7/31/2023

Python Based HTTP Proxy Server

Kondreddy Kiran

1. Overview of HTTP Proxy Server

1.1 What is Server?

- A server is a computer program or device that provides functionality for other programs or devices, called "clients".This architecture is called the client–server model. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients or performing computations for a client. A single server can serve multiple clients, and a single client can use multiple servers. A client process may run on the same device or may connect over a network to a server on a different device.

Here are some examples of servers:

- **Web servers:** These servers deliver web pages to clients.
- **Mail servers:** These servers send and receive email messages.
- **File servers:** These servers store and share files.
- **Database servers:** These servers store and manage databases.
- **Application servers:** These servers run applications that are accessed by clients.

Servers are typically high-performance computers that are designed to handle a large number of requests. They are often equipped with multiple processors, a lot of memory, and a fast network connection.

Servers are an essential part of the modern internet and computing infrastructure. They provide the foundation for many of the services that we use on a daily basis.

Here are some of the uses of servers:

- Storing and serving web pages
- Sending and receiving email
- Managing files

- Storing and managing databases
- Running applications
- Providing remote access to resources
- Hosting virtual machines
- Providing streaming media
- Providing disaster recovery

Servers can be either physical or virtual. A physical server is a dedicated computer that is used to run server software. A virtual server is a software-based emulation of a physical server. Virtual servers are often used to save space and resources.

1.2 What is Proxy Server?

- ➢ A proxy server is a server that acts as an intermediary between a client and a server. When a client requests a resource from a server, the proxy server intercepts the request and forwards it to the server. The proxy server then returns the response from the server to the client.

There are two main types of proxy servers:

- **Forward proxy:** A forward proxy is a server that sits between a client and a server. The client sends all of its requests to the proxy server, which then forwards the requests to the server.

- **Reverse proxy:** A reverse proxy is a server that sits between a server and a client. The server sends all of its requests to the proxy server, which then forwards the requests to the client.

Proxy servers are often used by businesses to improve the security and performance of their networks. They can also be used by individuals to protect their privacy and bypass geo-blocking.

Here are some examples of proxy servers:

- **Open proxies:** These are free proxy servers that are available to the public. They are often used to bypass geo-blocking or to hide the user's IP address.
- **Private proxies:** These are proxy servers that are owned by individuals or businesses. They are often used for security or performance reasons.
- **Residential proxies:** These are proxy servers that are hosted on residential IP addresses. They are often used to bypass geo-blocking or to make it appear that the user is located in a different country.

1.3 Services provides by Proxy Server

Proxy servers offer various services and functionalities, which include:

1. **Anonymity**: Proxy servers can hide the user's IP address and identity, providing a level of anonymity while accessing websites and online services. This is useful for privacy protection and bypassing geo-restrictions.
2. **Content Filtering**: Proxy servers can be configured to block access to certain websites or content categories based on predefined rules. This is often used in organizations to control and monitor employees' internet usage.
3. **Caching**: Proxy servers can cache frequently requested web pages and resources locally. When another user requests the same content, the proxy can serve it from its cache, reducing bandwidth usage and improving response times.
4. **Load Balancing**: Proxy servers can distribute incoming requests among multiple servers in a server farm. This helps balance the load and ensures efficient utilization of resources, leading to better performance and availability.
5. **Security and Firewall**: Proxy servers act as a buffer between the user and the internet, providing an additional layer of security.

They can block malicious traffic, prevent direct access to internal network resources, and protect against certain types of cyber attacks.

6. **Access Control**: Proxy servers can restrict access to specific websites or online services based on user authentication or IP address. This is common in corporate environments to enforce internet usage policies.

7. **Bypassing Restrictions**: Proxy servers can be used to bypass internet censorship and access content that might be blocked in certain regions or countries.

8. **Bandwidth Control**: In some cases, proxy servers can limit the amount of bandwidth allocated to specific users or devices, preventing excessive bandwidth consumption by certain applications or users.

