Sharif University of Technology

Department of Electrical Engineering

Data Networks

Final Project

HTTP Proxy Server

Student: Ali Soltani
Student ID: 403203449
Course Instructor: Dr. Mohammad Reza Pakravan
Semester: Spring 2025

Contents

1 Introduction

A proxy server acts as an intermediary between a client device and the internet. When a client sends a request to access a resource such as a website. the request is first routed through the proxy server, which then forwards it to the target server on behalf of the client. The response from the target server is similarly relayed back through the proxy to the client. This layer of indirection serves multiple purposes, including privacy enhancement, access control, caching, and traffic monitoring.

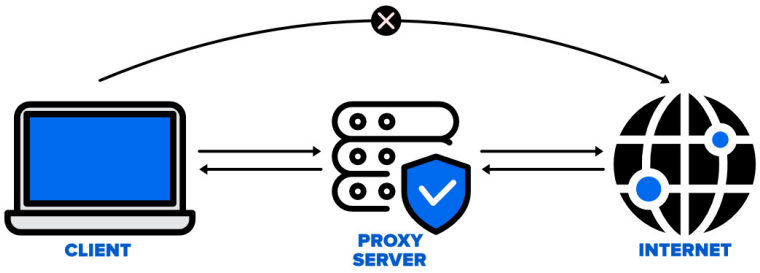

Figure 1: Proxy server.

The use of a proxy server offers several advantages:

- **Security:** Proxies help in providing a safe and secure connection to access the internet. It prevents attacks from hackers. By utilizing proxies, data breaches can be prevented and security risks can be minimized.
- **Saving bandwidth:** They help in fast access to the data that has been accessed very often by caching the content of the websites.
- **Privacy:** Proxy servers give clients the ability to browse the internet anonymously by providing alternate IP addresses.
- **Monitoring:** Proxy servers are used by organizations to log and monitor network traffic.
- **Controlling internet usage:** Proxy servers can be used to limit internet access for users. Organizations use proxy servers to limit the amount of access and usage for their employees.

2 Implemented Features

2.1 Basic Proxy Functionality

The basic functionalities of a proxy server are as follow:

- Handling client HTTP requests.
- Forwarding requests to the server.
- Relaying server responses back to the client.

Fig.2 displays how does a proxy server work.

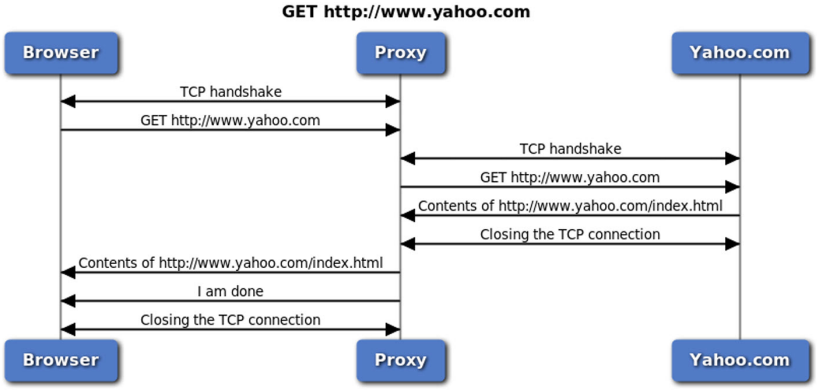

Figure 2: HTTP proxy Server functionallity.

To this end I defined the `ProxyServer` class which has the following attributes for having basic functionality:

- `local_ip`: The IP address which the client sends its requests to and receives the responses from. It is set to `127.0.0.1`.
- `listening_port`: The port which the client sends its requests to and receives the responses from. It is set to `8080`.
- `max_connections`: specifies the maximum number of queued connections that the socket can hold before it starts refusing new connection requests.
- `default_http_port`: The port that the proxy server sends requests to if the port is not specified in the request message. It is set to `80` by default.
- `packet_size`: The size of packets that the proxy receives in each connection. We assume that HTTP packet size is up to 4 KB.
- `time_out`: Setting time out for the proxy socket to ensures that a socket operation, like accepting or reading data, doesn't block indefinitely.

