Design and Implementation of an HTTP Network Proxy Server

A Project Report

Submitted in partial fulfillment of the requirements
for the course/project evaluation

Submitted by:
Abhinav Jayale
Enrl. No: 24115006
B.Tech, Electrical Engineering
Indian Institute of Technology Roorkee

Academic Year: 2025-26

Abstract

An HTTP Network Proxy Server acts as an intermediary between client applications and web servers, enabling controlled access to network resources while providing monitoring and filtering capabilities. This project focuses on the design and implementation of a multithreaded HTTP proxy server using C++. The proxy server receives HTTP requests from clients, forwards them to the destination servers, and relays the responses back to the clients.

The system incorporates domain-based blocking using a configurable file and maintains detailed logs of client requests for monitoring and debugging purposes. A thread-per-connection concurrency model is used to efficiently handle multiple client connections simultaneously. The project demonstrates practical implementation of core networking concepts such as socket programming, concurrency, and HTTP protocol handling. The developed proxy server runs locally and can be tested using standard command-line tools.

Problem Statement

With the increasing use of internet-based applications, there is a growing need for systems that can monitor, control, and manage network traffic. Direct communication between clients and servers provides limited control over access policies and logging mechanisms. This creates challenges in enforcing access restrictions, monitoring network usage, and debugging network-related issues.

The objective of this project is to design and implement an HTTP Network Proxy Server that can act as an intermediary between clients and web servers. The proxy should be capable of forwarding HTTP requests, blocking access to specified domains, handling multiple client connections concurrently, and maintaining logs of network activity. The solution should be lightweight, efficient, and executable locally without relying on external services.

System Architecture and Design

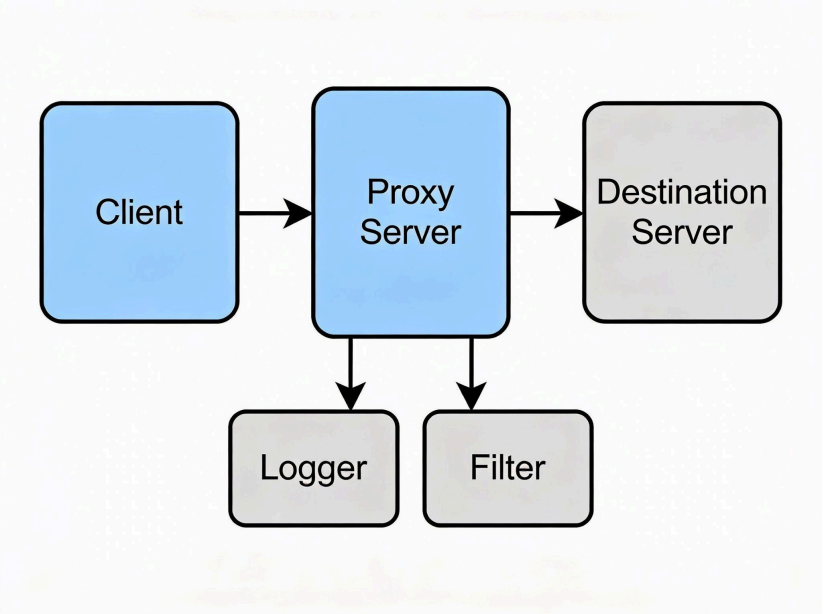

Figure 1: Architecture of the HTTP Network Proxy Server

The architecture of the HTTP Network Proxy Server follows a client–proxy–server model. Client applications send HTTP requests to the proxy server instead of directly communicating with the destination web servers. The proxy server processes each incoming request, applies filtering rules, forwards valid requests to the target servers, and relays the received responses back to the clients.

The server uses a thread-per-connection concurrency model, where each client connection is handled by a separate thread. This approach ensures isolation between client requests and allows the proxy to handle multiple clients simultaneously. A configuration file is used to store blocked domain names, enabling dynamic control over access restrictions without modifying the source code.

The proxy also incorporates a logging mechanism to record details of incoming requests and server responses. These logs assist in monitoring network activity and debugging potential issues during runtime.

Implementation Details

The HTTP Network Proxy Server is implemented in C++ using POSIX socket programming and multithreading. The project is modular in design, with each module responsible for a specific functionality. This modular approach improves code readability, maintainability, and scalability.

The server module is responsible for initializing the proxy server, creating a listening socket, and accepting incoming client connections. For every new client connection, a separate thread is spawned using the thread-per-connection model. This allows the proxy server to handle multiple client requests concurrently without blocking other connections.

The HTTP parser module processes incoming HTTP requests from clients. It extracts essential information such as the request method, host name, and target URL. This information is used to determine whether the requested domain is allowed or blocked and to correctly forward the request to the destination server.

The domain blocking functionality is implemented using a configuration file that contains a list of blocked domains. Before forwarding a request, the proxy server checks whether the requested domain exists in this list. If the domain is blocked, the request is denied and an appropriate response is returned to the client.

The logging module records details of proxy activity, including client requests and server responses. Logs are stored in a dedicated log file and are useful for monitoring network usage and debugging errors during execution. Together, these modules form a complete and efficient HTTP proxy solution.

Experimental Setup and Results

The proxy server was tested in a local Linux environment using standard command-line tools. The system was compiled using the GNU Compiler Collection (GCC) and executed on a machine running Ubuntu. Clients interacted with the proxy server using the curl command, which allowed HTTP requests to be routed through the proxy.

To verify correct functionality, multiple test cases were performed. In the first test, HTTP requests were sent to allowed domains through the proxy server. The proxy successfully forwarded the requests to the destination servers and returned the corresponding responses to the clients. These requests were also recorded correctly in the log file.

In the second test, domain blocking functionality was evaluated. Specific domain names were added to the blocked domains configuration file. When requests were made to these domains, the proxy server denied access and prevented the requests from being forwarded. This confirmed that the access control mechanism was working as intended.

Additionally, multiple client requests were issued simultaneously to evaluate the concurrency model. The thread-per-connection approach enabled the proxy server to handle concurrent requests without crashing or blocking. Overall, the experimental results demonstrate that the proxy server functions correctly and reliably under normal usage conditions.

Usage and Sample Output

The HTTP Network Proxy Server is executed locally through the command-line interface. After building the project using the provided Makefile, the proxy server is started from the terminal. Once running, the server listens on a predefined port and waits for incoming client connections.

Client applications can send HTTP requests through the proxy server using tools such as curl. By specifying the proxy address and port, requests are routed through the proxy instead of directly reaching the destination server. The proxy processes each request, applies domain filtering rules, and forwards valid requests accordingly.

Sample output can be observed in two forms. First, the terminal displays status messages indicating incoming connections and request handling. Second, detailed logs are generated in the proxy log file, which record request information and processing details. These logs serve as a record of proxy activity and demonstrate the correct functioning of request forwarding and access control mechanisms.

This command-line based interaction serves as the primary interface for using and testing the proxy server and provides clear visibility into its internal operations.

Conclusion

This project successfully demonstrates the design and implementation of an HTTP Network Proxy Server using C++. The proxy server provides controlled access to web resources by acting as an intermediary between clients and servers. Key functionalities such as request forwarding, domain-based blocking, concurrency handling, and logging were implemented and tested successfully.

The project offers practical exposure to core networking concepts including socket programming, multithreading, and HTTP protocol handling. The modular design of the system makes it extensible, allowing additional features such as authentication, caching, or HTTPS support to be incorporated in the future. Overall, the project serves as a solid foundation for understanding proxy server architectures and network traffic management.