its processing speed. At any point, when the limit size is exceeded, the cache will begin dequeuing data on the head to clear up space for incoming data.

Request Handler(Management Console in code due to naming error):

Implementation:

Request Handler

The Request Handler is implemented to take in request from web clients and handles them properly based on types of request such as GET and CONNECT.

Management Console

The management console in the form of a printer associated with operations of the proxy server, allowing it to print out all the traffic data of each execution of threads' requests.

Code:

```java
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.net.URL;
import java.security.cert.Certificate;

import javax.imageio.ImageIO;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLPeerUnverifiedException;

public class ManagementConsole implements Runnable{
        //initialize variables
        Socket client;
        BufferedReader clientReq;
        BufferedWriter serverRes;
        private Thread ClientToServerHTTPSConnection;
```

```
    // suppose to be request handler but can't change name as
       file was created
    public ManagementConsole(Socket clientSocket){
            this.client = clientSocket;
            try{
                    this.client.setSoTimeout(3000);
                    clientReq = new BufferedReader(new
                       InputStreamReader(client.
                       getInputStream()));
                    serverRes = new BufferedWriter(new
                       OutputStreamWriter(client.
                       getOutputStream()));

            } catch (SocketTimeoutException e) {
                    System.out.println("Socket␣" + client.
                       getLocalPort() + "␣Error:␣Connection␣
                       TimeOut");
        e.printStackTrace();
            } catch (IOException i) {
                    i.printStackTrace();
            }
    }

    // creates a thread to enable continious stream of data
      flow
    class HTTPSTransmission implements Runnable{

            // input stream from proxy to client
            InputStream is;
            // output stream from proxy to server
            OutputStream os;
            int bufferSize;

            /**
             * Creates Object to Listen to Client and
               Transmit that data to the server
             * @param proxyToClientIS Stream that proxy uses
               to receive data from client
             * @param proxyToServerOS Stream that proxy uses
               to transmit data to remote server
             */
            public HTTPSTransmission(InputStream
               proxyToClientIS, OutputStream proxyToServerOS)
                 {
                    this.is = proxyToClientIS;
                    this.os = proxyToServerOS;
                    bufferSize = 4096;
        }

        @Override
        public void run(){
                try {
                        // Read byte by byte from client
                           and send directly to server
                        byte[] buffer = new byte[
                           bufferSize];
                        int read;
                        do {
                                read = is.read(buffer);
                                if (read > 0) {
                                        os.write(buffer,
                                           0, read);
                                        if (is.available
                                           () < 1) {
                                                os.flush
                                                   ();
                                        }
                                }
                        } while (read >= 0);
                }
                catch (SocketTimeoutException e) {
                        e.printStackTrace();
                }
                catch (IOException io) {
                        System.out.println("Buffer␣Read␣
                           TimeOut␣from␣Proxy␣to␣Client")
                           ;
                        io.printStackTrace();
                }
        }
}

// error message displayed to client when blocked page
   accessed
private void blockedURLRequested(String URL){
        try {
                String message = "Access␣to␣" + URL + "␣
                   denied␣message:␣\n" +
                                                           "HTTP
                                                              /1.0␣
                                                              403␣
                                                              Access
                                                              ␣
                                                              Forbidden
                                                              \n" +
                                                           "User−
                                                              Agent
                                                              :␣
                                                              ProxyServer
                                                              /1.0\
                                                              n" +
                                                              "\r\n
                                                              ";
                serverRes.write(message);
                serverRes.flush();
        } catch (IOException e) {
                e.printStackTrace();
        }

}


// application to print certs of https site
// possible if using https connection
private void printCertificate(String https_url){
        try {
                URL url = new URL(https_url);
                HttpsURLConnection con = (
                   HttpsURLConnection)url.openConnection
                   ();
            if(con!=null){

               try {

                                System.out.println("
                                   Response␣Code␣:␣" +
                                   con.getResponseCode())
                                   ;
                                System.out.println("
                                   Cipher␣Suite␣:␣" + con
                                   .getCipherSuite());
                                System.out.println("\n");

                                Certificate[] certs = con
                                   .getServerCertificates
                                   ();
                                for(Certificate cert :
                                   certs){
                                   System.out.println("
                                      Cert␣Type␣:␣" +
                                      cert.getType());
                                   System.out.println("
                                      Cert␣Hash␣Code␣:␣"
```

```
                                                + cert.hashCode());
                                             System.out.println("
                                                Cert␣Public␣Key␣
                                                Algorithm␣:␣"
                                                                                 +
                                                                                    c