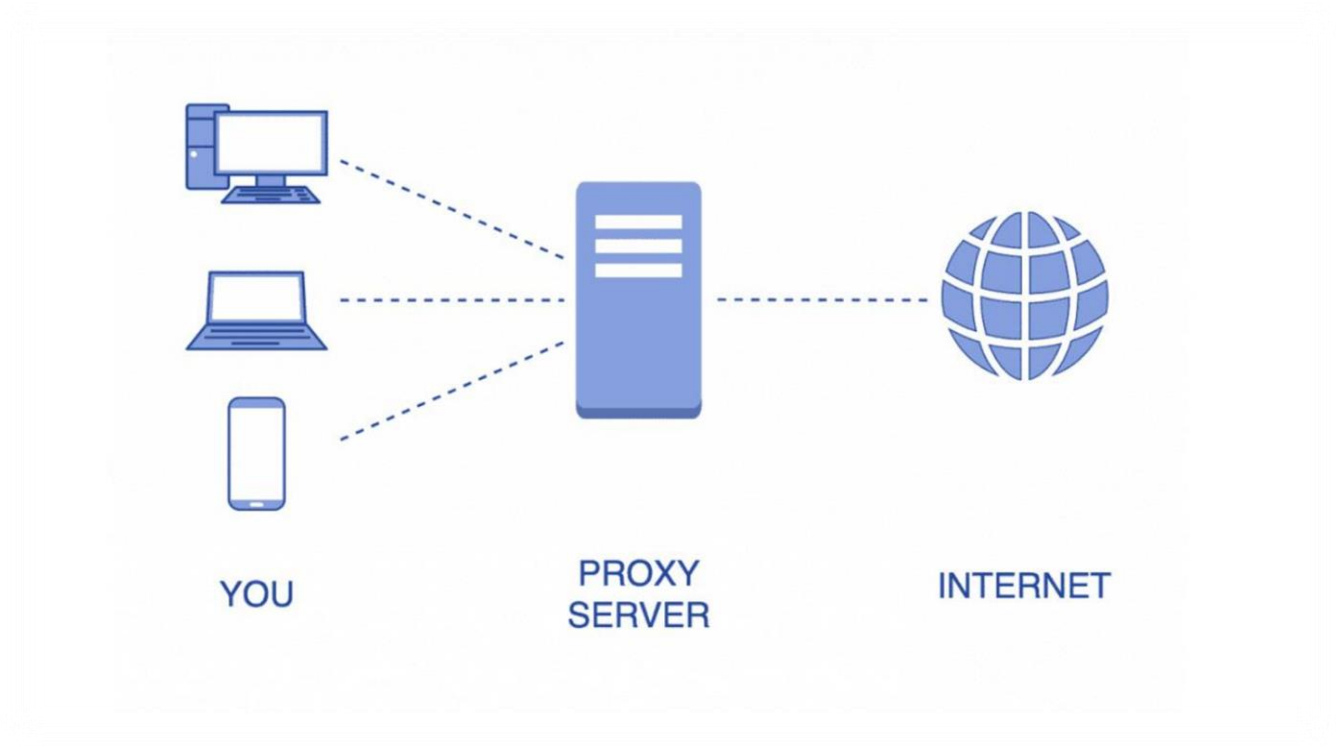
9. **SSL Decryption**: Some advanced proxy servers can decrypt and inspect SSL-encrypted traffic for security purposes, like detecting and blocking potential threats.

10. **Protocol Conversion**: Proxy servers can convert requests and responses between different protocols, allowing clients with different communication protocols to interact seamlessly**.**

1.4 Understanding working of Proxy Server

Here's how a proxy server works in more detail:

1. A client sends a request to a proxy server.
2. The proxy server checks to see if it has the requested resource cached. If it does, the proxy server returns the cached resource to the client.
3. If the proxy server does not have the requested resource cached, it forwards the request to the server.
4. The server sends the response to the proxy server.
5. The proxy server returns the response to the client.



Let's break down the working of a proxy server into several key aspects:

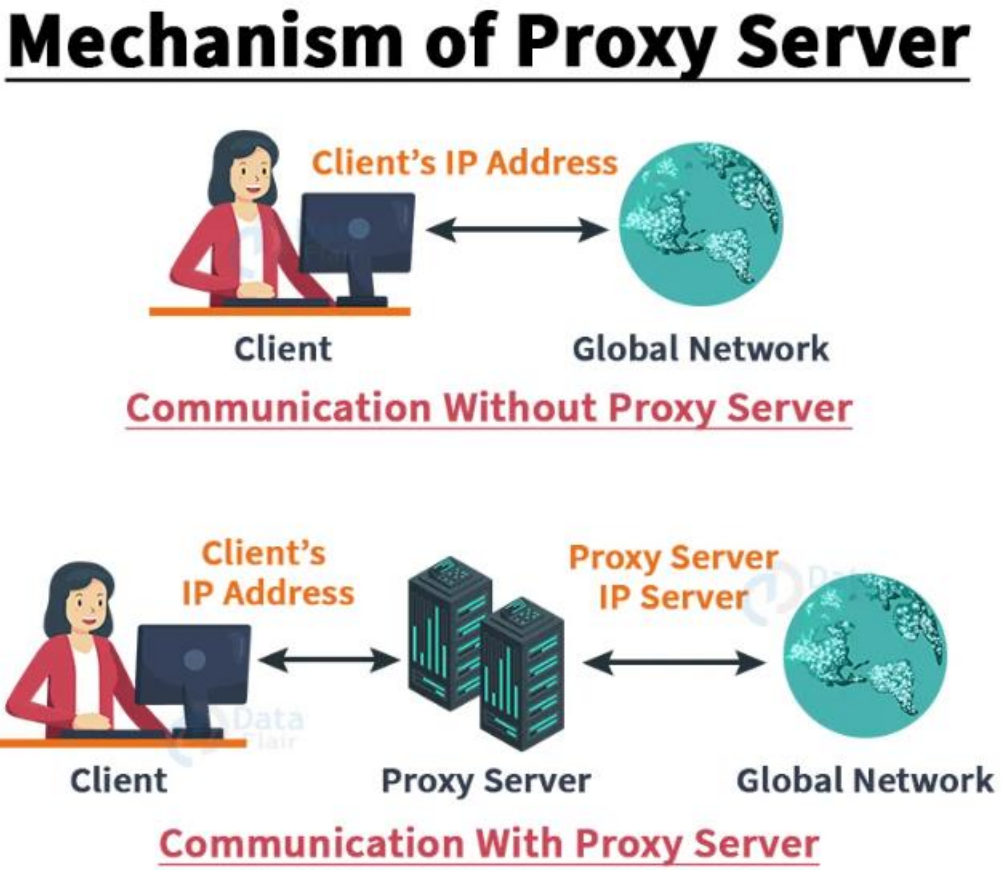
1. **Request interception**: When a client (e.g., a user's web browser) wants to access a web resource (e.g., a website), it sends a request to the proxy server instead of directly reaching out to the destination server. The client configures its network settings to use the proxy server.

