ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF MECHANICAL AND ELECTRICAL ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

NETWORK AND INFORMATION SECURITY ASSIGNMENT

HTTP JAVA PROXY

MENTESNOT ERTIBU FTP 0708/11

Proxy servers

In the broadest term a proxy server is A proxy server is an intermediary server that acts as a gateway between a client device (such as a computer, tablet, or smartphone) and another server or network. When a user requests a resource (such as a web page or a file) from the Internet, their device sends the request to the proxy server instead of directly to the destination server.

The proxy server then evaluates the request, forwards it to the destination server on behalf of the client device, receives the response, and sends it back to the client device. This way, the client device remains anonymous to the destination server, as the destination server sees the request as coming from the proxy server, not the client device.

There are several types of proxy servers, including:

Web proxies: These are the most common type of proxy servers, and they are used primarily to access websites that are restricted by network filters or firewalls. Web proxies act as an intermediary between the client device and the destination server, allowing users to access restricted websites without revealing their identity.

Web proxies are also known as HTTP proxies, as they primarily handle web traffic using the HTTP protocol. They work by intercepting web requests from client devices and forwarding them to the destination server on behalf of the client. When the response is received, the proxy server sends it back to the client device. Web proxies are often used to bypass content filters or censorship, to hide browsing activity from network administrators, or to access content that is geographically restricted.

Some web proxies are free to use, while others require a paid subscription. Free web proxies may be less reliable, may not offer as many features or configuration options, and may be more susceptible to being blocked by destination servers or network filters.

Anonymous proxies: These proxies are designed to conceal the client device's IP address and other identifying information, making it difficult for websites to track or trace the user's activity.

Anonymous proxies are designed to conceal the user's identity and protect their privacy. They work by masking the user's IP address and other identifying information, making it difficult for websites to track or trace the user's activity. Anonymous proxies may be used for legitimate purposes, such as protecting personal information while browsing the Internet, or for more nefarious activities, such as evading law enforcement or carrying out cyber attacks.

There are several types of anonymous proxies, including high-anonymity proxies, which conceal the user's IP address and other identifying information completely, and transparent proxies, which disclose some information about the user's identity.

Reverse proxies: These proxies are used by web servers to receive requests from the Internet and forward them to the appropriate server within the local network. Reverse proxies can improve website performance by caching frequently requested content and distributing traffic across multiple servers.

Transparent proxies: These proxies are used primarily by network administrators to monitor and filter traffic within a network. Transparent proxies intercept all traffic and can be used to block access to specific websites or filter out unwanted content.

Transparent proxies may be less effective at protecting user privacy, as they may not conceal the user's IP address or other identifying information. Users may also be able to bypass transparent proxies using techniques such as using a VPN or accessing the Internet through a different network.

With all their similarities virtual private networks (VPN) get mistaken for proxy servers, so to clear this misconception I will be explaining what VPNs are and how they fundamentally differ from proxy servers.

At its core, a VPN uses a combination of encryption and tunneling technologies to create a secure connection between a user's device and a remote server. When a user connects to a VPN, their internet traffic is routed through an encrypted tunnel to the remote server, which acts as a proxy for the user's online activity. This means that any data transmitted between the user's device and the internet is protected by strong encryption, making it more difficult for hackers, government agencies, and other third parties to intercept or monitor the data.

The encryption used by VPNs is typically based on advanced cryptographic protocols, such as AES (Advanced Encryption Standard), which is considered one of the strongest encryption algorithms available. This ensures that the data transmitted through the VPN is highly secure and difficult to decrypt, even if intercepted by a third party.

Tunneling technology is also a key component of VPNs. VPNs create a virtual tunnel between the user's device and the remote server, which helps to ensure that the user's online activity is protected from prying eyes. Tunneling protocols used by VPNs include PPTP (Point-to-Point Tunneling Protocol), L2TP/IPSec (Layer 2 Tunneling Protocol/Internet Protocol Security), and OpenVPN, each with their own strengths and weaknesses.

When computers are connected to a Virtual Private Network (VPN), they can communicate with each other in much the same way as computers on a local area network (LAN). This is made possible by the use of several key technologies that allow the VPN to mimic the functionality of a LAN network.