                                                                                    .
                                                                                    g
                                                                                    ()
                                                                                    .
                                                                                    g
                                                                                    ()
                                                                                    )
                                                                                    ;
                                             System.out.println("
                                                Cert␣Public␣Key␣
                                                Format␣:␣"
                                                                                 +
                                                                                    c
                                                                                    .
                                                                                    g
                                                                                    ()
                                                                                    .
                                                                                    g
                                                                                    ()
                                                                                    )
                                                                                    ;
                                             System.out.println("\n
                                                ");
                                  }

                                  } catch (
                                     SSLPeerUnverifiedException e)
                                     {
                                          e.printStackTrace();
                                  } catch (IOException e){
                                          e.printStackTrace();
                                  }
                        }
                  } catch (MalformedURLException e) {
                          e.printStackTrace();
            } catch (IOException e) {
                          e.printStackTrace();
            }
```

```
        }

        // printing cert using httpssslurlconnection
        private void connectHTTPSRequest(String urlLink){
                String url = urlLink.substring(7);
                String split[] = url.split(":");
                url = split[0];
                // print certificate of https website
                printCertificate("https://" + url);
                int port  = Integer.valueOf(split[1]);

                try{
                        // Only first line of HTTPS request has
                           been read at this point (CONNECT *)
                        // Read (and throw away) the rest of the
                           initial data on the stream
                        for(int i=0;i<5;i++){
                                clientReq.readLine();
                        }

                        // Get actual IP associated with this URL
                            through DNS
                        InetAddress linkAddr = InetAddress.
                           getByName(url);

                        // Open a socket for proxy to the remote
                           server
                        Socket socket = new Socket(linkAddr, port
                           );
                        socket.setSoTimeout(10000);

                        // Send Connection established to the
                           client
                        String res = "HTTP/1.0␣200␣Connection␣
                           established\r\n" +
                                                    "Proxy-Agent:␣
                                                       ProxyServer
                                                       /1.0\r\n" +
                                                    "\r\n";

                        serverRes.write(res);
                        serverRes.flush();



                        // Client and Remote will both start
                           sending data to proxy at this point
```

```
// Proxy needs to asynchronously read
   data from each party and send it to
   the other party


//Create a Buffered Writer betwen proxy
   and remote
BufferedWriter proxyToServerBW = new
   BufferedWriter(new OutputStreamWriter(
   socket.getOutputStream()));

// Create Buffered Reader from proxy and
   remote
BufferedReader proxyToServerBR = new
   BufferedReader(new InputStreamReader(
   socket.getInputStream()));



// Create a new thread to listen to
   client and transmit to server
HTTPSTransmission clientToServerHttps =
                 new HTTPSTransmission(
                    client.getInputStream
                    (), socket.
                    getOutputStream());

ClientToServerHTTPSConnection = new
   Thread(clientToServerHttps);
ClientToServerHTTPSConnection.start();
int bufferSize = 4096;

// Listen to remote server and relay to
   client
try {
        byte[] buffer = new byte[
           bufferSize];
        int read;
        do {
                read = socket.
                   getInputStream().read(
                   buffer);
                if (read > 0) {
                        client.
                           getOutputStream
                           ().write(
                           buffer, 0,
                           read);
```

```
                                                      if (socket.
                                                         getInputStream
                                                         ().available()
                                                          < 1) {
                                                              client.
                                                                 getOutputStream
                                                                 ().
                                                                 flush
                                                                 ();
                                                      }
                                              }
                                      } while (read >= 0);
                              }
                              catch (SocketTimeoutException e) {
                                      System.out.println("Socket␣Time␣
                                         Out␣Error");
                                      e.printStackTrace();
                              }
                              catch (IOException e) {
                                      e.printStackTrace();
                              }


                              // Close Down Resources
                              if(socket != null){
                                      socket.close();
                              }

                              if(proxyToServerBR != null){
