Project Mid-Term Report
Smart Proxy Server

Team 6:
Adit Kaushik, 21110133
Anish Karnik, 21110098
Ayush Modi, 21110039
Rutwik More, 21110133

1. Abstract

The purpose of this mid-term report is to provide an overview of, and demonstrate the progress we have made thus far in our project for the development of a smart proxy server. The report highlights an outline of the activities we have completed so far, and further lays down the roadmap for future tasks we aim to accomplish in order to develop a fully-functional and advanced smart proxy server. This report thus summarizes our team's progress in this project and provides a roadmap to the future tasks we wish to accomplish to further enhance and add more advanced functionalities to our smart proxy server.

2. Outline of Activities Done So Far

As per our proposed intended work, we firstly started off with breaking down and dividing the overall project into distinguishable smaller tasks and divided them amongst our group members wherein each of us worked on the assigned tasks.

We commenced by performing extensive background research on proxy servers and identifying the key features we wanted our proxy to have. We then set up our Python development environment and began working on the core proxy functionalities. So far we have implemented a basic HTTP/HTTPS proxy that can accept connections, parse requests, forward them to target servers, and relay responses back to clients. Additional caching and optimization features are currently being worked on.

The following is an outline of the activities we have done so far:

- Background study on the core concepts which are required for the creation of a proxy server.
- Research on the currently available proxy servers present over the internet.
- Distinguishing and deciding how we intend to make our proxy server "smart".
- Identified the areas from where we need to start and finalized the initial tasks to be accomplished.
- Implemented functionalities: Proxy Server Creation, Request Accepting and Parsing, Basic Caching
- All the tasks accomplished so far have been elaborated in detail in the 5th section of this report.

3. Background Study

As part of our initial project planning, we conducted extensive research on various aspects of proxy servers to build a strong technical foundation of proxy servers.

- Studied the history, evolution and taxonomy of proxy servers starting from early firewall proxies to forward/reverse web proxies and beyond. This provided perspective on the changing roles and applications of proxies over time.
- Researched common proxy architectures like transparent, anonymous, intercepting, etc. Along with their use cases, advantages and implementation methods. This helped us shape and make our own proxy design decisions.
- Studied research papers on innovative proxy technologies like proxy re-encryption, active caching, mobile proxies etc. to understand state-of-the-art advancements in this field.
- Investigated modern proxy protocols like WebSockets, HTTP/2, IPv6, DNS over HTTPS, and their support in different proxy implementations.

This comprehensive background study provided in-depth knowledge covering all aspects of proxies - architectures, protocols, features, optimizations and emerging technologies. It established a strong foundation to ideate, design and develop our own high-performance smart proxy server. We studied the fundamentals of proxy servers and how they operate, including forward and reverse proxies. We looked at open source proxies like Squid and industry leaders like Nginx to understand their architectures. We researched methods for optimizing proxy performance such as caching, compression, connection pooling etc.

Based on this extensive background study on proxy servers, we finalized what all features we will be implementing in our smart proxy server by selecting the ones which will be relevant with respect to the scope of this project.

4. Toolchain Setup

a. Python: We are using the Python programming language for the creation of the proxy server. The primary reason for this choice is the wide-range of libraries and frameworks that Python supports and that we can leverage the libraries with conviction for the various features we plan to add to our proxy server. Thus, we installed and configured Python on our machine.

b. Libraries Used: socket, threading, logging.
   - Socket: This library is required for the creation of network sockets for communication. Make a server socket to receive incoming connections. Create client sockets to connect to remote servers. Sockets are used to send and receive data.
   - Threading: This library is used to create and manage threads, allowing for concurrent job execution. Handle many client connections at the same time by starting a new thread for each one.
   - This library generates log messages that provide information about the program's progress. Important events should be recorded, such as when the proxy server boots up, accepts a connection, receives/sends data, or blocks incoming requests.

c. Postman: For testing and monitoring the API calls made to and from the proxy server.

5. Tasks Accomplished

Github link: https://github.com/anish-karnik/CN_Project

We have successfully implemented the following tasks till now:

1. Creation of the basic server framework that listens on a specified port.

   A server framework is a set of software components that provide the functionality and structure for building a server application. A server application is a program that runs on a server machine and responds to requests from clients over a network. A port is a number that identifies a specific communication channel on a server machine. A server can listen on one or more ports and accept connections from clients that use the same port.
   For our implementation, we are using port number as 8000 and ip address as 127.0.0.1.