- `short_time_out`: Setting time out for the remote socket to ensures that a socket operation, like accepting or reading data, doesn't block indefinitely.
- `sock`: Proxy server socket which is responsible for exchanging data between client and the server.
- `is_running`: A flag to determine whether the proxy server is running or not. We start and stop the proxy by setting this flag to `True` and `False`.

I added another attributes such as `blocked_urls` and `cache_memory` which I will explain in the next subsections. Two methods, `start` and `stops`, are defined for starting and stopping the proxy server respectively.
In `start` at first a socket object is instantiateed for establishing a network communication. Then the proxy server start listenning on `(local_ip:listening_port)` and as a new request arrives it starts a thread to handle the request.
For each request a thread is responsible for handling the request. Each thread receives the request from the client, Then the URL must be parsed to determine the server location. Note that we don't need to specify the server's IP address. After sending the request to the server, proxy server receives the responses and the moment it receives the responses, it sends them to the client. To ensure the connection is not kept when the server has nothing to send, remote socket is closed after `short_time_out` seconds.

2.2 Blocking Specific URLs

Blocking URLs in proxy servers restricts access to specific websites or content to enforce security policies, comply with regulations, or manage network resources effectively.
To this end a list of blocked URLs are kept in `blocked_urls.txt`. A list of blocked URLs is kept in `blocked_urls`. `block_url` method gets an input `block` which works as follow:

- `if block == 1`: Read the `blocked_urls.txt` file and load the blocked URLs in `blocked_urls` list.
- `if block == 0`: Empty `blocked_urls` list.

For having the blocking URL feature I added a snippet as it checks the URL requested by the client and if the url is in the `blocked_urls` list, it returns a custom "Access Denied" HTML page for blocked URLs.

2.3 Web Caching

Caching involves storing copies of frequently accessed data, such as web pages or files, closer to the user to reduce retrieval time and bandwidth usage. By keeping this data, subsequent requests for the same content can be served faster without fetching it from

the original source. This improves performance, especially for repetitive access patterns, reduces server load, and lowers latency for users. It can also save costs in environments with limited or metered bandwidth.
In proxy servers, caching stores copies of web content, such as HTML pages, images, or scripts, on the proxy itself after the first request. When a client requests the same content again, the proxy serves the cached version instead of contacting the origin server, significantly reducing response time and internet bandwidth usage.

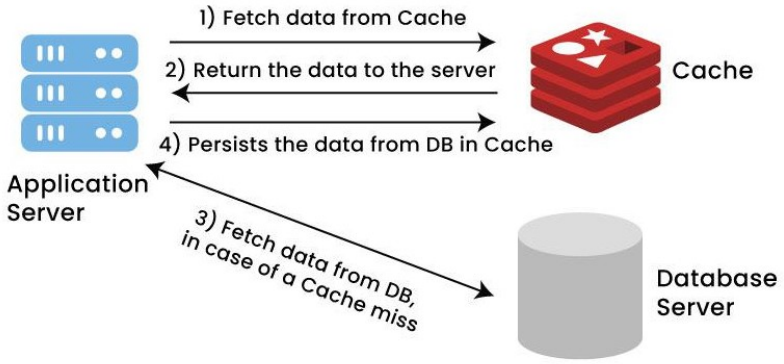

Figure 3: Web caching.

In this project `cache_memory` is a dictionary responsible for keeping cached data received from the server. Each requested URL is used as a key in the cache_memory dictionary, with its corresponding server response stored as a list of data chunks.
When a client requests a URL, the proxy first checks if it exists in the cache.

- If present, the stored response is directly sent to the client without contacting the origin server, minimizing latency and bandwidth usage.
- If absent, the proxy connects to the target server, forwards the request, streams the incoming response to the client in real time, and stores the received data in the cache for future requests.

This approach ensures faster load times for repeated requests and reduces unnecessary server load, while maintaining accurate and complete responses by caching the full server output.