2. **Forwarding requests**: Upon receiving the client's request, the proxy server forwards it to the destination server (e.g., the website server) on behalf of the client.

3. **Response interception**: The proxy server then receives the response from the destination server.

4. **Forwarding responses**: After receiving the response from the destination server, the proxy server forwards it back to the client.

1.5 Let’s understand with an example:



2. Requirements Analysis

2.1 Aim:

- To build a software based proxy server allowing the real time communication between private and public networks for HTTP traffic.
- Use network programming in python.
- When a client requests a web page or any other resources from the internet, the HTTP proxy server forwards that request to the web server on behalf of the client. Similarly, when the web server responds, the proxy server intercepts the response and sends it back to the client.

2.2 Functional Requirements

Software implementation should fulfill following requirements:

• Allow to access HTTP traffic to all clients.

• Accept incoming HTTP request.

• Accept incoming HTTP request.

• Forward valid request to intended destination website.

• Provide response back to client requesting website.

2.3 Functional Design

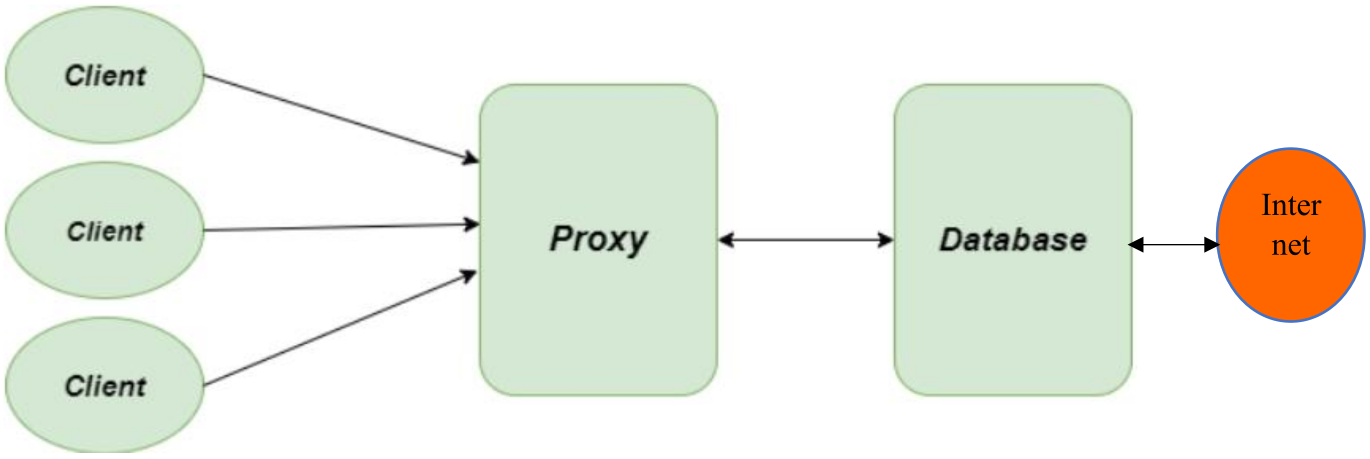