One of the most important technologies used by VPNs is the use of a private IP address space. When a user connects to a VPN, they are assigned an IP address from the VPN's private IP address space. This means that all the devices on the VPN are assigned IP addresses that are only visible within the VPN network, and are not visible to devices outside the VPN. This allows the devices on the VPN to communicate with each other as if they were on a LAN network.

Another important technology used by VPNs is the use of tunneling protocols. Tunneling protocols create a secure, encrypted tunnel between a user's device and the VPN server, which helps to protect data transmitted between the user's device and the VPN. This tunnel also allows the VPN server to act as a gateway, enabling devices on the VPN to communicate with each other even if they are located in different geographic locations.

In addition to these technologies, many VPNs also provide features that are similar to those found on LAN networks. For example, many VPNs include features such as network file sharing, printer sharing, and remote access to network resources. These features help to ensure that users on the VPN can work together and access the same resources in a way that is similar to how they would if they were on a LAN network.

Overall, the ability of computers on a VPN to communicate like computers on a LAN network is made possible by the use of private IP address spaces, tunneling protocols, and other networking technologies that allow the VPN to mimic the functionality of a LAN network. These technologies help to ensure that users on the VPN can work together and access the same resources in a secure and private environment, even if they are located in different geographic locations.

Building a proxy server in JAVA

In this report, we will discuss the design and implementation of a Java-based http only proxy server. Our proxy server is designed to route http requests through it. It can handle incoming requests from clients, route traffic to the appropriate destination, and return the response to the client. We will also discuss the features and capabilities of our proxy server.

Overall, our Java-based proxy server represents an important solution to the problem of internet security. It provides users with a secure and reliable way to access online resources, while also protecting their sensitive data from prying eyes. Through this report, we hope to provide a detailed understanding of our proxy server, its features, and its capabilities

The server listens on port 8085 for incoming client connections. When a client sends a request, the server parses the string of the request into host, port, and specific route of the request. It then establishes a connection with the host and sends the request for the specific route. When the response is received from the host, it is sent back to the client by writing the binary information on a buffer.

```
byte [] buffer = new byte[4096];
            int bytes_read;
            while ((bytes_read = netIn.read(buffer)) != -1)
            {
            clientOut.write(buffer, 0, bytes_read);
            firefox.write(buffer, 0, bytes_read);
            firefox.flush();
            clientOut.flush();
            }
```

In here clinetOut is the object communication with the standard input output STDOUT. It is just there for debugging purposes and writes the binaries on the screen for us. Firefox object is the object declared upon a client (which in usual circumstances is a web browser) connecting with our proxy server.

```
   public static void main (String args []) {
        Client myProxy = new Client(8085);
        myProxy.listen();

}
```

```
      public Client (int port) {
       new Thread(this).start();
       try {
           // Create the Server Socket for the Proxy
           serverSocket = new ServerSocket(port);
```

```
            System.out.println("Waiting for client on port " +
serverSocket.getLocalPort() + "..");
            running = true;
        }
```

This is the constructor to class Client

```
    public void listen(){
        while(running){

            try {
                // serverSocket.accpet() Blocks until a connection is
made
                Socket socket = serverSocket.accept();

                // Create new Thread and pass it Runnable
RequestHandler
                Thread thread = new Thread(new Handler(socket));


                thread.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();
            }
        }
    }
```

listen() method of class client

Handler class is what's going to handle the communication between the client(a web browsers) and our proxy server from this point on.

```
   public Handler (Socket socket) {
        this.socket = socket;
        try{
            this.socket.setSoTimeout(4000);
            clientReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            clientWriter = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
            firefox = new
DataOutputStream(this.socket.getOutputStream());
        }
        catch (IOException e) {
            e.printStackTrace();
        }
   }
```