2.4 Traceroute

Tracerouting is a network diagnostic technique used to determine the path packets take from a source machine to a specified destination host. It works by sending packets with incrementally increasing Time-To-Live (TTL) values. Each intermediate router decrements the TTL and returns an ICMP “Time Exceeded” message when the TTL reaches

zero. By collecting the IP addresses and round-trip times of these responses, tracerouting reveals each "hop" along the route, allowing network administrators to:

- Identify network bottlenecks or delays.
- Locate failures or misconfigurations along the route.
- Understand routing paths taken through the internet or a private network.

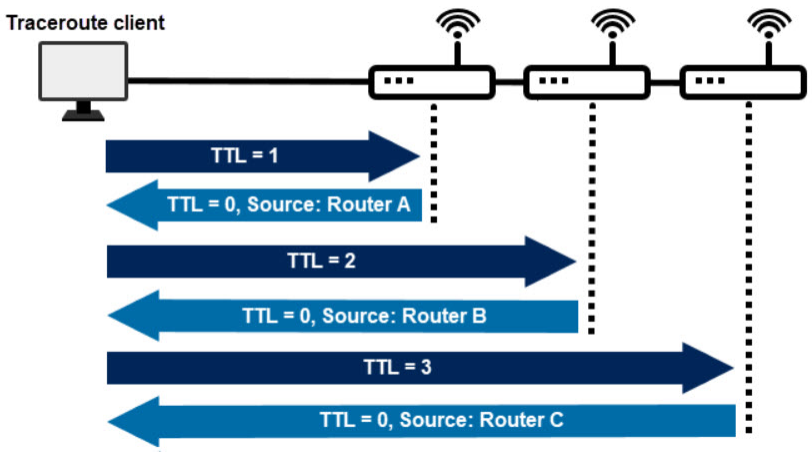

Figure 4: Traceroute.

I defined a class named `Traceroute` to run the traceroute in my code. It stores two main pieces of information:

- `target`: The hostname or IP address to trace.
- `hop`: The maximum number of hops (routers) the trace will follow.

In `set_values` the values of `target` and `hop` can be modified. `run` is used for For running the tracerout.

3 Testing and Evaluation

The functionality of the developed application was verified through a series of controlled tests. Each available command was executed individually to ensure correct behavior and expected output. The proxy server was tested for startup, shutdown, URL blocking, and cache clearing. The traceroute functionality was evaluated using different target hosts and hop limits to validate accuracy and reliability.

3.1 Commands

The implemented program provides an interactive command-line interface that allows the user to control a proxy server.

1. `Start Proxy:` Launches the proxy server in a separate background thread if it is not already running.
2. `Stop Proxy:` Stops the proxy server if it is currently active.
3. `Load Blocked URLs:` Loads a predefined list of URLs to be blocked by the proxy.
4. `Clear Blocked URLs:` Removes all URLs from the block list, allowing unrestricted access.
5. `Clear Cache:` Empties the proxy's cache memory to remove stored data.
6. `Run traceroute:` Prompts the user for a target address and hop limit, then performs a traceroute to display the path network packets take to the destination.

If an unrecognized command is entered, the program notifies the user of the invalid input.

3.2 Start Proxy

The `Start Proxy` command was executed to initiate the proxy server. This test assessed whether the server process could be started without error and begin listening for incoming connections.