                                      proxyToServerBR.close();
                              }

                              if(proxyToServerBW != null){
                                      proxyToServerBW.close();
                              }

                              if(serverRes != null){
                                      serverRes.close();
                              }


                      } catch (SocketTimeoutException e) {
                              String line = "HTTP/1.0␣504␣Timeout␣
                                 Occured␣after␣10s\n" +
                                              "User-Agent:␣ProxyServer
                                                 /1.0\n" +
                                              "\r\n";
                              try{
                                serverRes.write(line);
                                serverRes.flush();
                        } catch (IOException ioe) {
                                ioe.printStackTrace();
                        }
                }
                catch (Exception e){
                        System.out.println("Error␣on␣HTTPS␣:␣" +
                           urlLink );
                        e.printStackTrace();
                }
        }

        // handling https request
        private void HandleGETRequestWithCachedCopy(String
           urlLink){
                // get file from cache
                File file = ProxyServer.getCachedPage(urlLink);
                //try to get the file type
                String fileType = file.getName().substring(file.
                   getName().lastIndexOf("."));

                try{
                        // if file is image
                        if((fileType.contains(".jpeg")) ||
                           fileType.contains(".jpg") ||
                                        fileType.contains(".gif")
                                           || fileType.contains(
                                          ".png")){

                                BufferedImage img = ImageIO.read(
                                   file);

                                // check if getting results to
                                   send response code to client
                                if(img!=null){

                                        String res = "HTTP/1.0␣
                                           200␣OK\n" +
                                                                    "
                                                                       Proxy
                                                                       -
                                                                       agent
                                                                       :
                                                                       ␣
                                                                       ProxyS
                                                                       /1.0\
                                                                       n
```

```
                                                                   ”

                                                                   +

                                                                 ”
                                                                   \
                                                                   r
                                                                   \
                                                                   n
                                                                   ”

                                                                   ;


                                  serverRes.write(res);
                                  serverRes.flush();
                                  System.out.println(”Image
                                     ␣retrieval␣SUCCESS.”);
                                  ImageIO.write(img,
                                     fileType.substring(1),
                                      client.
                                     getOutputStream());
                             }

                             // image read failed or not
                                received from URL link server
                             else{
                                  String res = ”HTTP/1.0␣
                                     404␣NOT␣FOUND\n” +
                                                                ”
                                                                  Proxy
                                                                  −
                                                                  agent
                                                                  :
                                                                  ␣
                                                                  ProxySe
                                                                  /1.0\
                                                                  n
                                                                  ”

                                                                  +

                                                                ”
                                                                  \
                                                                  r
                                                                  \
                                                                  n
                                                                  ”

                                                                  ;
```

```
                serverRes.write(res);
                serverRes.flush();
                System.out.println("
                  Cached␣Image␣retrieval
                  ␣FAILED.");
                return;
            }
        }

        // if file is text based
        else {
            BufferedReader br = new
              BufferedReader(new
              InputStreamReader(new
              FileInputStream(file)));

            // send response code
            String res = "HTTP/1.0␣200␣OK\n"
              +
                        "Proxy-
                          agent
                          :␣
                          ProxyServer
                          /1.0\
                          n" +
                        "\r\n";
            serverRes.write(res);
            serverRes.flush();

            String line;
            while((line = br.readLine()) !=
              null){
                serverRes.write(line);
            }
            serverRes.flush();

            // Close resources
            if(br != null) br.close();
        }

        if(serverRes!=null)serverRes.close();

    } catch (IOException e) {
        System.out.println("Error␣Sending␣Cached␣
          file␣to␣client");
        e.printStackTrace();
    }
}
```

```
    private void HandleGETRequestWithoutCachedCopy(String
        urlLink){

            // separate file name and file extension type
            String fileName = urlLink.substring(0,urlLink.
                lastIndexOf("."));
            String fileType = urlLink.substring(urlLink.