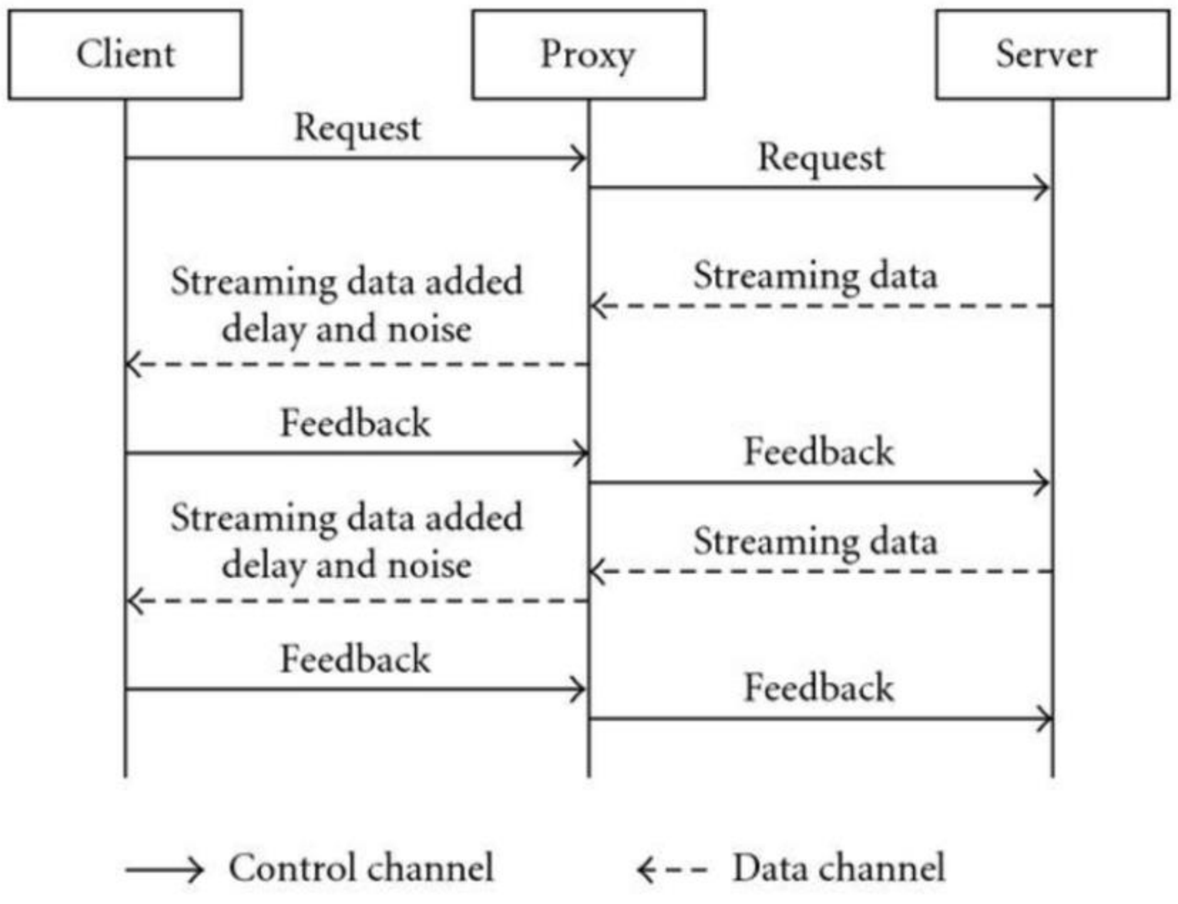


***Fig:** Functional Design*

Here is the functional design of a Python-based HTTP proxy server:

- The server will listen for incoming HTTP requests on a specified port.
- When a request is received, the server will forward the request to the original web server.
- The server will then receive the response from the web server and forward it back to the client.
- The server can optionally modify the request or response before forwarding it.
- For example, the server can add or remove headers, or it can encrypt or decrypt the data.

Here is a sequence diagram of http proxy sever



2.4 Environment Setup

Here are the steps on how to set up a Python-based HTTP proxy server:

1. Firefox Web Browser
2. Internet connection (minimum 10Mbps)
3. Additional Physical device (to use as a client)

3. Implementation

3.1Listening Client Requests

To listen to client requests in a Python-based HTTP proxy server, you can use the following steps:

- Building server side socket
    - socket(family, protocol)
- Binding the socket with address (IP and port Number)

- bind((ip,port)) ➔ on which proxy server will run

3.2 Accepting and Displaying client data

Here are the steps on how to accept and display client data in a Python-based HTTP proxy server:

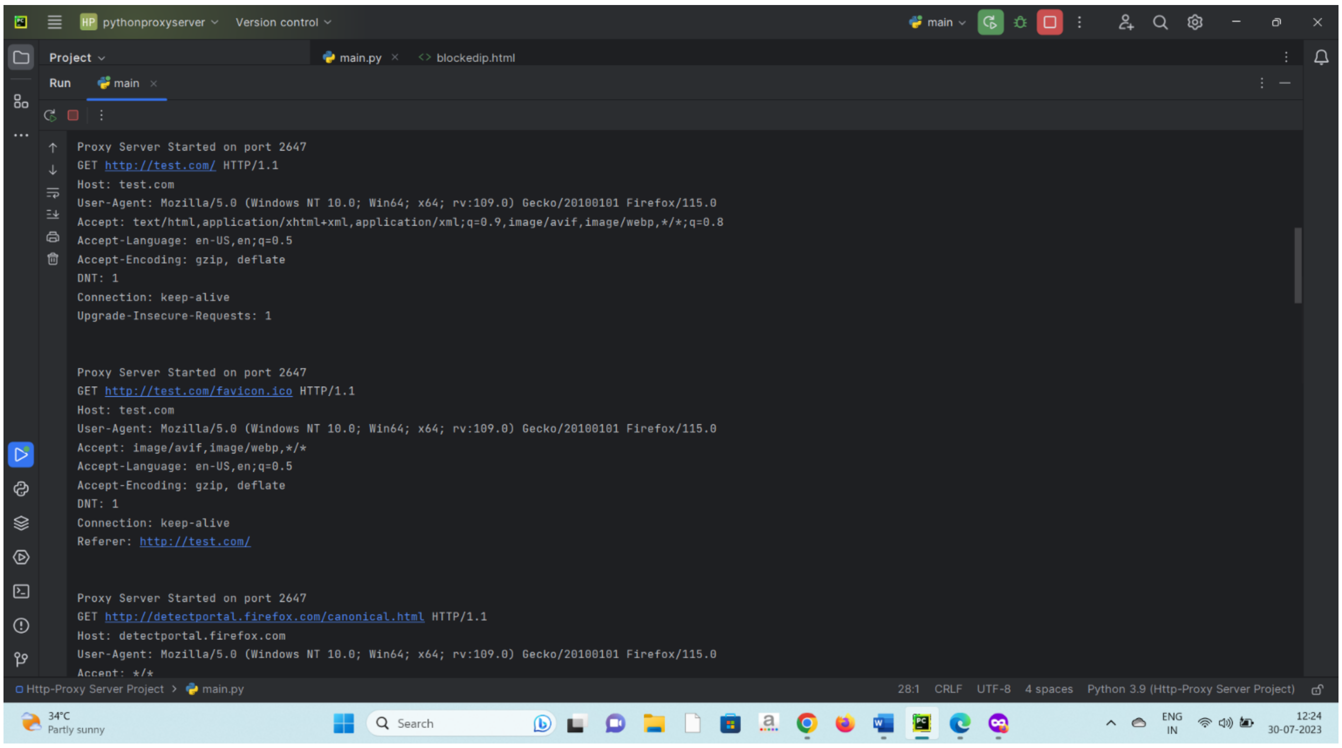
1. Import the necessary Python libraries, such as socket and re.
2. Create a socket object and bind it to a port.

   →newSocket, clientAddr = accept()

3. Listen for incoming connections from clients.
4. When a client connects, create a new service socket for providing service.
5. It returns newly created socket for writing response and clients socket address.(IP and port of Client)
6. Parse the HTTP request and extract the client's data.
7. Display the client's data.
8. Close the connection with the client.

3.3 Sending Simple Hello World to Client

HTTP request sent by a client to a server shown in below test window:

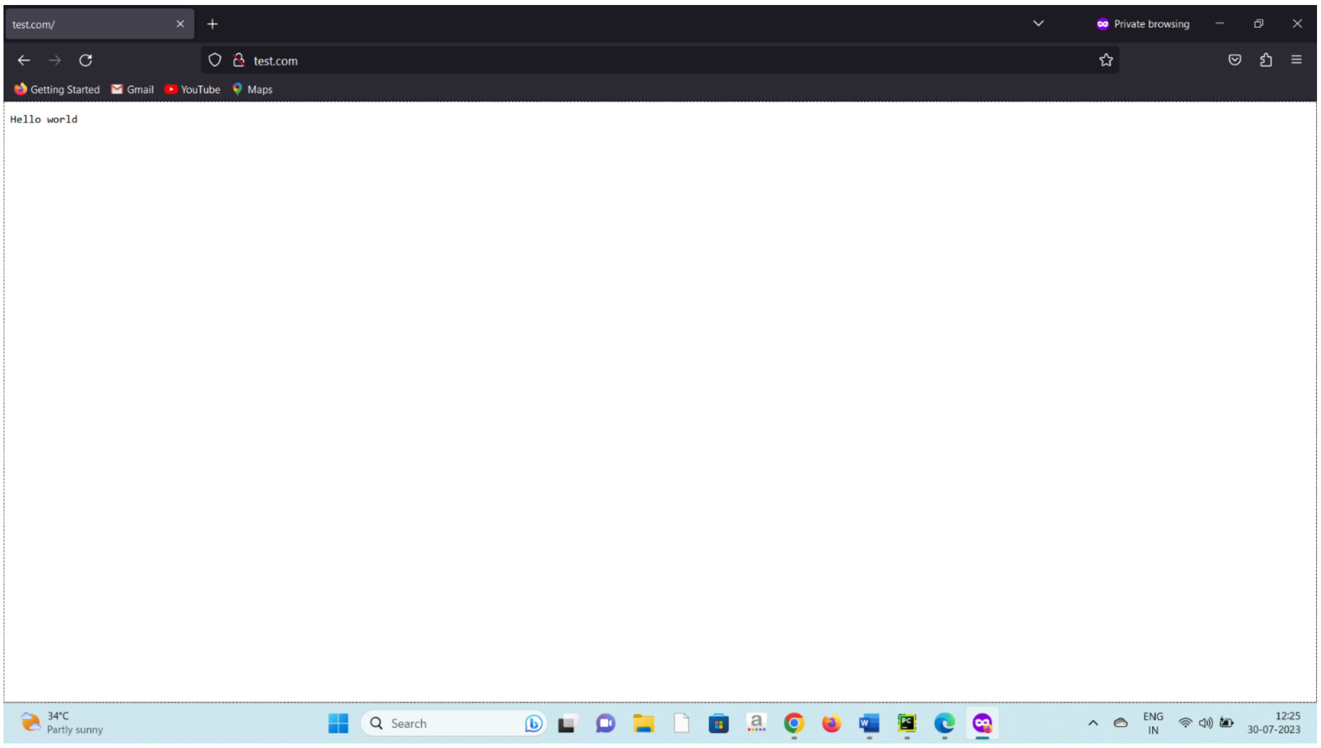


```
Proxy Server Started on port 2647
GET http://test.com/ HTTP/1.1
Host: test.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1


Proxy Server Started on port 2647
GET http://test.com/favicon.ico HTTP/1.1
Host: test.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://test.com/


Proxy Server Started on port 2647
GET http://detectportal.firefox.com/canonical.html HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
```