2. Accepting and parsing incoming client requests:

   a. Accepting Incoming requests.
      We create a socket and bind it to a port number on our machine. We then listen for incoming connections from clients who want to use your proxy server.

   b. Reading Data from the Client socket.
      We read the data sent by the client through the socket connection.

   c. Parsing the HTTP request.
      We parse the request packet and get data such as method, path and protocol from it. We also check whether the path is of http or https and accordingly get the port number — 80 for http and 443 for https.

   d. Handling the HTTP request.
      We decide what to do with the request based on its content. We do not allow access to predefined blocked websites.

   e. Forwarding the request to the target server.
      Once the website that is requested is recognised to not be in the list of blocked websites, we check if the website data is already present in the cache on the server. If it is not, we create a new http request and forward the request to the target server.

   f. Receiving and subsequently forwarding the Responses.
      Once we get the response from the target server, we first cache it and then forward it to the client.

3. Adding the caching mechanism.
   For the caching mechanism, we use a python dictionary, with keys as the path of the requested websites and corresponding values as the data fetched from the server. We pre-defined the maximum number of urls that we can store in the cache.

We have used a simple cache replacement algorithm. If the cache overflows, we delete the least recently requested url.

The implementation details of the above steps are explained as follows.

- ➔ Basic proxy framework - Created a socketserver based framework that can accept TCP connections on a specified port and handle the socket I/O.
- ➔ HTTP proxy - Parses HTTP requests, extracts host and path, forwards request to target server, relays response back to client.
- ➔ HTTPS proxy - Allows CONNECT method to tunnel SSL traffic. Uses Python's SSL module to wrap sockets for encrypted transmission.
- ➔ Concurrent connections - Added multithreading to handle multiple clients simultaneously.
- ➔ Access logging - Logs all incoming requests with details like client IP, requested URL etc.

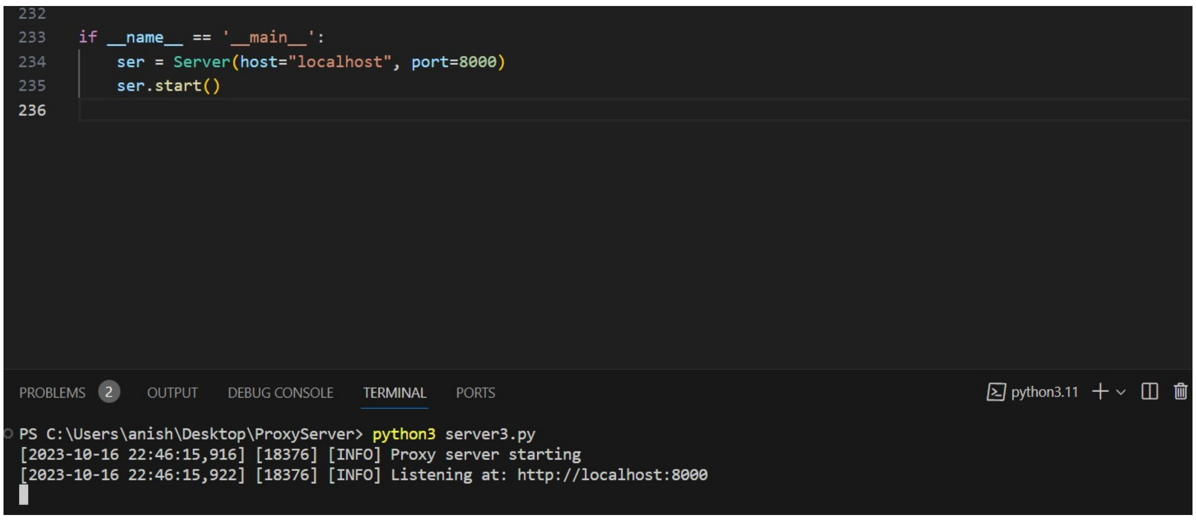
We have shown the outcomes of all the accomplished tasks in the figures and diagrams section below in this report. It highlights all the features that we have implemented till now along with attached screenshots of Postman and terminal outputs.

Finally, an overview of the entire process of the work of the proxy server has been depicted with the help of a flowchart in a step-by-step manner.

Kindly refer to the figures and diagrams section below for reference.