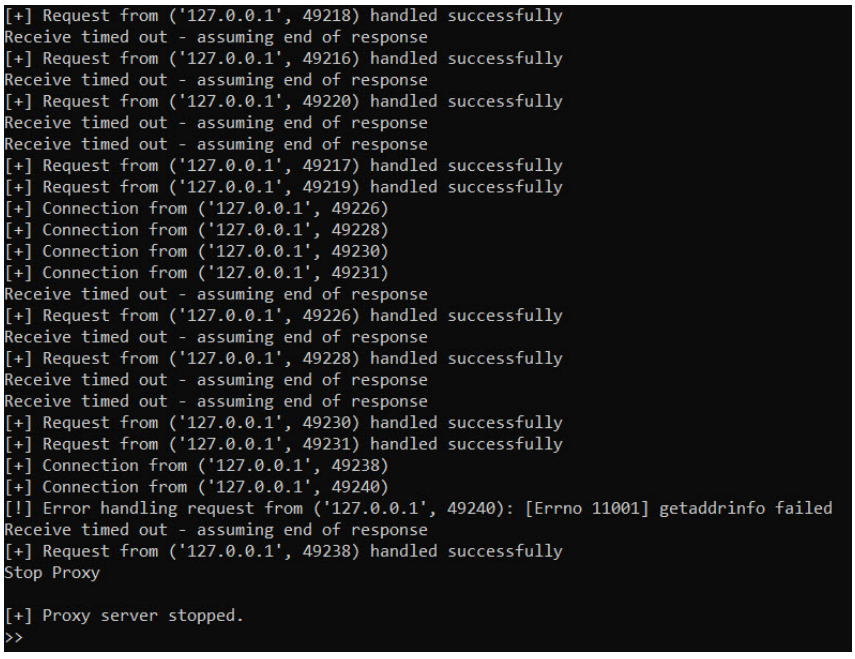

Figure 5: Start Proxy & Stop Proxy.

3.3 Stop Proxy

The `Stop Proxy` command was tested when the proxy server was active. This helped evaluate the program's ability to gracefully terminate a running process.

3.4 Load Blocked URLs

For this test, a sample file containing multiple domain names was loaded into the program using the `Load Blocked URLs` command. After Entering this request the program reads `blocked_urls.txt` file and display it. The program can immediately apply the blocking rules to live traffic.

```
[+] Proxy server stopped.
>> Clear Cache
[+] Cach cleared.
>> Load Blocked URLs
[+] Blocked URLs loaded:
http://www.example.com/page.html
http://www.testingmcafeesites.com/
http://www.testingmcafeesites.com/testcat_dr.html
http://www.testingmcafeesites.com/testcat_im.html
www.http2demo.io
http://detectportal.firefox.com/cononical.html
http://links.twibright.com/
http://httpforever.com/
>> Start Proxy
[+] Proxy server started.
[+] Proxy server started listening on 127.0.0.1:8080

>> [+] Connection from ('127.0.0.1', 49334)
[!] Blocked URL requested: http://httpforever.com/
[+] Connection from ('127.0.0.1', 49336)
[!] Blocked URL requested: http://httpforever.com/favicon.ico
```

Figure 6: Load Blocked URLs.

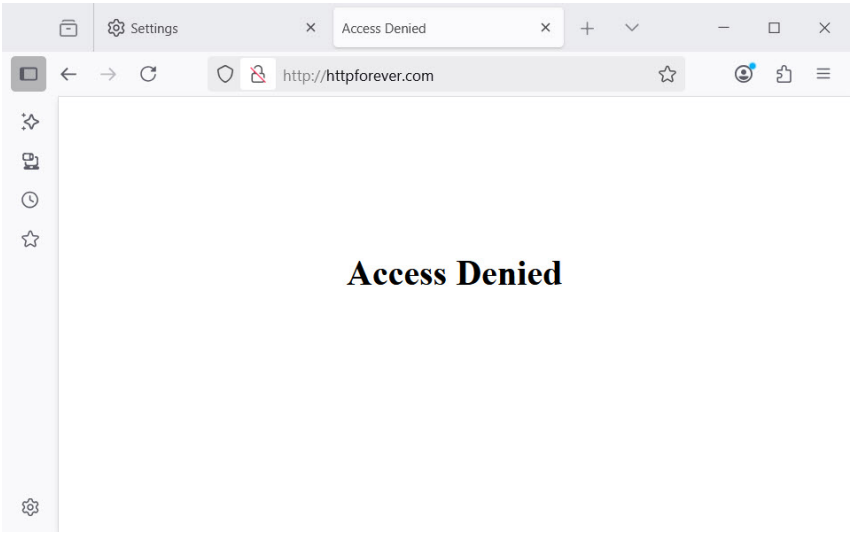

Figure 7: Access Denied HTML page.

