Design and Implementation of a Custom Network Proxy Server

Shreyansh Kuntal
shreyansh_k@ph.iitr.ac.in

22324018

Contents

1 System Architecture

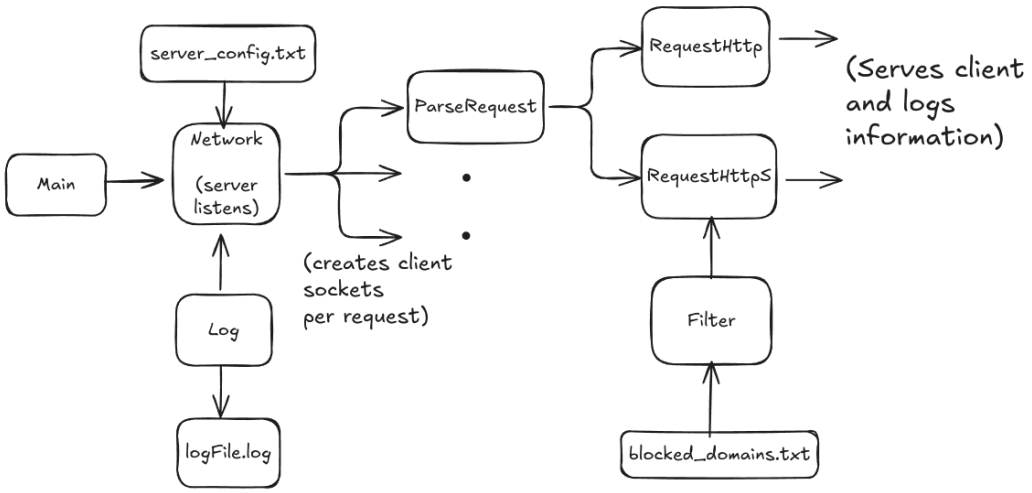

Figure 1: High-Level Architecture of the Proxy Server

Description of the various components in Figure 1 is given below:

1.1 Main

The `Main` component serves as the entry point of the application. It creates an instance of `Network` class and calls its `runProxyServer` method.

1.2 Network

The `Network` component is responsible for setting up the server socket on the port specified in `server_config.txt` file and listening for incoming client connections.
Further it accepts headers from client and uses `ParseRequest` to figure out the protocol(`HTTP` or `HTTPS`) and other stuff such as target host, port, etc. Based on the protocol it calls `RequestHttp` or `RequestHttpS` component to carry out further transfer of data.

1.3 Log

Uses the standard `java.util.logging` API to log information of every request to a text file.
By default it rotates the log files in a count of 5 and a maximum size of 1 MB.

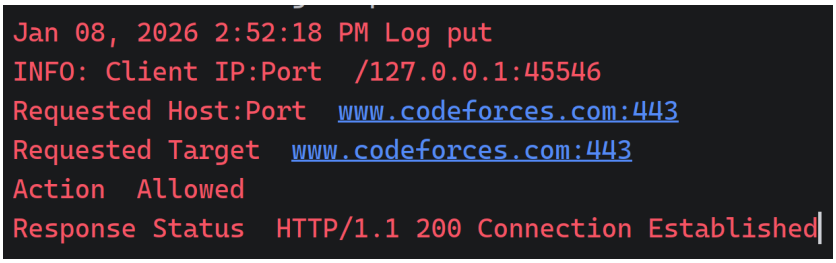

Figure 2: Example of information logged

1.4 ParseRequest

The `ParseRequest` component parses client's first request's headers and extracts information which can be accessed through its methods:

- `getHost` - such as www.codeforces.com
- `getMethod` - GET, POST, CONNECT
- `getRequestTarget` - requested webpage URL
- `getHostPort` - by default 80 for HTTP, 443 for HTTPS

1.5 RequestHttp

The `RequestHttp` component handles standard HTTP requests. It uses `Filter` component to check for blocked domains. If found blocked it responses with code 403. If valid it forwards body of the exact number of bytes specified by `Content-Length` header received from server. If `Content-Length` header is not found(in case of chunked encoding) it forwards 0 bytes(apart from headers). Finally it logs the information.

1.6 RequestHttpS

The `RequestHttpS` component handles HTTPS requests using the CONNECT method. It also uses `Filter` component. After responding with 200 OK, it acts as a tunnel between client and server and forward bytes bidirectionally using threading without interpreting them. Finally it logs the information.