6. Figures and Diagrams:

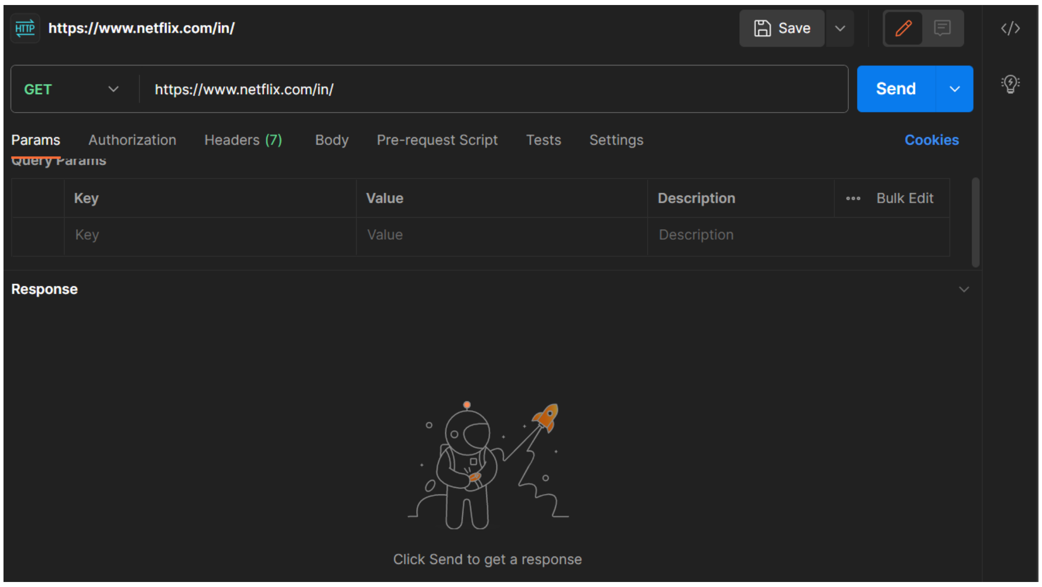
Here we would be displaying the output of the work achieved till now. This includes accepting incoming requests, giving responses, checking if the url is present in the cache and returning Cache hit or miss messages along with sending the data.



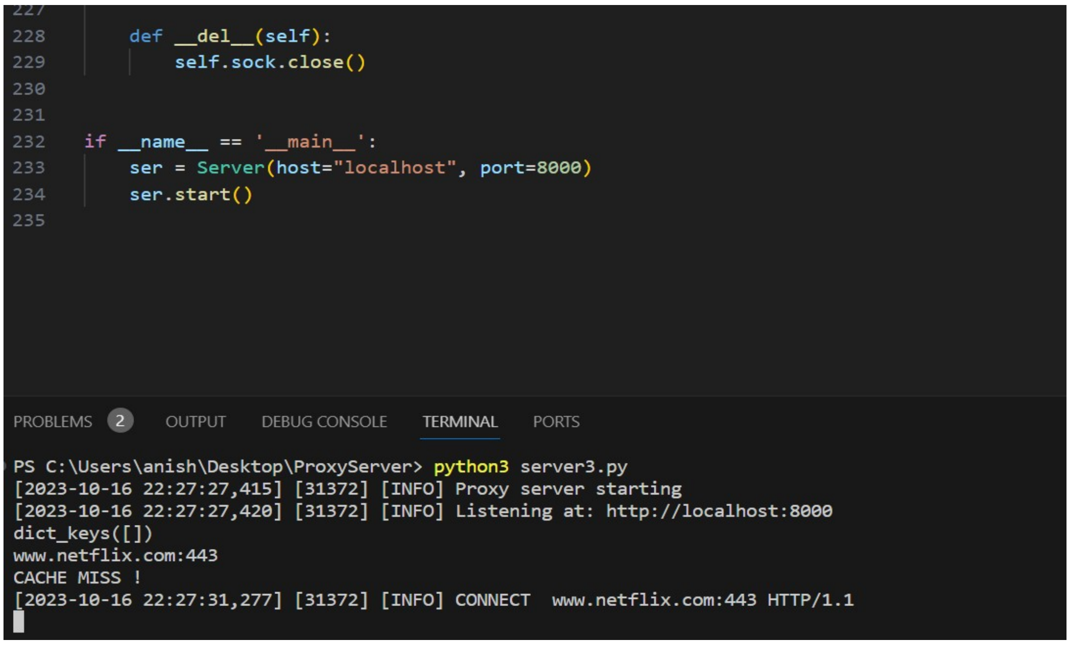
```python
if __name__ == '__main__':
    ser = Server(host="localhost", port=8000)
    ser.start()
```

```
PS C:\Users\anish\Desktop\ProxyServer> python3 server3.py
[2023-10-16 22:46:15,916] [18376] [INFO] Proxy server starting
[2023-10-16 22:46:15,922] [18376] [INFO] Listening at: http://localhost:8000
```

