

RockNet

Ajay Kumar¹, Thapisnu², Raguram³, Mrs. A. Nalini⁴, Mr. M. Nadeesh⁵

^{1,2,3}Diploma Student, Computer Engineering, Nachimuthu Polytechnic College, Pollachi, Tamil Nadu, India

⁴Head of the Department, Department of Computer Engineering, Nachimuthu Polytechnic College, Pollachi, Tamil Nadu, India

⁵Lecturer, Department of Computer Engineering, Nachimuthu Polytechnic College, Pollachi, Tamil Nadu, India

Abstract

Day by day, the number of users is increasing on the internet, and the web servers need to cater to the requests constantly. Also, if compared to the past years, this year, due to a global pandemic and lockdown in various countries, the requests on the web have surged exponentially. The complexity of configuring a web server is also increasing as the development continues. In this paper, we propose a RockNet web server, which is highly scalable and can cater to many requests at a time. Additionally, to ease users from the configuration of the web server, we introduced a Graphical User Interface which is beginner-friendly.

Keywords: Rust, HTTP, Web Server, GUI, Asynchronous, Concurrency.

1. Introduction

Whenever a user accesses the web, he/she uses an HTTP client program called a web browser to fetch contents from what's called a web server, which is an HTTP server program. HTTP client and server normally use TCP as the transport layer protocol. Whenever a user visits a particular website, the browser will send an HTTP GET request with some metadata stored in HTTP request headers to the web server. The web server will parse the request and prepare the HTTP response. If the server doesn't find the requested content, then it will respond with a 404-status code. Alternatively, if the server finds the requested content, it will respond with a 200-status code. There are many types of status codes, i.e., Informational, Successful, Redirects, Client errors, and Server errors. Websites can also be dynamically generated, which is integrated with some database engines, but in this paper, we'll be going to limit our discussion to static websites only.

Traffic on web servers is increasing rapidly due to many clients connecting to the web server in such a short time interval. That's why the server must cope with many requests at a time, therefore support for concurrency and parallelism is indispensable. There are many concurrency models available, but each one has its drawbacks. Configuring a web server is also a tedious task and, if not configured properly, it can be vulnerable to security attacks.

Considering the above problems, in this paper, we introduce the RockNet web server that can handle many requests at a time by utilizing multiple cores of the processor and also make use of the non-blocking IO operations provided by the operating system. We have also solved the complication of configuration by introducing a lightweight, user-friendly GUI.

2. BACKGROUND

A. HTTP Protocol

Hyper Text Transfer Protocol is one of the most widely used web protocols. It is a stateless protocol in which each transaction is independent of all other transactions. There are HTTP clients, which in most cases are web browsers, that initialize the transactions by sending HTTP requests, which are replied to by the HTTP server's response. Mainly there are 2 types of HTTP requests: GET & POST. GET is used to fetch content from the server, and POST is used to submit content to the server. In HTTP requests, there are numerous headers like the method, which is used to indicate whether the request type is GET or POST, URI or HOST, which is used to indicate the server's domain name and, in some cases, the requested file, version, which is used to indicate HTTP protocol versions. On the other hand, HTTP response contains headers like status code, which is used to denote the result of the transaction, content-type, which specifies what kind of file is transferred, and server, which contains server name and version.

There are multiple versions of the HTTP protocol [1], namely HTTP/1.0, HTTP/1.1, HTTP/2, HTTP/3, in which HTTP/3 is still in the development phase. HTTP/1.1 supports pipelining, virtual hosting, and chunked responses. HTTP/2 is faster than HTTP/1.1 and supports additional features such as server push and multiplexing.

B. HTTP over TLS (HTTPS)

For encrypting transactions between client and server, HTTPS is used. Secure HTTP or HTTPS is achieved using the Transport Layer Security protocol. For RSA-based encryption process, the client first sends a Client Hello message with some random bytes called "Client Random," and the server replies with a Server Hello message with some random bytes called "Server Random" and an SSL certificate for authentication purposes. Now the client creates a "premaster secret," which is also random bytes but encrypted with the server's public key and sends it to the server. The server then decrypts the premaster secret, and both client and server create a session key based on client random, server random, and premaster secret. Now all the communication between client and server will be encrypted with a session key. There is another encryption process called Diffie-Hellman, which is moderately different than RSA.