1.7 Filter

The `Filter` component uses `isGood` method to determine whether a request should be blocked based on domains specified in the file `blocked_domains.txt`.

1.8 Config files

Config files should strictly follow the format specified in Figure 3 and Figure 4.

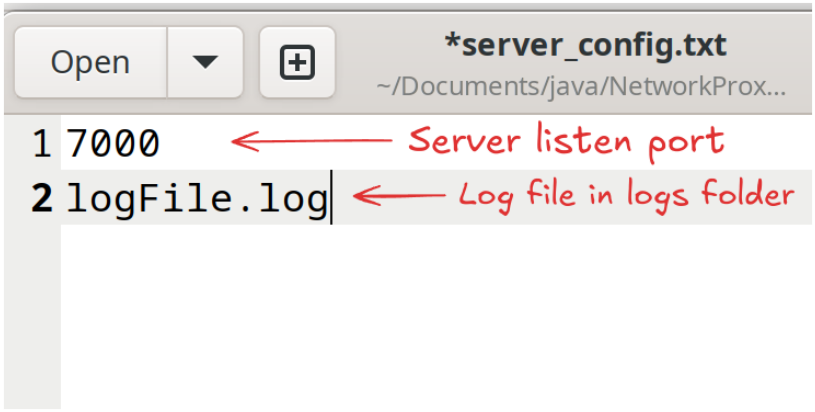

Figure 3: `server_config.txt`

2 Concurrency Model and Rationale

The proxy server is designed to handle multiple client connections concurrently using a **thread-per-connection** concurrency model. In the thread-per-connection model, the server listens for incoming client connections on a

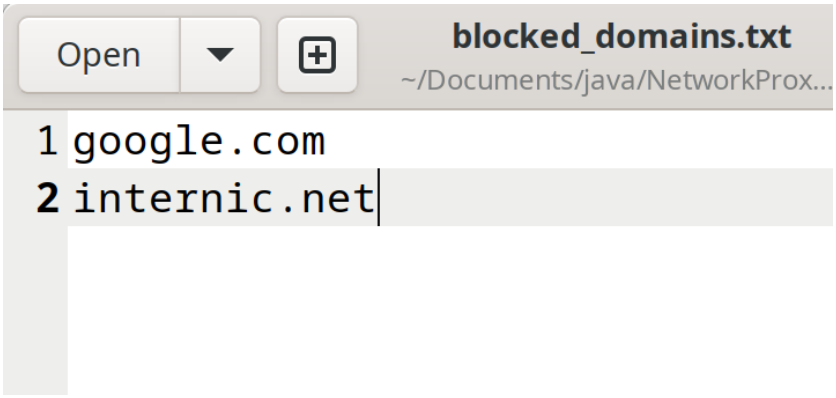

Figure 4: `blocked_domains.txt`. Each domain on new line.

server socket. For each accepted client connection, a new thread is created to handle the request independently.

2.1 Rationale for Model Choice

- **Simplicity:** The model is straightforward to implement and understand.
- **Adequate Performance for Moderate Load:** For a limited number of concurrent clients, this approach provides acceptable performance.

3 Error Handling

The proxy server handles basic exceptions to ensure stability and graceful failure during runtime such as:

- **Invalid Requests:** Requests that do not conform to the expected HTTP format or unresolved URL are rejected without forwarding. Code 403 is sent back to client.
- **I/O Exceptions:** Input and output operations are enclosed in exception handling blocks to prevent the server from crashing.
- **Graceful Termination:** Sockets and streams are closed automatically by JVM when SIGINT/SIGTERM are raised.

These mechanisms help maintain server robustness and prevent a single client error from affecting other active connections.

4 Limitations

- **Scalability:** The thread-per-connection model does not scale efficiently for a very large number of concurrent clients.
- **No Caching:** The proxy does not implement response caching, which could improve performance for repeated requests.

5 Demonstrations

Logs of all the tests are saved in logs/logFile.log.

```
shreyk@debian:~/Documents/java/NetworkProxy/tests$ curl -x localhost:7000 http://www.iitr.ac.in
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
shreyk@debian:~/Documents/java/NetworkProxy/tests$ curl -x localhost:7000 -I http://www.iitr.ac.in
HTTP/1.1 301 Moved Permanently
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 08 Jan 2026 14:02:41 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://www.iitr.ac.in/
```

Figure 5: Simple tests.