1. Listening Requests and Starting the Proxy Server



2. Sending GET Request for Netflix Website Using Postman

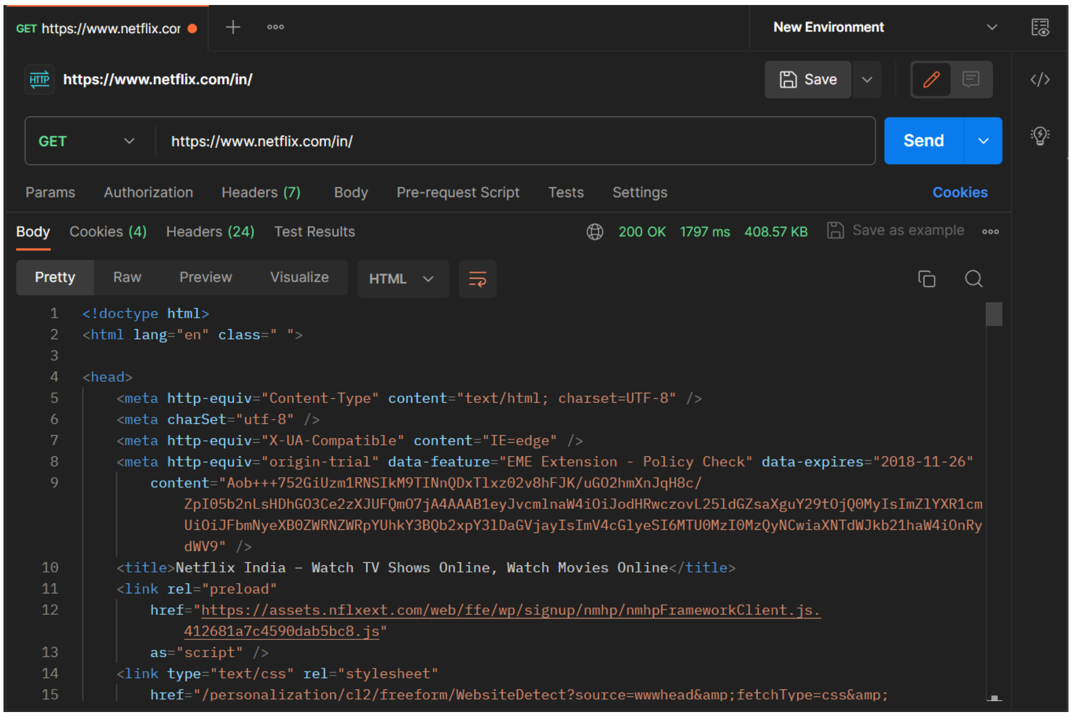


```
    def __del__(self):
        self.sock.close()


if __name__ == '__main__':
    ser = Server(host="localhost", port=8000)
    ser.start()
```

```
PS C:\Users\anish\Desktop\ProxyServer> python3 server3.py
[2023-10-16 22:27:27,415] [31372] [INFO] Proxy server starting
[2023-10-16 22:27:27,420] [31372] [INFO] Listening at: http://localhost:8000
dict_keys([])
www.netflix.com:443
CACHE MISS !
[2023-10-16 22:27:31,277] [31372] [INFO] CONNECT  www.netflix.com:443 HTTP/1.1
```

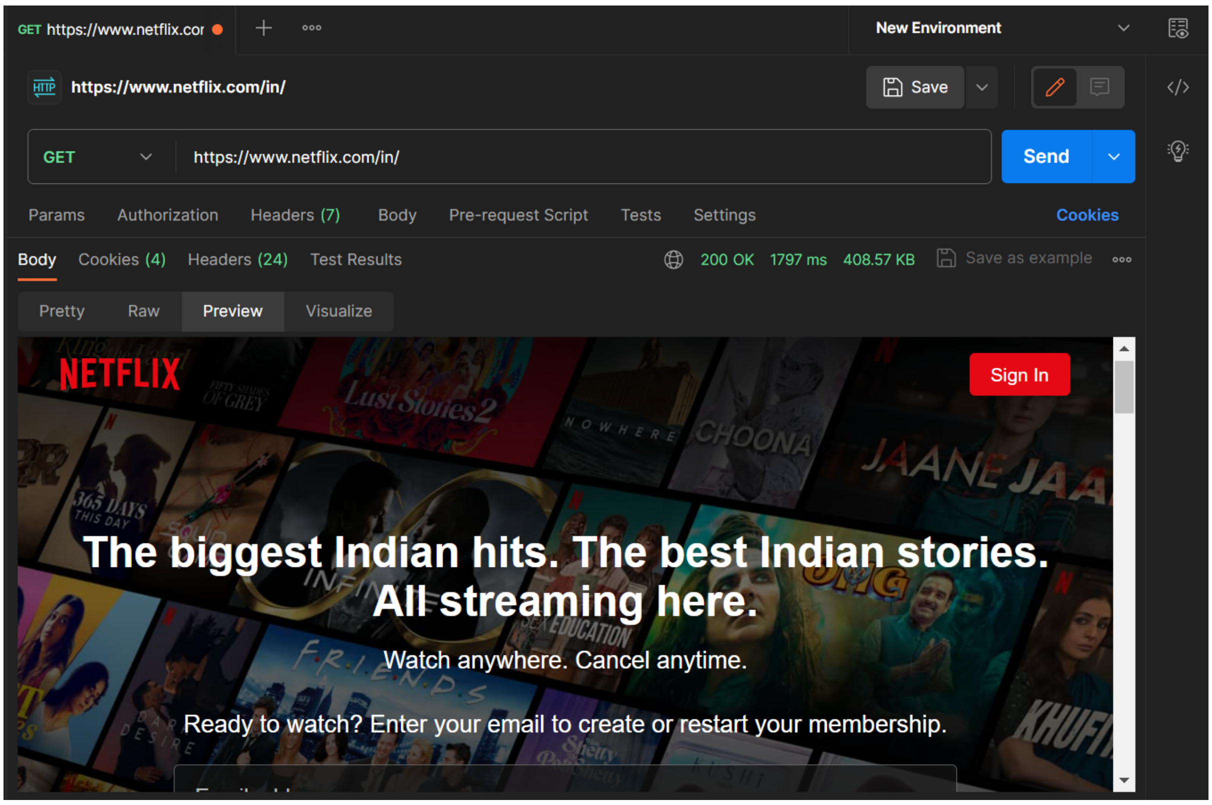
3. Receiving the request, Sending the response and Displaying Cache Miss