3.5 Clear Blocked URLs

This test aimed to confirm that the `Clear Blocked URLs` command successfully removed all previously loaded entries from the internal block list. The ability to restore access to the formerly blocked websites was validated, and checks were performed to ensure the program's filtering logic was effectively disabled for these domains.

```
Clear Blocked URLs
[+] Block URLs cleared.
>> [+] Connection from ('127.0.0.1', 49337)
Receive timed out - assuming end of response
[+] Request from ('127.0.0.1', 49337) handled successfully
[+] Connection from ('127.0.0.1', 49342)
[+] Connection from ('127.0.0.1', 49344)
[+] Connection from ('127.0.0.1', 49345)
[+] Connection from ('127.0.0.1', 49346)
[+] Connection from ('127.0.0.1', 49347)
[+] Connection from ('127.0.0.1', 49348)
Receive timed out - assuming end of response
[+] Request from ('127.0.0.1', 49342) handled successfully
Receive timed out - assuming end of response
Receive timed out - assuming end of response
[+] Request from ('127.0.0.1', 49344) handled successfully
Receive timed out - assuming end of response
Receive timed out - assuming end of response
[+] Request from ('127.0.0.1', 49347) handled successfully
[+] Request from ('127.0.0.1', 49346) handled successfully
[+] Request from ('127.0.0.1', 49348) handled successfully
Receive timed out - assuming end of response
[+] Request from ('127.0.0.1', 49345) handled successfully
```

Figure 8: Clear Blocked URLs.

3.6 Clear Cache

Initially, the cache was left intact, and a previously visited page was refreshed. The request was served directly from the cache, resulting in noticeably shorter load times compared to the first visit (Fig.9). Afterwards, the `Clear Cache` command was executed, and the same page was refreshed again (Fig.5). This time, the loading process took as long as the initial visit, since all resources had to be retrieved from the original server instead of the local cache. This observation highlights the performance benefits of caching and the effect of its removal.

3.7 Run Traceroute

I ran the traceroute command by specifying a target host and hop limit. The output correctly displayed the network path with the expected number of hops, confirming that the feature works as intended.

```
[+] Connection from ('127.0.0.1', 49441)
[+] Request from ('127.0.0.1', 49441) handled successfully using cache.
[+] Connection from ('127.0.0.1', 49446)
[+] Request from ('127.0.0.1', 49446) handled successfully using cache.
[+] Connection from ('127.0.0.1', 49447)
[+] Connection from ('127.0.0.1', 49448)
[+] Connection from ('127.0.0.1', 49449)
[+] Connection from ('127.0.0.1', 49450)
[+] Connection from ('127.0.0.1', 49451)
[+] Connection from ('127.0.0.1', 49452)
[+] Request from ('127.0.0.1', 49447) handled successfully using cache.
[+] Request from ('127.0.0.1', 49448) handled successfully using cache.
[+] Request from ('127.0.0.1', 49449) handled successfully using cache.
[+] Request from ('127.0.0.1', 49450) handled successfully using cache.
[+] Request from ('127.0.0.1', 49451) handled successfully using cache.
[+] Request from ('127.0.0.1', 49452) handled successfully using cache.
[+] Connection from ('127.0.0.1', 49454)
[+] Connection from ('127.0.0.1', 49455)
[+] Connection from ('127.0.0.1', 49456)
[+] Connection from ('127.0.0.1', 49457)
[+] Request from ('127.0.0.1', 49456) handled successfully using cache.
[+] Request from ('127.0.0.1', 49455) handled successfully using cache.
[+] Request from ('127.0.0.1', 49457) handled successfully using cache.
Receive timed out - assuming end of response
```

Figure 9: Sending data from cache to the client.

```
>> Run traceroute
Enter target: httpforever.com
Enter number of hops: 5

Tracing route to httpforever.com [146.190.62.39]
over a maximum of 5 hops:

  1     2 ms     1 ms     1 ms  192.168.254.181
  2     *        *        *     Request timed out.
  3    76 ms    22 ms    26 ms  10.196.23.193
  4    74 ms    34 ms    25 ms  10.196.89.145
  5     *        *        *     Request timed out.

Trace complete.
```

Figure 10: Run Tracerout.