Http-Proxy Server Project > main.py
28:1 CRLF UTF-8 4 spaces Python 3.9 (Http-Proxy Server Project)
34°C Partly sunny
Search
ENG IN
12:24 30-07-2023

Fig: HTTP request sent by a client to a server

Output:



```
Hello world
```

3.4 Designing warning Pages

It consists of three basic types of warning pages, which includes:

• **Blocked IP**: page returned when user tries to access blocked website.

• **Blocked keyword**: page returned when user tries to search blocked keyword.

• **Blocked IP**: when user tries to access internet from blocked IP.

3.5 Getting and parsing requests

- Parsing means retrieving the URL from the request sent by the browser.
- Most common string function contains substring.

Parsed request will provide three important things:

a. Request method: GET/POST/CONNECT
b. Request URL
c. Request Host

3.6 Forwarding valid request over public network

Forwarding valid requests over a public network refers to the process of sending data packets from one network device to another across the internet or any publicly accessible network. This forwarding process allows data to reach its intended destination, enabling communication and data exchange between different devices and systems on the internet.

It includes the following things:

• Module: requests

• pip install requests

• Import request module in python

• Prepare request header

• headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:55.0) Gecko/20100101 Firefox/55.0',}

- response = req.get(url,headers=headerObj)
- content = response.content

3.7 **Source Code:**

a. main.py

```
from socket import *
from ResponseManager import *
from RequestParser import *
import requests as req


class PythonProxyServer:
    def __init__(self, port):
        self.ip = '0.0.0.0'
        self.port = port
        self.http_header = "HTTP/1.1 200 OK\n\n"
        self.socket = socket(AF_INET, SOCK_STREAM)
        self.socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
        self.socket.bind((self.ip, self.port))
        self.responseManager = ResponseManager()
        self.isValid = True

    def listen(self, backlog):
        self.socket.listen(backlog)
        print("Proxy Server Started on port " +
str(self.port))

    def accept(self):
        self.socket.accept()
        self.responseWriter, self.clientAddr =
self.socket.accept()
        self.requestStr =
self.responseWriter.recv(1024).decode()
        print(self.requestStr)
        reqParser = RequestParser(self.requestStr)
        self.Url = reqParser.getReqUrl()
        self.host = reqParser.getReqHost()
        self.isBlockedHost(self.host)
        self.isBlockedKeyword(self.Url)
        self.forwardRequest()

    def isBlockedHost(self, hostName):
        if (hostName == "yahoo.com","Movierulz.com"):
            msg = self.responseManager.getBlockedWebsitePage()
            self.isValid = False
            self.responseWriter.sendall(msg.encode("utf-8"))
            self.responseWriter.close()
    def isBlockedKeyword(self, url):
        if (str(url).__contains__("movies")):
            msg = self.responseManager.getBlockedKeywordPage()
            self.isValid = False
            self.responseWriter.sendall(msg.encode("utf-8"))
            self.responseWriter.close()

    def forwardRequest(self):
        if (self.isValid == True):
            try:
                header = {
                'User-Agent':'Mozilla/5.0 (Macintosh; Intel
Mac OS X 10.12;rv:55.0)Gecko/20100101 Firefox/55.0',}
                resp = req.get(url=self.url,headers=header)
                self.responseWriter.sendall(resp.content)
            except Exception as e:
                print(e)


if __name__ == '__main__':
    while True:
          proxy = PythonProxyServer(2647)
          proxy.listen(5)
          proxy.accept()
```