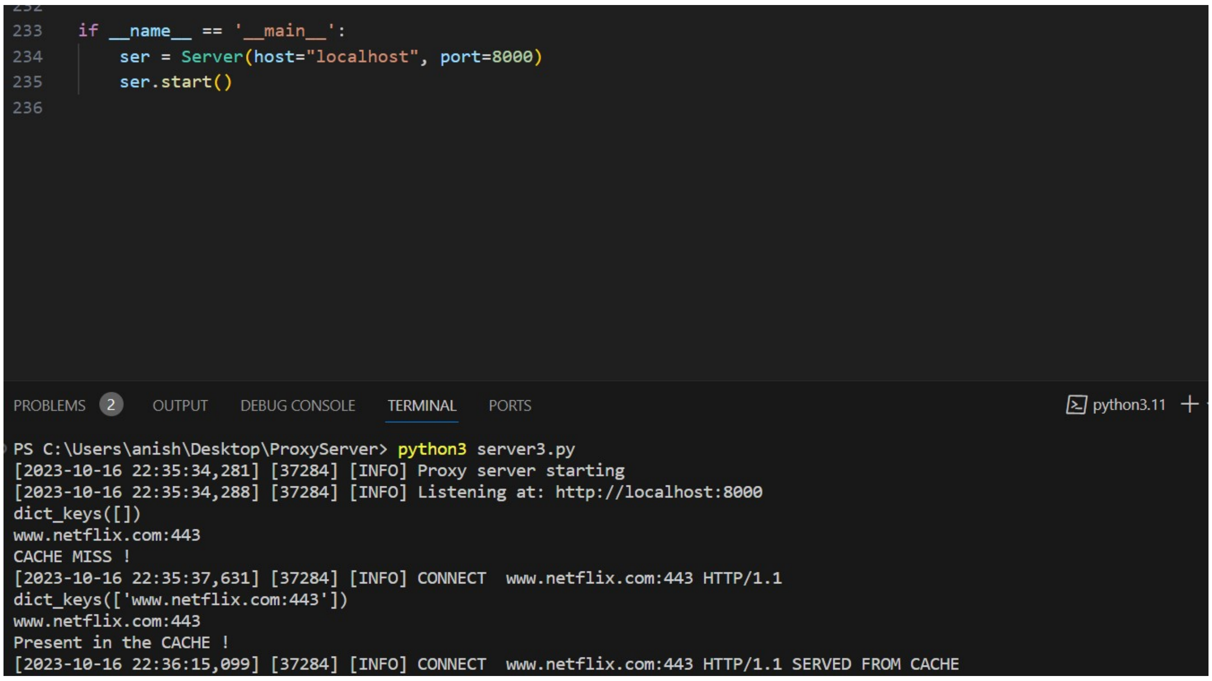
```
<!doctype html>
<html lang="en" class=" ">

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta charSet="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta http-equiv="origin-trial" data-feature="EME Extension - Policy Check" data-expires="2018-11-26"
        content="Aob+++752GiUzm1RNSIkM9TINnQDxTlxz02v8hFJK/uGO2hmXnJqH8c/
            ZpI05b2nLsHDhGO3Ce2zXJUFQmO7jA4AAAB1eyJvcmlnaW4iOiJodHRwczovL251dGZsaXguY29tOjQ0MyIsImZlYXR1cm
            UiOiJFbmNyeXB0ZWRNZWRpYUhkY3BQb2xpY3lDaGVjayIsImV4cGlyeSI6MTU0MzI0MzQyNCwiaXNTdWJkb21haW4iOnRy
            dWV9" />
    <title>Netflix India - Watch TV Shows Online, Watch Movies Online</title>
    <link rel="preload"
        href="https://assets.nflxext.com/web/ffe/wp/signup/nmhp/nmhpFrameworkClient.js.
            412681a7c4590dab5bc8.js"
        as="script" />
    <link type="text/css" rel="stylesheet"
        href="/personalization/cl2/freeform/WebsiteDetect?source=wwwhead&amp;fetchType=css&amp;
```

