Telecommunication Proxy Server

Kmla Sharma, 13319349

Introduction

This report will outline the description and implementation of a locally hosted proxy server, programmed in Java. It is a multithreaded program, employing the “man in the middle” design. Once the user’s browser has been connected to the locally hosted port number of the proxy, all HTTP requests are forwarded from the user’s browser (the client), to the java proxy. The proxy then relays this request to the server, and returns the response to the client. In order to save bandwidth and time, certain websites are cached.

Implementation & Design

1. Handling HTTP Requests:

In order to successfully transmit a HTTP request from the client to the server and return the response, each thread must follow the subsequent protocol:

1. ***Create the Server and Client socket:*** The server socket loops infinitely listening for client connections on the designated port number (“8001” in this case), and the client socket is created by accepting the server connection.

2. ***Set up the streams:*** There are four necessary BufferedStreams to be created within the proxy, to handle a) Retrieving data sent from the client (the request), b) Transmitting this data to the server, c) Retrieving data from the server (the response), d) Transmitting this data back to the client.
   Once a website has been requested by the user’s browser, this data of the request is converted into a byte array from a BufferedInputStream. It is then transmitted to the server, as a request, using a BufferedOutputStream. The response will then be read into another BufferedInputStream, which can then be easily converted to a byte array. This response is then relayed back to the user’s browser, using a BufferedOutputStream, where the corresponding webpage will be displayed. All threads must flush their input/output streams once the client has successfully received the response.

2. Dynamically Blocking Websites:

Running alongside the various threads, a user is also able to block/unblock various URLs via the management console. In order to ensure dynamic additions and removals, an array list was the data structure selected to hold all currently blocked URLs. Before a HTTP request is sent to the server, the proxy ensures that the website is not blocked, and if it is, a

simple “error” html webpage is parsed into a byte array and relayed back to the client, with the request discarded.
Note: It is extremely important to ensure that all accesses to the shared blocking array lists are synchronised, to avoid the threads overwriting/accessing incorrect data.

3. Caching Requests

In order to reduce time and bandwidth consumption, cacheable webpages are stored and can be fetched and returned back to the client without having to make contact with the server. Therefore, before a request can be transmitted, the proxy’s local cache must be checked.

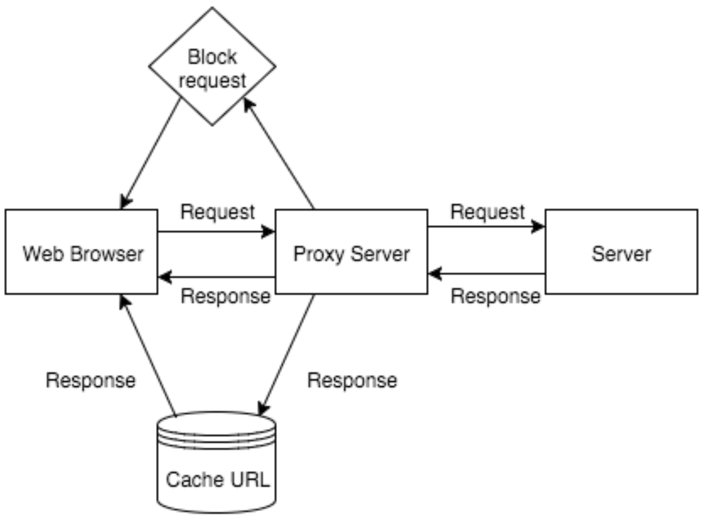
Two Hashtables were created to dynamically store the URLs. The first cache stores the URL, and it’s corresponding byte array response (that, when sent back to the client, would display the webpage). The other cache stores the URL, and two time stamps; one pertaining to the time the URL was cached at, and the second pertaining to the time the entry can stay in the cache for. A URL can only be cached once it’s response specifies the maximum amount of seconds it can be cached for, within the “max-age” section of the header.

Similar to the blocking mechanism, all accesses to both caches must be synchronised between the threads, as multiple threads will be transmitting and caching requests, therefore they must hold exclusive access to the cache when writing to it.

Checking Expiries

Before a response can be retrieved from the cache, it must be checked for expiry. Should the requested URL be expired, it must be removed from the cache and the proxy must contact the server for a response. In order to check for expiry, the expiration time stamp (time it was cached at + time it can stay in the cache for) is compared to the current time stamp. The user can also view what is currently in the cache via the Management Console.

The System:



Conclusion:

I feel that my implementation of connecting to the server and client and transmitting the HTTP responses and requests proved to be successful. Both caching and blocking mechanisms work efficiently and reduce time and bandwidth consumption, and the user can follow the requests and responses via the management console and IDE console.