



**INDIAN INSTITUTE OF TECHNOLOGY
GANDHINAGAR**

CS 433 : Project Final Report

Group ID : 1

Project Title : Smart Proxy Server

Group Members :

Aslaliya Juhil : 20110030

Meet Vankar : 20110112

Tejas Parmar : 20110125

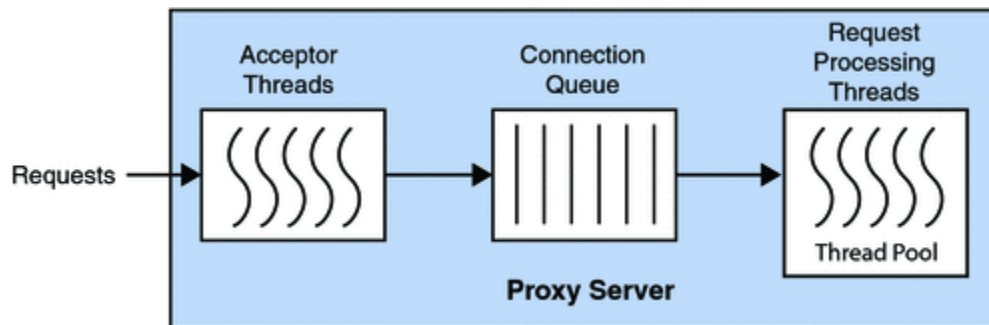
Patel Prey : 20110132

Submitted to:

Prof. Sameer G Kulkarni

Introduction

A proxy server is a server that stores the site-response-data of websites. Generally, proxy servers are used to reduce latency (page loading time) of websites for the clients. It does so by storing the response of visited websites on its local memory. The client's request first comes to the proxy server. The proxy searches its memory to find the site-response. If found, the proxy server itself sends the response back to the client. Thus, it greatly reduces the time duration that would have been required if the client were to actually locate the original web-server that hosts the site. If the site is not found in its memory, it makes a request to the original server, which takes time a bit longer than what it would have taken if there were no proxy server. However, the average (overall) response time is reduced, because the cached response can be sent to multiple clients who request the same.



The most simple, or 'dumb' proxy server, serves as a cache for response as mentioned above, as well as fetch the data from server if not in the cache. However, a 'smart' proxy server can do many other tasks, like firewall filtering, URL blocking, Ad blocking, etc., in addition to caching.

We, in this project, are trying to implement a 'smart' proxy server with some or all of the features mentioned above.

Tasks Accomplished

Configuration

- Systems : Personal Laptops
- Programming Language : Python
- Data Structures : Dictionaries (Maps) for cache, OOPs for Server object
- Network Management : Addresses, Ports, Sockets
- Logger : Text file (.txt)

Features

- Caching Mechanism
- Uncaching Mechanism
- Dynamic URL handling
- Threaded Connection handling
- Logger file for debugging and keeping records of events
- Functionality to manually stop the server without terminating the runtime
- URL Blacklisting
- Advertisement URL Blocking (Ad-blocking)

Implementation Overview

- Implemented a smart proxy server.
- Our server uses sockets for interaction.
- A class, whose object acts as the proxy server itself, is created.
- It binds the socket to an incoming request specific network-IP and port.
- It also keeps a dictionary, acting as cache, for storing responses from the data-server.
- A timeout-based uncaching mechanism.
- Threaded handling of the clients' requests for more efficiency.
- Applying locks on cache to manage its usage between threads.
- A count of cache items and cache-update events are kept for debugging purposes.
- A log-file and a logger are coded for debugging and analysis purposes.

Actual Implementation of Different Features

- A continuously running function listens for new client connections and processes them in separate threads via invoking 'Request Handling' function. This allows concurrent handling of a number of clients' requests.
- Created a function called `logg` to handle logging. It, both, displays messages on the terminal screen as well as records them in the corresponding log file.

- **Request Handling function** - Manages activities of each request by any client.
 - When a client, such as a web browser, sends a request to the proxy server, this function processes the request. It extracts the URL and the method used (HTTP only our case to make it work) from the client's request.
 - **Cache** - Maintained a cache to store responses that have been previously requested for. When a client sends a request (like - fetch a url), the server first checks if its response is present in the cache. If found, the server sends the cached response to the client. Thus, it helps the client reduce the loading time.
 - If the requested web page is not found in the cache, the proxy server acts as an intermediary client. It extracts the destination server's information from the client's request and establishes a connection to the destination web server. It then sends a copy of the client's request to the web server and receives the appropriate response (response can be data or error detail). The web-server connection is closed. The response is sent to the client. Client connection is closed. Then, the response is cached in the proxy's cache for future use. Thus, closing the connection before caching helps further reduce the latency.