4. HTML response received for the request sent


GET https://www.netflix.cor
New Environment
https://www.netflix.com/in/
Save
GET
https://www.netflix.com/in/
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies (4)
Headers (24)
Test Results
200 OK
1797 ms
408.57 KB
Save as example
Pretty
Raw
Preview
Visualize
NETFLIX
Sign In
The biggest Indian hits. The best Indian stories. All streaming here.
Watch anywhere. Cancel anytime.
Ready to watch? Enter your email to create or restart your membership.

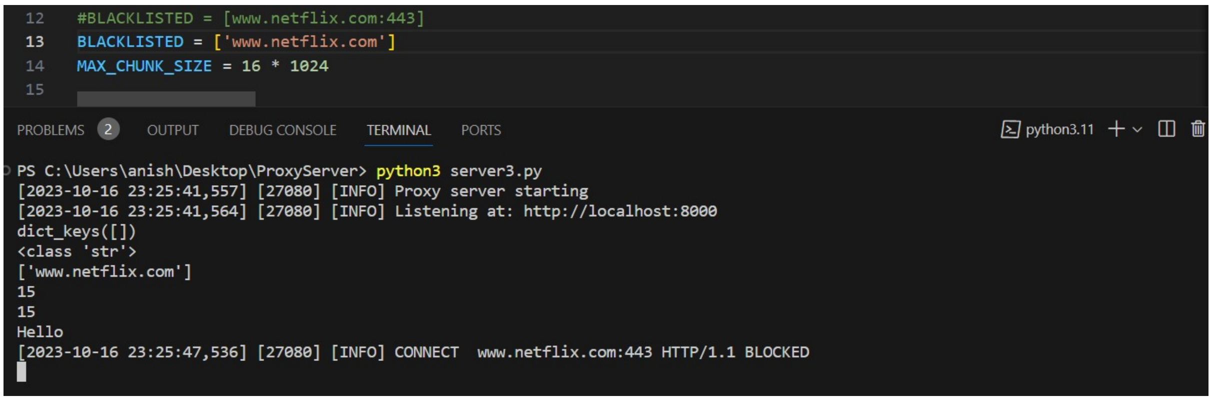
5. Preview of the website and Again sending the server a request



```python
if __name__ == '__main__':
    ser = Server(host="localhost", port=8000)
    ser.start()
```

```
PS C:\Users\anish\Desktop\ProxyServer> python3 server3.py
[2023-10-16 22:35:34,281] [37284] [INFO] Proxy server starting
[2023-10-16 22:35:34,288] [37284] [INFO] Listening at: http://localhost:8000
dict_keys([])
www.netflix.com:443
CACHE MISS !
[2023-10-16 22:35:37,631] [37284] [INFO] CONNECT  www.netflix.com:443 HTTP/1.1
dict_keys(['www.netflix.com:443'])
www.netflix.com:443
Present in the CACHE !
[2023-10-16 22:36:15,099] [37284] [INFO] CONNECT  www.netflix.com:443 HTTP/1.1 SERVED FROM CACHE
```

