Report-1 - Rudy: a small web server

Md Sakibul Islam

September 10, 2025

1 Introduction

This report documents the creation and performance testing of a small web server called Rudy, written in the Erlang programming language. The tasks required making an HTTP parser and setting up a server able to respond to requests and run benchmarks to check for performance. The tests were performed without delay and with a 40 ms delay introduced artificially, as well as using several terminals simultaneously to mimic multiple clients. Results present very high-speed response by rudy(server) when there exists no delay, evidently influenced by a 40ms delay on the speed of request handling. Testing with various terminals shows the same results and provides a rather useful perception of the current server behavior.

1.1 Problem Statement

By accomplishing this task, hands-on experience was gained in building a web server from scratch in Erlang. There were three main objectives of the assignment:

- Understanding the basics of HTTP request parsing.
- Learn how to use Erlang's gen_tcp socket API.
- Structure the server that can accept and respond to client connections.

To measure the time of server, the given benchmark program (test) is run to send repeated requests and check how quickly the responses were returned. The delay was also added to the server which see the difference of waiting time that could occur when reading from files or running server-side scripts.

2 Methodology

I have done 3 things into the boilerplate.

- In http file, I implemented the parser for HTTP "GET" requests with the helper functions to construct responses.
- At server file, I added the part of accepting connections, parses the requests and sends back the responses. A delay of 40 ms in the reply function as per instructions.
- In test file, the benchmark client that connects to rudy which sends 100 requests and track the total time of it.

Benchmarks were performed in three different ways. Initally without any delay then added the 40ms delay and lastly running from 5 terminals at the same time, each sending 100requests.

3 Results

3.1 Without Artificial Delay

Time for 100 requests: **35123** μ**s** (≈ 0.035 s).

Requests per second:

$$\frac{100}{0.035123} \approx 2848\,\text{req/s}$$

3.2 With Artificial Delay (40ms)

Time for 100 requests: **4182915** μ**s** (≈ 4.18 s).

Requests per second:

$$\frac{100}{4.18} \approx 24\,\text{req/s}$$

3.3 Five Terminals in Parallel (With Delay)

When running five benchmark clients at the same time, each sending 100 requests:

Terminal	**Time (μs)**	**Time (s)**	**Requests/sec**
T1	4203018	4.203	23.8 req/s
T2	4201367	4.201	23.8 req/s
T3	4201595	4.202	23.8 req/s
T4	4206921	4.207	23.8 req/s
T5	4174217	4.174	23.9 req/s

Table 1: Benchmark results with five parallel terminals

All five terminals reported results in the same range as the single-terminal test with delay.

4 Bonus : File Serving

I have added file serving to make Rudy more extensive and useful. I have enhanced the server to serve actual files (HTML, CSS, JS, text) instead of static messages. Also added automatic content-type detection based on file extensions. Implemented proper HTTP headers (Content-Type Content-Length). I handled URL query parameters correctly. Hence it successfully served multiple file types to web browsers.

5 Conclusion

After implementing Rudy it revealed two key insights like server achieves exceptional throughput (2800+ requests/second) when unhindered by I/O operations,when unhindered by I/O operations, but performance plummets to just 24 requests/second when introducing realistic 40ms delays, demonstrating that I/O latency is the fundamental bottleneck in web serving. Other than that this constraint, the task successfully evolved from a basic HTTP protocol exercise into a fully functional file-serving web server capable of handling multiple content types and concurrent connections, validating Erlang's strength for building scalable network services while highlighting the critical importance of optimizing I/O operations in real-world deployments.