b. RequestParser.py

```
class RequestParser:
    def __init__(self,requestString):
        lines = str(requestString).splitlines();
        self.reqMethod = lines[0].split(" ")[0]
        self.reqUrl = lines[0].split(" ")[1]
        self.reqHost = lines[1].split(" ")[1]
    def getReqMethod(self):
        return self.reqMethod
    def getReqUrl(self):
        return self.reqUrl
    def getReqHost(self):
        return self.reqHost
```

c. ResponseManager.py

```
import fileinput

class ResponseManager:
    def __int__(self):
        self.filename = ""
    def getBlockedUserPage(self):
        f = open("blockedip.html","r")
        str = f.read()
        return str;
    def getBlockedKeywordPage(self):
        f = open("blockedkeyword.html","r")
        str = f.read()
        return str;
    def getBlockedWebsitePage(self):
        f = open("blockedwebsite.html","r")
        str = f.read()
        return str;
```

d. Blockedip.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Blocked Website</title>
</head>
<body>
      <center>
             <h1 style="color:red">Your ip is Blocked</h1>
     </center>
</body>
</html>
```

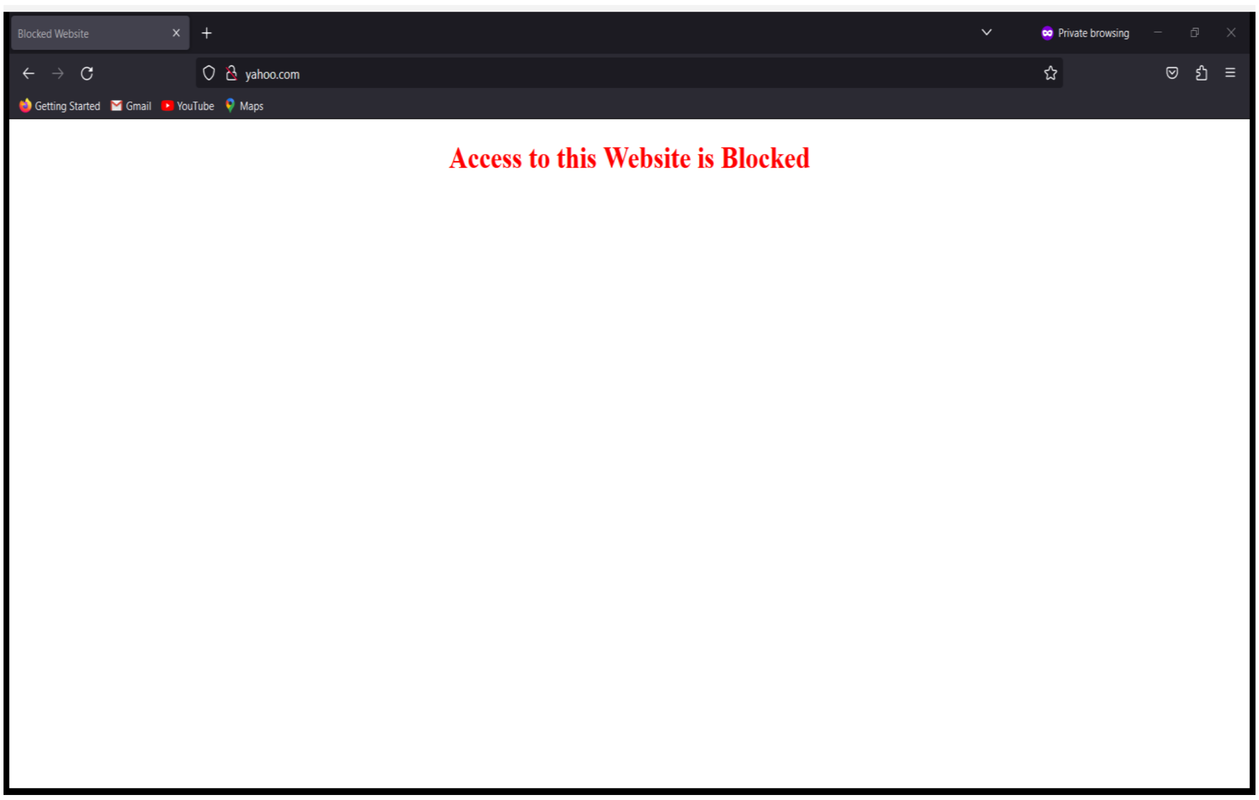
e. Blockedkeyword.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Blocked Website</title>
</head>
<body>
     <center>
             <h1 style="color:red">Keyword you have searched
for is blocked</h1>
     </center>
</body>
</html>
```

