simge, sembol, logo, yazı tipi, ticari marka içeren bir resim

Açıklama otomatik olarak oluşturuldu yazı tipi, simge, sembol, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

**T.C.**

**MARMARA UNIVERSITY**

**FACULTY of ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

Computer Networks

Project Report

December 29, 2024

Group Members

Musa Özkan– 150121058

Hakkı Kokur – 150120033

Yasin Küçük – 150120062

1. **Project Summary**

In this project, we built **a multi-threaded HTTP server** and **a proxy server**, then **tested their performance** using **“wrk”.** The HTTP server spawns threads for each request, **dynamically returns HTML** documents based on URI size **(100–20,000 bytes),** and **logs all interactions**. It responds with the **appropriate HTTP error codes** for invalid sizes or methods. The proxy server, listening on **port 8888**, **forwards valid requests** to the HTTP server, **returns “414”** for URIs **over 9,999 bytes**, and **“404”** if the **server is unavailable**. It can also **cache content** using **LRU replacement policy** and **handle Conditional GET** requests. The “wrk” tests measured **latency**, **throughput**, and **concurrency** to evaluate overall system performance.

1. **Design Process**

Throughout our development, we made several high-level decisions that guided the overall architecture and functionality of both the HTTP server and the proxy server. Below is an overview of those key design choices, informed by our incremental progress as seen in the [Git commit history](https://github.com/hkokur/network-http-proxy-server-project.git):

A screenshot of a computer

Description automatically generated

1. **Clear Separation of Responsibilities:**

* We chose to keep the HTTP server and the proxy server as distinct modules. This separation allowed us to focus on each module’s requirements—such as request parsing, response generation, or request forwarding—without tangling the logic. This also made debugging and testing more straightforward, as each component could be verified independently.

1. **Thread-Based Concurrency**

* From the start, we opted for a multi-threaded design. Each client connection is handled by a dedicated thread, ensuring that slow or blocking operations on one request do not affect others. Our commit logs show an early emphasis on concurrency and refactoring to handle multiple requests reliably.

1. **Scalable Request Handling**

* Because our server had to handle URIs specifying a range of possible document sizes, we designed a flexible request-parsing scheme that cleanly validated input and preserved system stability even under invalid or extreme requests. We extended this validation logic into the proxy server to maintain consistent error handling, such as returning 414 Request-URI Too Long for overly large requests.

1. **Caching Strategy**

* Once the basic proxy functionality was established, we introduced a lightweight caching layer to boost performance. We selected an LRU-style approach, as reflected in later commits labeled “Caching and some comment improvements” and “Fixes on cache,” to efficiently evict the least recently used entries. This choice balanced simplicity with the ability to handle workloads where repeated requests could benefit from cached responses.

1. **Conditional GET Logic:**

* Our final design choice was to incorporate Conditional GET for bonus functionality, distinguishing between “modified” and “unmodified” content using the parity (even/odd) of the file length. This design, noted in commits like “Implemented Conditional GET with relevant conditions,” kept the logic simple yet demonstrated how the proxy could cooperate with the server to reduce bandwidth and unnecessary round-trips.

By evolving our system in incremental steps—first building a robust multi-threaded server, then adding proxy capabilities, followed by caching and conditional validations—we ensured each core design decision was tested and integrated as we moved forward. This methodical process (as evidenced in our Git history) kept the code base maintainable and allowed us to refine each design choice based on real-world testing and feedback.