                lastIndexOf("."), urlLink.length());

            // trim off the http://www.
            fileName = fileName.substring(fileName.indexOf(".
                ")+1);

            // remove illegal characters for filenames
            fileName = fileName.replace(".", "_");
            fileName = fileName.replace("/", "__");

            // replace the trailing / to .html as file
                extension type for saving
            if(fileType.contains("/")){
                    fileType = fileType.replace(".", "_");
                    fileType = fileType.replace("/", "__");
                    fileType = fileType + ".html";
            }

            // combine the currect filename and extension
                type
            fileName = fileName + fileType;
            System.out.println("filename created: " +
                fileName);

            // check if we successfully created a file
            boolean caching = true;
            File file = null;
            BufferedWriter filebw = null;

            try{
                    // create an abstract pathname to store
                        the cache
                    file = new File("cache_files/"+fileName);

                    // check if there is a previous version
                        of cached file stored
                    if(!file.exists()){
                            file.createNewFile();
                    }
                // writer for copying output stream from
                   GET request into file
                filebw = new BufferedWriter(new
                   FileWriter(file));
        } catch(IOException io){
                caching = false;
                System.out.println("Create␣File␣Error,␣
                   Aborting␣Caching");
                io.printStackTrace();
        }

        try{
                // handling image files
                if((fileType.contains(".jpeg")) ||
                   fileType.contains(".jpg") ||
                                  fileType.contains(".gif")
                                     || fileType.contains(
                                    ".png")){

                        URL url = new URL(urlLink);
                        BufferedImage img = ImageIO.read(
                           url);

                        // check if getting results to
                           send response code to client
                        if(img!=null && file!=null){
                                ImageIO.write(img,
                                   fileType.substring(1),
                                    file);
                                String res = "HTTP/1.0␣
                                   200␣OK\n" +
                                                          "
                                                            Proxy
                                                            —
                                                            agent
                                                            :
                                                            ␣
                                                            ProxySe
                                                            /1.0\
                                                            n
                                                            "

                                                            +

                                                          "

                                                            \
                                                            r
                                                            \
                                                              n
                                                              ”
                                                              ;
                    serverRes.write(res);
                    serverRes.flush();
                    System.out.println("Image
                       ␣retrieval␣SUCCESS.");
                    ImageIO.write(img,
                       fileType.substring(1),
                        client.
                       getOutputStream());
                }

                // image read failed or not
                   received from URL link server
                else{
                    String res = "HTTP/1.0␣
                       404␣NOT␣FOUND\n" +
                                                             ”
                                                              Proxy
                                                              —
                                                              agent
                                                              :
                                                              ␣
                                                              ProxySe
                                                              /1.0\
                                                              n
                                                              ”

                                                              +

                                                             ”
                                                              \
                                                              r
                                                              \
                                                              n
                                                              ”
                                                              ;
                    serverRes.write(res);
                    serverRes.flush();
                    System.out.println("Image
                       ␣retrieval␣FAILED.");
                    return;
                }
            }
```

```
                        // handling pure text files or html
                        else{
                                // create connection using url
                                URL url = new URL(urlLink);
                                HttpURLConnection con = (
                                   HttpURLConnection)url.
                                   openConnection();

                                // setting properties for
                                   connection
                                con.setUseCaches(false);
                                con.setDoOutput(true);
                                con.setRequestProperty("Content-
                                   Type",
                                                "application/x-
                                                   www-form-
                                                   urlencoded");
                                con.setRequestProperty("Content-
                                   Language", "en-US");

                                // print response code
                            if(con!=null){

                                try {
                                                System.out.
                                                   println("
                                                   Response␣Code␣
                                                   :␣" + con.
                                                   getResponseCode
                                                   ());