C. Virtual Hosting

Hosting every single website with an individual server is quite expensive and inefficient. That's why support for virtual hosting is necessary. Through virtual hosting, one can host many websites using a single web server. There are mainly 2 types of virtual hosting: Port-based & Domain Name-based. Using port-based virtual hosting, one can utilize multiple ports for each website hosted on a single machine, and therefore only a single IP address is used. In domain-based virtual hosting, a single port and IP address are utilized, and the domain name of the website is used to distinguish between multiple websites. There is also an IP-based virtual hosting to host multiple websites on a single machine but utilizing multiple IP addresses.

3. WORKING AND IMPLEMENTATION

RockNet web server is developed to ease the configuration of websites in development environments as the other web servers in the industry only support terminal-based configuration, which can be time-consuming for beginners. On the other hand, RockNet supports Graphical User Interface (GUI), allowing users to easily host websites in a development environment and focus more on website development

rather than server configuration. The RockNet web server consists of two major components: RockNet Core and RockNet GUI.

A. RockNet

This module constitutes the core functionality of the RockNet web server. Its primary responsibility is to read the configuration file, either generated by the RockNet GUI or provided by the user, and allocate resources accordingly for each website mentioned in the file. RockNet supports both HTTP and HTTPS protocols. Unencrypted HTTP communication uses the HTTP/1.1 version, while encrypted HTTPS communication is powered by the more efficient HTTP/2.0 version. Within the Rust ecosystem, various crates enhance the program's functionality. For RockNet, the Hyperium H2 crate implements the HTTP/2.0 specification, and the rustls crate facilitates transport layer security (TLS) operations. With rustls, RockNet can perform the TLS 1.3 handshake, significantly reducing encryption setup time. TLS 1.3 requires only one round-trip to establish a connection, compared to two round-trips needed in TLS 1.2 [3].

Additionally, RockNet employs the TLS extension Application-Layer Protocol Negotiation (ALPN) [4], which streamlines protocol selection during the TLS handshake. By negotiating HTTP/2 directly within the handshake, ALPN eliminates the need for the extra round-trip required by the traditional Connection-Upgrade process, enhancing communication efficiency. As described earlier, RockNet supports both port-based and domain name-based virtual hosting.

1. Port-Based Virtual Hosting: RockNet creates separate kernel-level threads for each website, isolating individual websites' behaviour and optimizing the use of multi-core processors by distributing the load across cores.
2. Domain-Based Virtual Hosting:
 - For HTTP communication, RockNet utilizes the HOST header in requests to distinguish between domain names.
 - For HTTPS communication, the server leverages the TLS extension Server Name Indication (SNI) to identify the SSL certificate and private key for each website dynamically at runtime. This enables each virtually hosted website to use a unique SSL certificate and private key for secure communication.
3. RockNet also incorporates a key feature of HTTP/2: Server Push [5][6]. This allows the server to send resources proactively without waiting for client requests, reducing latency and improving throughput by bundling necessary dependencies with the response. However, the server must carefully select the files for Server Push to avoid unnecessary bandwidth consumption if the client does not require the sent resources.

B. RockNet GUI

This module is isolated from the RockNet core as it contains various GUI elements that are independent of the server itself. As discussed earlier, configuring a web server can be daunting for beginners. A GUI-based configuration tool simplifies this process, making it more accessible. The primary function of the GUI is to generate the configuration file, which acts as a bridge between the RockNet core and the RockNet GUI. Unlike popular web servers in the industry that rely on complex configuration file formats such as XML, JSON, or YAML—known for being error-prone and difficult to manage—RockNet adopts the TOML (Tom's Obvious Minimal Language) format. TOML is straightforward, easy to understand, and simple to parse, making it an ideal choice for this purpose [7].

The RockNet GUI is developed using the FLTK (Fast Light Tool Kit) framework in the Rust program.

g language, making it lightweight. The GUI has a low memory footprint, ranging between 45–50MB, and produces a compact binary of approximately 5MB. Including this small binary in the installation package has a negligible impact on its size. Additionally, since FLTK and Rust are cross-platform, the GUI and the entire RockNet web server support all major operating systems.

The GUI provides additional functionalities such as monitoring the web server's status, including the current CPU load and memory usage. It also aids in troubleshooting by enabling users to read or modify log levels to more verbose settings, helping identify specific issues. With the GUI, users can avoid directly interacting with the configuration file. Instead, the GUI dynamically lists all hosted websites and their parameters as defined in the configuration file, streamlining server management.