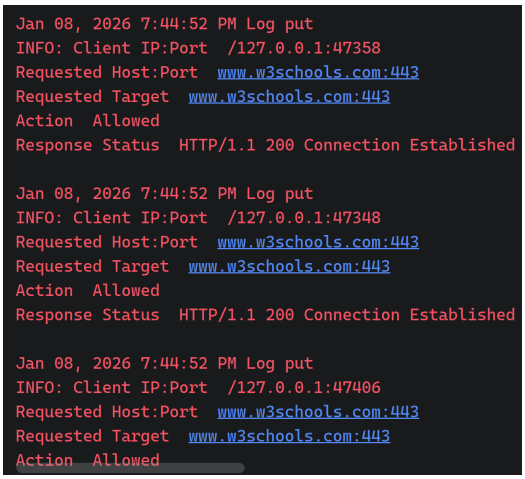

Figure 6: Concurrent tests. `script` in tests folder make 20 concurrent request. 3 of which shown in figure with same timing. Output saved in tests/out.

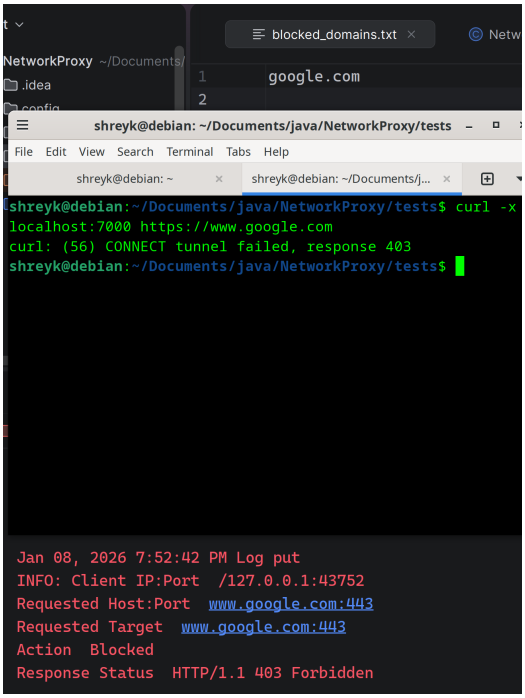

Figure 7: Blocking test. 403 response sent by proxy to client when it tries to access google.com which is in blocklist.

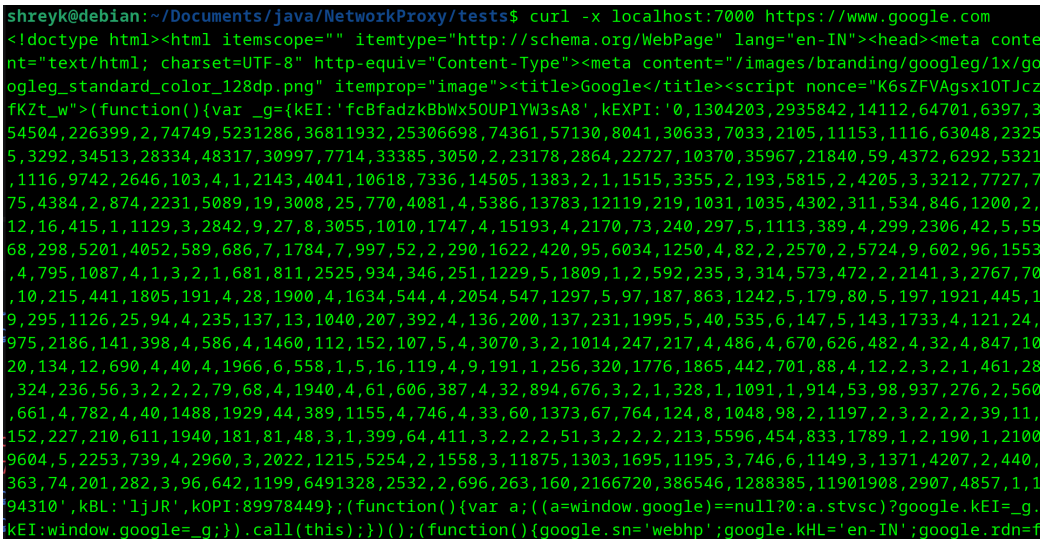

Figure 8: CONNECT test. curl request to https://www.google.com responds with webpage content indicating fine working of proxy.