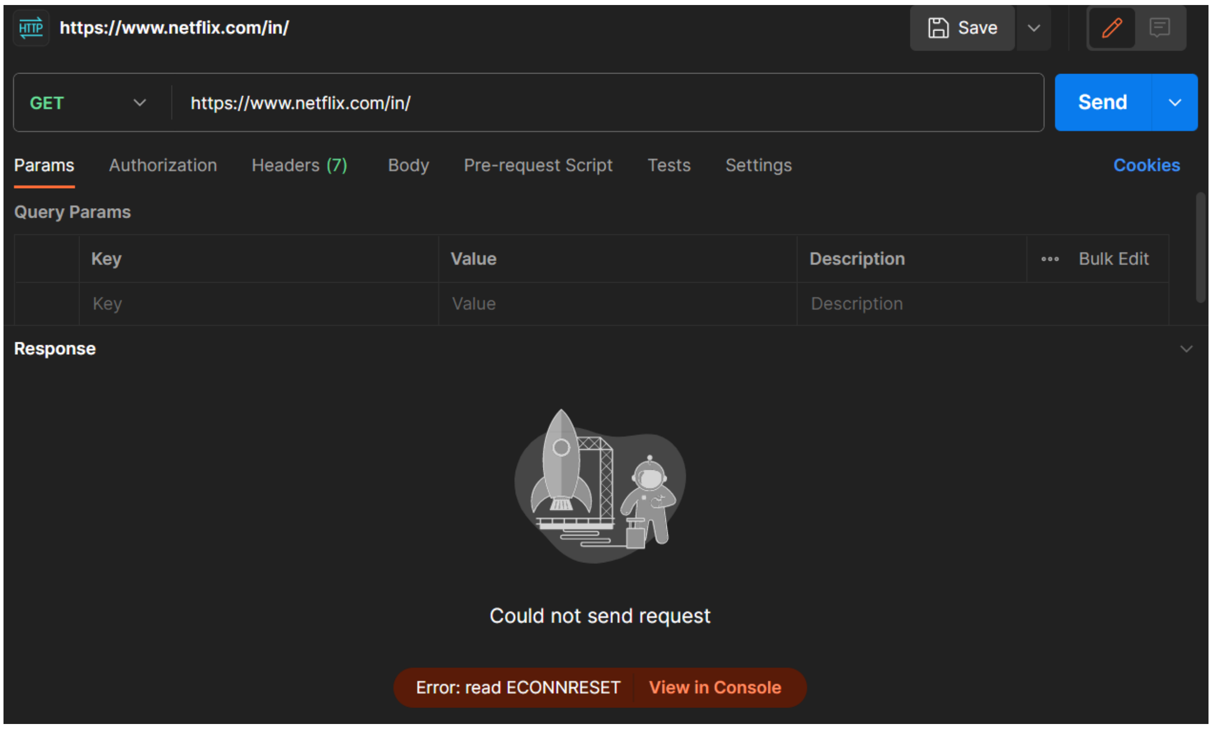
6. If URL present in the Cache then display Cache hit and and send the response



```python
#BLACKLISTED = [www.netflix.com:443]
BLACKLISTED = ['www.netflix.com']
MAX_CHUNK_SIZE = 16 * 1024
```

```
PS C:\Users\anish\Desktop\ProxyServer> python3 server3.py
[2023-10-16 23:25:41,557] [27080] [INFO] Proxy server starting
[2023-10-16 23:25:41,564] [27080] [INFO] Listening at: http://localhost:8000
dict_keys([])
<class 'str'>
['www.netflix.com']
15
15
Hello
[2023-10-16 23:25:47,536] [27080] [INFO] CONNECT  www.netflix.com:443 HTTP/1.1 BLOCKED
```

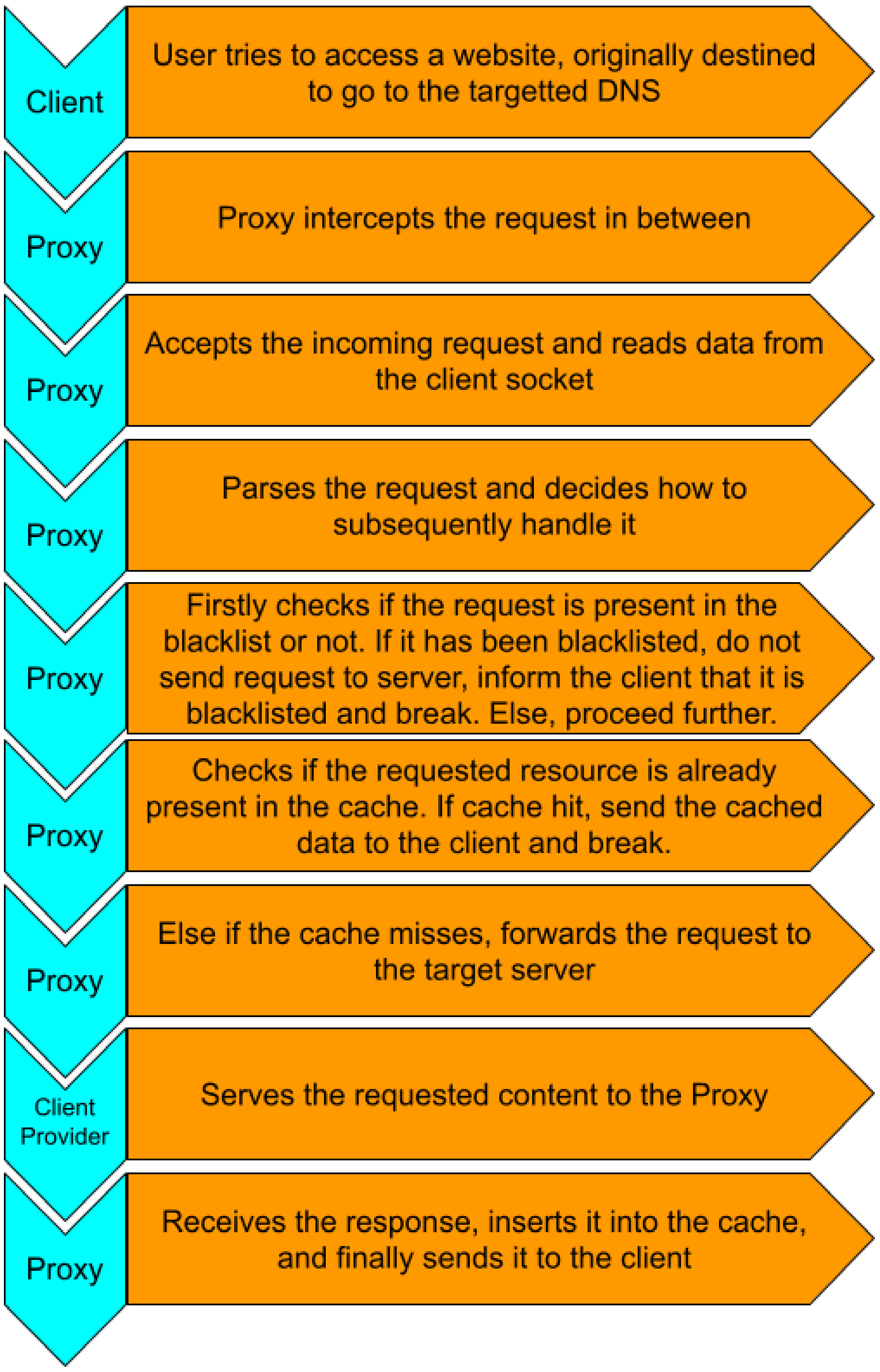
7. If the Blacklisted website is requested we decline the request displaying Blocked