- **Cache Invalidation** - We have implemented a timeout-based uncaching. When we cache a site, we store the timestamp of caching along with the site data. A continuously running function in an independent thread periodically checks the cache and removes the outdated URLs and the data associated with them. The invalidation criterion is a data item being in the cache for more than a fixed time. We invalidate the item if the time elapsed since it has last been cached is greater than the fixed invalidation time.

- **Blacklisting** - With the help of Blacklisting feature, users can block particular websites they want. We implemented it by keeping track of the websites each user wants to block. We have created a dictionary ``Blacklist`` which has the user's ip-address as key and a set of blocked websites as value. We are allowing users to add or remove any website from the blacklist just by using their browser's search bar.

- For example, if a user is visiting ``iitgn.ac.in`` website, and if he wants to block it, then: Insert the keyword ``_block_`` after ``https://`` (or any other protocol) and press Enter to send that blacklisting request to the proxy. From then onwards, whenever the user will visit that website, he will get a message stating that the site is Blacklisted by him. To un-blacklist it, the user can do the same, but with the keyword ``_unblock_``. (Note: We could also have used any other keyword. However, since ``_`` (underscore) are not legal characters for the domain name of a website. Hence, we decided to use them to differentiate between users' requests to block or unblock a website than search a website.

- **Ad-Blocking** - One of the methods to implement ad-blocking is through a proxy server, which acts as an intermediary between the user's device and the internet. In this approach, the proxy server filters web content, blocking requests to known ad-serving domains before they reach the user's browser. The effectiveness of this method hinges on maintaining an extensive and updated list of ad-serving URLs, which can be sourced from publicly available ad-blocking lists like EasyList. However, this technique also poses challenges, particularly in distinguishing between legitimate web content and advertisements, as over-blocking can lead to the unintended removal of useful website features or content.

Resolved Limitations

We were able to resolve all the obstacles and limitations mentioned in the previously submitted report. details of which are mentioned below.

Proxy Server Able to Fetch Data from Server :

- During our previous (midsem) submission, we stated that we were not able to fetch a response from any website. All we were getting was an error, and our proxy server was caching the error. We analysed the situation keenly, did research on internet, and tried out a number of possible implementations that can resolve the issue. Finally, we found out that our implementation was correct, however, there was a different issue.
- Earlier, we tried only 1 system or 2 system setups. After midsem, we did thorough research and found out how to make it work. We were only changing our Windows system's proxy settings, and trying to connect to an HTTP website. However, we found that there is another setting in browser of client that we need to make, to make it properly connect and fetch data from such a proxy server. We did that, changed structure of the code a little bit, and it was finally working for HTTP websites!

Implemented Cache Invalidation : In our previous submission, we did not have implemented an uncaching mechanism. If it had been remain the same, the cache will eventually fill up and the RAM will exhaust. Hence, we implemented an cache-invalidating mechanism to resolve it. The details for this mechanism are already presented above in the report.

Current Obstacles and Limitations

Cache Invalidation - More Sophisticated Techniques : We decided not to implement more sophisticated caching methods based on site access frequency, the number of users accessing the site, or the frequency of the site updates. The reason is that, currently, only few persons connect to the proxy. We are also accessing a very small number of sites.


Making Proxy work for HTTPS protocol : Currently our proxy is working only for HTTP. We tried to make it possible for the HTTPS as well but couldn't achieve it. The reason being the SSL/TLS interception's implementation, which is a highly complex process involving decrypting and re-encrypting data packets using certificate(s). This process not only requires significant technical expertise to implement, but also raises ethical, legal, and privacy concerns.

Validating correct working of Ad-blocking feature : Due to the abstract security provided by HTTPS, which encrypts data between the client and server, the number of websites supporting HTTP only are becoming less and less. Since our proxy server is working only for HTTP, it is even rarer to find an HTTP website that features advertisements. We tried to find at least one such HTTP website, but could not find it. We also thought about making our proxy work for HTTPS, but it also has difficulties already mentioned as above.

Learnings from the Project

- How the client-server connection works
- The forms of client requests and how to extract information from it
- Wireshark to analyze and debug our implementation
- Combining python, wireshark, and VS Code debugging features
- Difficulties in implementing an HTTPS proxy
- What all aspects to keep in mind while implementing a proxy

References

1. <https://www.javatpoint.com/what-is-a-proxy-server-and-how-does-it-work>
2. <https://www.upguard.com/blog/proxy-server>
3. <https://github.com/jerrinss5/Multi-threaded-Proxy-Server/blob/master/server.py>
4. <https://www.quora.com/How-do-I-create-a-proxy-server-in-Python>
5. https://www.wireshark.org/docs/wsug_html_chunked/
6. <https://mininet.org/walkthrough/>
7. <https://easylist.to/>
8.  Introduction to Network Packet Analysis with Wireshark