f. Blockedwebsite.html

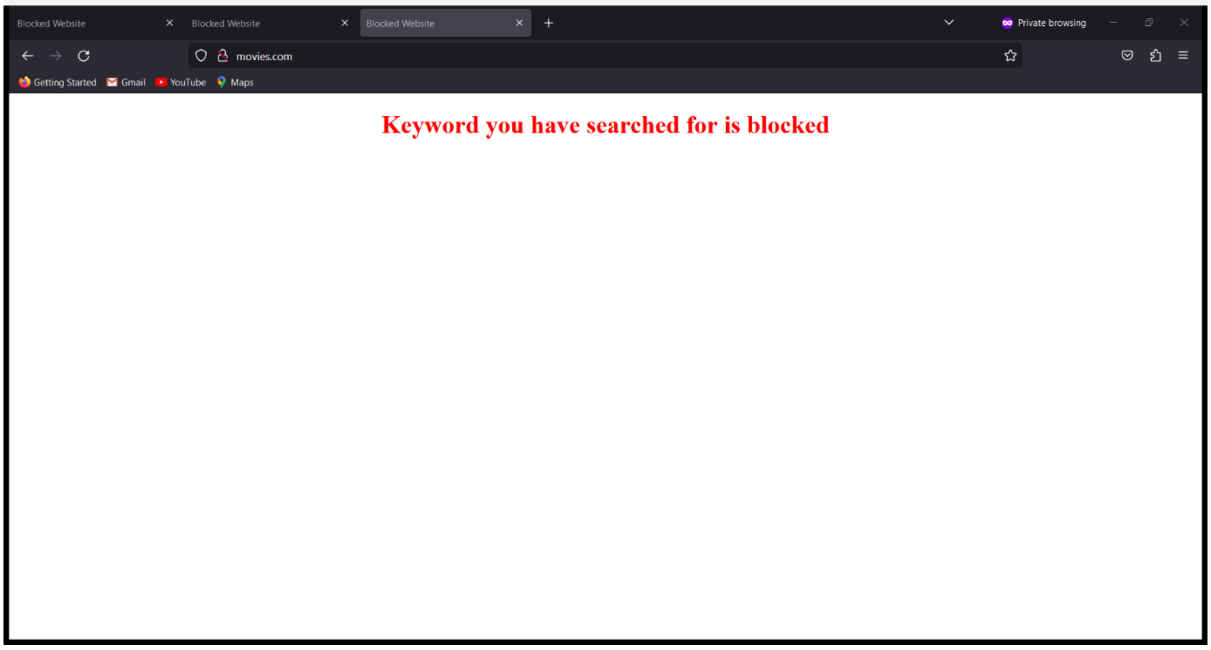
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Blocked Website</title>
</head>
<body>
     <center>
             <h1 style="color:red">Access to this Website is
Blocked</h1>
     </center>
</body>
</html>
```

Output:



Access to this Website is Blocked

Fig: Blockedwebsitepage



Keyword you have searched for is blocked

Fig: Blockedkeywordpage

4. Testing

There are four main types of testing used in Python based HTTP proxy servers:

- **Unit testing**: Unit testing is the most basic type of testing. It tests individual units of code, such as methods or functions. Unit tests are typically short and simple, and they should be easy to run and maintain.
- **Feature testing**: Feature testing tests the functionality of the proxy server as a whole. Feature tests typically involve making requests to the proxy server and verifying that the responses are correct.
- **Integration testing**: Integration testing tests how different components of the proxy server interact with each other. Integration tests typically involve making requests to the proxy server and verifying that the requests are forwarded to the correct destination.
- **Performance testing**: Performance testing tests how the proxy server performs under load. Performance tests typically involve

making a large number of requests to the proxy server and measuring the response times.

➢ By using a variety of testing methods, you can ensure that your Python based HTTP proxy server is reliable, secure, and easy to use.

Here are some additional tips for testing Python based HTTP proxy servers:

- Use a test framework to automate the testing process.
- Use a variety of test data to ensure that the proxy server is tested under different conditions.
- Test the proxy server with different clients, such as web browsers, command-line tools, and mobile apps.
- Test the proxy server with different operating systems and hardware platforms.

By following these tips, you can ensure that your Python based HTTP proxy server is thoroughly tested and ready for deployment.

5. Conclusion

The conclusion of a Python based HTTP proxy server is a summary of the key points of the documentation. It should reiterate the purpose of the proxy server, its features, and how to use it. The conclusion should also provide any final thoughts or recommendations for the reader.

➢ Here is an example of a conclusion for a Python based HTTP proxy server:

- I have described how to create and use a Python based HTTP proxy server. This proxy server can be used to access websites and web services anonymously, to bypass firewalls, and to cache web pages.

- The proxy server is easy to install and use, and it is highly customizable. It can be used with any web browser or Python program.

- We hope that this documentation has been helpful.