Could not send request

Error: read ECONNRESET View in Console

8. Not Received the Response on Postman


Client
User tries to access a website, originally destined to go to the targetted DNS
Proxy
Proxy intercepts the request in between
Proxy
Accepts the incoming request and reads data from the client socket
Proxy
Parses the request and decides how to subsequently handle it
Proxy
Firstly checks if the request is present in the blacklist or not. If it has been blacklisted, do not send request to server, inform the client that it is blacklisted and break. Else, proceed further.
Proxy
Checks if the requested resource is already present in the cache. If cache hit, send the cached data to the client and break.
Proxy
Else if the cache misses, forwards the request to the target server
Client Provider
Serves the requested content to the Proxy
Proxy
Receives the response, inserts it into the cache, and finally sends it to the client

A flowchart representation of the entire process of sending a request from the client to getting a response along with the proxy server's interception in between.

7. Future Roadmap

As of now, we have successfully implemented a basic proxy server which accepts and parses incoming requests from the clients and responds to requests accordingly whether the requested resource is present in cache or not.

Proceeding our work done till now, we plan to accomplish the following tasks next:

a. Caching Improvements - Fine-tune cache expiration, add conditional request support, implement cache digests for improved cache utilization.
b. Geo-targeting: Smart proxy servers can be used to target specific geographic locations, which can be useful for tasks such as web scraping and market research.
c. Advanced filtering: Smart proxy servers can be configured to filter traffic based on various criteria, such as IP address, domain name, and port number. This can be useful for blocking malicious traffic and protecting your privacy.
d. The proxy could be enhanced to do lookahead, retrieving all documents that are likely to be accessed soon. For example, all the documents referenced by a document that was requested recently, including all the inlined images.
e. The HTTP protocol can be further enhanced to allow multipart requests and responses; this will allow both caching and mirroring software to refresh large amounts of files in a single connection, rather than re-connecting to the remote server once for each file. Multipart messages are also needed by Web clients for retrieving all the inlined images with one connection.
f. Basic Authentication - Add username/password based access control for private proxy use.
g. Logging & Monitoring - Add ability to log metrics like latency, hits, traffic.
h. Security Review - Implement validations and rate limiting.
i. Load Balancing - Add load balancing capabilities to spread incoming requests across multiple backend origin servers based on load. Implement health checks and failover mechanisms.

We intend to work through this roadmap to transform the basic proxy into an enterprise-ready solution with rich functionality, security, scalability and optimized performance.

8. References:

1. Luotonen, A. (1994). World Wide Web Proxies. W3C. https://www.w3.org/History/1994/WWW/Proxies/

2. Wessels, D. (2004). Proxies. W3C. https://www.w3.org/Daemon/User/Proxies/Proxies.html

3. W3C. (2015). Using Proxy Servers. W3C. https://www.w3.org/Library/User/Using/Proxy.html

4. AVG. (2022). Smart DNS proxy server vs VPN. AVG Signal. https://www.avg.com/en/signal/smart-dns-proxy-server-vs-vpn

5. Nginx Inc. (2022). Introduction to reverse proxying. Nginx. https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/#introduction

6. Nginx Inc. (2022). Reverse proxy server. Nginx Resources. https://www.nginx.com/resources/glossary/reverse-proxy-server/

7. Nginx Inc. (2022). Basic HTTP features. Nginx. https://nginx.org/en/#basic_http_features