Handler Class constructor

clientReader and clientWriter were implementations of communication with the web browser using readline() since we pivoted into writing binaries into a buffer readline wouldn't work. We went with clientReader.readLine() to read requests incoming from the browser and with object firefox we were able to pass buffered streams to the client.

```
        String request;
        // String request = "GET http://www.aastu.edu.et/index.html
HTTP/1.0\r\n";
        try{
            request = clientReader.readLine();
        }
        catch (IOException e) {
            System.out.println("error minamn : "+e.getMessage());
```

```
            return;
        }
```

```
String[] components = request.split("\\s+");
            String requestType = components[0];
            String requestRoute = components[1];
            String protocolVersion = components[2];

            // Extract the host from the request route
            int hostStart = requestRoute.indexOf("://") + 3;
            int hostEnd = requestRoute.indexOf("/", hostStart);
            String host = requestRoute.substring(hostStart, hostEnd);

            // Extract the request route from the request route
            int routeStart = requestRoute.indexOf("/", hostEnd);
            String route = requestRoute.substring(routeStart);

            // Print the parsed components
            System.out.println("Request Type: " + requestType);
            System.out.println("Host: " + host);
            System.out.println("Request Route: " + route);
        try (Socket socket = new Socket(host, 80)) {
            OutputStream output = socket.getOutputStream();
            System.out.println("Are you getting here?");
            InputStream input = socket.getInputStream();
            DataOutputStream dout = new DataOutputStream(output);
```

String requested is parsed into different parts of what makes up a get request. And then we establish a TCP connection with the host on port 80.

During the presentation I had a weird problem where it seemed to only work on the AASTU website and even on it it was acting very strangely. That's partly attributed to me debugging the

string parser on the AASTU website and brute forcing my way into what I thought at the time was a solution.

```
try (Socket socket = new Socket(host, 80)) {
            OutputStream output = socket.getOutputStream();
            System.out.println("Are you getting here?");
            InputStream input = socket.getInputStream();
            DataOutputStream dout = new DataOutputStream(output);
            // requestType+"/"+file+"HTTP/1.0\r\n";
            // dout.writeBytes("GET
"+"/"+matcher.group(2)+"HTTP/1.0\r\n");
            // dout.writeBytes("GET /
HTTP/1.0\r\nHost:165.232.112.241\r\n\r\n");
            dout.writeBytes("GET "+route+"HTTP/1.0\r\n");
```

Initially i did that but upon debugging with the AASTU website i decide to go with the following :

```
      try (Socket socket = new Socket(host, 80)) {
            OutputStream output = socket.getOutputStream();
            System.out.println("Are you getting here?");
            InputStream input = socket.getInputStream();
            DataOutputStream dout = new DataOutputStream(output);
            // requestType+"/"+file+"HTTP/1.0\r\n";
            // dout.writeBytes("GET
"+"/"+matcher.group(2)+"HTTP/1.0\r\n");
            // dout.writeBytes("GET /
HTTP/1.0\r\nHost:165.232.112.241\r\n\r\n");
            // dout.writeBytes("GET "+route+"HTTP/1.0\r\n");
            dout.writeBytes("GET "+route+"HTTP/1.0\r\n\r\n");
            //dout.writeBytes("GET "+route+" HTTP/1.0\r\nHost:
"+host+"\r\n\r\n");
            // System.out.println("GET "+route+"HTTP");
```

Finally this seemed to do the trick and my proxy now works on any http site

```
        try (Socket socket = new Socket(host, 80)) {
            OutputStream output = socket.getOutputStream();
            System.out.println("Are you getting here?");
            InputStream input = socket.getInputStream();
            DataOutputStream dout = new DataOutputStream(output);
            // requestType+"/"+file+"HTTP/1.0\r\n";
            // dout.writeBytes("GET
"+"/"+matcher.group(2)+"HTTP/1.0\r\n");
            // dout.writeBytes("GET /
HTTP/1.0\r\nHost:165.232.112.241\r\n\r\n");
            // dout.writeBytes("GET "+route+"HTTP/1.0\r\n");
            // dout.writeBytes("GET "+route+"HTTP/1.0\r\n\r\n");
            dout.writeBytes("GET "+route+" HTTP/1.0\r\nHost:
"+host+"\r\n\r\n");
            // System.out.println("GET "+route+"HTTP");
```