                                            } catch (
                                               SSLPeerUnverifiedException
                                               e) {
                                                e.printStackTrace
                                                   ();
                                            } catch (IOException e){
                                                e.printStackTrace
                                                   ();
                                            }
                            }

                                BufferedReader in = new
                                   BufferedReader(new
                                   InputStreamReader(con.
                                 getInputStream()));
                                String res = "HTTP/1.0 200 OK\n"
                                  +
                                              "Proxy-agent: 
                                                 ProxyServer
                                                 /1.0\n" +
                                              "\r\n";
                                serverRes.write(res);

                                String line;
                    StringBuilder content = new StringBuilder();

                    while ((line = in.readLine()) != null) {
                        content.append(line);
                        content.append(System.lineSeparator());
                        serverRes.write(line);

                        if(caching)      filebw.write(line);
                    }
                    // print traffic data in console
                    System.out.println("no of bytes:" + content.
                       toString().length());
                    System.out.println(content.toString());

                    // close buffered reader after operation
                    if(in!=null) in.close();
                        }

                        if(caching){
                                filebw.flush();
                                ProxyServer.cachePage(urlLink,
                                   file);
                        }

                        //close down resources
                        if(filebw!=null)filebw.close();
                        if(serverRes!=null)serverRes.close();
                }catch (MalformedURLException e){
                e.printStackTrace();
            } catch (ProtocolException p){
                p.printStackTrace();
            } catch(IOException i){
                i.printStackTrace();
            }
        }

        //run the thread request handler
        @Override
```

```
    public void run(){
            String input;
            try {
                    // read inputs
                    input = clientReq.readLine();
                    System.out.println("Socket␣" + client.
                       getLocalPort() + "␣Request:␣" + input)
                       ;
            } catch (IOException e) {
                    e.printStackTrace();
                    System.out.println("Socket␣" + client.
                       getLocalPort() +"␣ERROR:␣Can't␣read␣
                       input␣requests␣from␣client");
                    return;
            }

            // get url n its type
            String[] split = input.split("␣");
            String req = split[0];
            String url = split[1];

            if(!url.substring(0,4).equals("http")){
                    url = "http://" + url;
            }
            try {
                    System.out.println("host:␣" +new URL(url)
                       .getHost());
            } catch (MalformedURLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }

            // check if the site is blocked, send forbidden
               message
            if (ProxyServer.isBlocked(url)){
                    System.out.println(url + "␣site␣
                       connection␣is␣blocked");
                    blockedURLRequested(url);
                    return;
            }

            // if not blocked
            else{
                    // send https connection
                    if(req.equals("CONNECT")){

                            System.out.println("Connecting␣to
                               :␣" +url);
```

```
                                connectHTTPSRequest(url);
                        }

                        // send http connection
                        else{
                                // handle connection with cached
                                  copy
                                if(ProxyServer.hasCachedCopy(url)
                                  ){
                                        System.out.println("Cache
                                          ␣copy␣found␣for␣HTTP␣
                                          GET" + url);
                                        long currentTime = System
                                          .currentTimeMillis();
                                        HandleGETRequestWithCachedCopy
                                          (url);
                                        long diff = System.
                                          currentTimeMillis() −
                                          currentTime;
                                        System.out.println("With␣
                                          Cache:␣" + diff);
                                }
                                // establish http connection
                                  since no copy for requested
                                  site
                                else{
                                        System.out.println("Cache
                                          ␣copy␣not␣found␣for␣
                                          HTTP␣GET" + url);
                                        long currentTime = System
                                          .currentTimeMillis();
                                        HandleGETRequestWithoutCachedCopy
                                          (url);
                                        long diff = System.
                                          currentTimeMillis() −
                                          currentTime;
                                        System.out.println("
                                          Without␣Cache:␣" +
                                          diff);
                                }
                        }
                }
        }


}
```

Weakness:

The traffic data displayed on Request Handler can only be differentiated between normal data and exceptions raised. This makes debugging abnormal problems difficult as there are way too many data flowing through the proxy web server in a continuous stream.

Improvements:

One approach in dealing with such situations is to assigned different color codes to threads based on their request execution conditions. The colors will represent the status and state of the request being executed and indicate whether the result returned is positive or not using ratings. Such method will save more time for testers to figure out which request generated is facing problems and be sorted out early to avoid congestion in the traffic data pool.