Figure 1. RockNet Web Server GUI Dashboard

C. Concurrency Model

There are various concurrency models or design patterns available to handle multiple requests simultaneously. One simple approach is the Thread Pool Model, where a group of pre-spawned threads is prepared to handle incoming requests. When a request arrives, one of the available threads is assigned to process it. This enables the server to handle multiple requests in parallel, with the number of requests served concurrently limited by the initial number of spawned threads. While this model can improve throughput, it has some drawbacks. The number of threads is fixed, and creating threads incurs operating system overhead. Additionally, if the number of requests is much lower than the number of threads, many threads will remain idle, leading to a waste of computational resources.

Another approach is the Event-Driven Model, in which servers handle multiple requests using non-blocking IO operations provided by the operating system. In this method, a single thread functions as an event scheduler. Its tasks include accepting connections, reading requested files, and sending responses back to the client. All operations are asynchronous and non-blocking, allowing the thread to avoid waiting for any single event to complete. Instead, the operating system notifies the thread upon event completion. The event-driven model has a significant advantage in resource efficiency, but its effectiveness depends on the operating system's support for non-blocking IO. While most modern operating systems like Linux and Windows offer non-blocking support for network IO, their support for non-blocking disk IO is either lacking or experimental. Consequently, some operations may become synchronous and blocking, leading to potential performance degradation.

The RockNet web server is based on the Tokio framework [8], which is a runtime for writing asynchronous, event-driven, and non-blocking applications in the Rust programming language. Tokio is based on AMPED or Asymmetric Multi-Process Event-Driven architecture, as described in [9]. This architecture is an extension of the above-described event-driven architecture and resolves its major constraints by

introducing worker threads. By assigning blocking or synchronous tasks to the worker thread, the event scheduler remains free to assign other upcoming requests to available worker threads. In this way, we can achieve maximum throughput and efficiency. The default number of worker threads is equal to the number of cores available in the working system.

Tokio is not limited to AMPED architecture because it also supports other features like a work-stealing queue, in which the load amongst worker threads is distributed uniformly. This ensures that all the CPU cores are equally utilized.

4. RESULT AND DISCUSSION

To evaluate and benchmark the performance of RockNet and Apache (a widely used web server in the industry) [10], we utilized the Apache Benchmark (AB) tool [11] to perform load testing and stress testing on the servers. The tests were conducted on a host system with the following configuration: AMD E300 Dual Core CPU clocked at 1.3GHz, 4GB RAM running at 800MHz, and Arch Linux with Kernel version 6.10.43. It is important to note that the results are dynamic and may vary depending on factors such as system configuration, ambient temperature, and operating system.

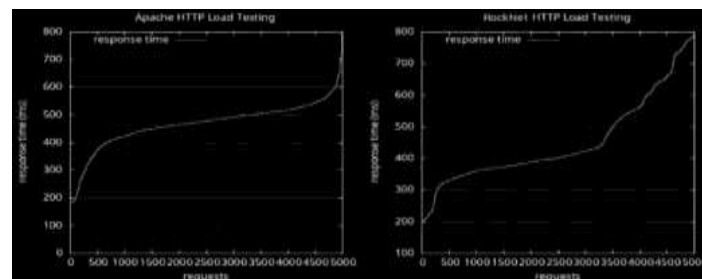


Figure 2. Graphical comparison of response time with respect to HTTP requests

In this experiment, both servers were subjected to 5000 unencrypted HTTP requests, with 500 requests sent concurrently in each batch, resulting in 10 batches of 500 requests. This test aimed to observe how the servers handle a high volume of concurrent requests and how their response times are affected during this period. From the analysis of the graph, it is evident that as the number of requests reaches the range of 750 to 800, the response time of the Apache web server begins to rise sharply, reaching approximately 400ms. This behaviour highlights a critical point where the server's ability to manage concurrent requests starts to degrade significantly.

By analyzing the above graphs, we can see that when the number of requests reaches 750-800, the response time of Apache web is increasing rapidly around 400ms. The RockNet web server demonstrates a response time of 340ms, which is significantly lower than the Apache web server. When the number of requests increases from 1000 to 3500, the response time for the Apache web server rises from 410ms to 500ms. In contrast, under the same conditions, RockNet's response time increases from 340ms to 450ms, still outperforming Apache. However, when the number of requests exceeds 3500, RockNet's response time begins to grow faster than Apache's. At 5000 requests, both servers converge at a response time of 800ms.

5. CONCLUSION & FUTURE SCOPE

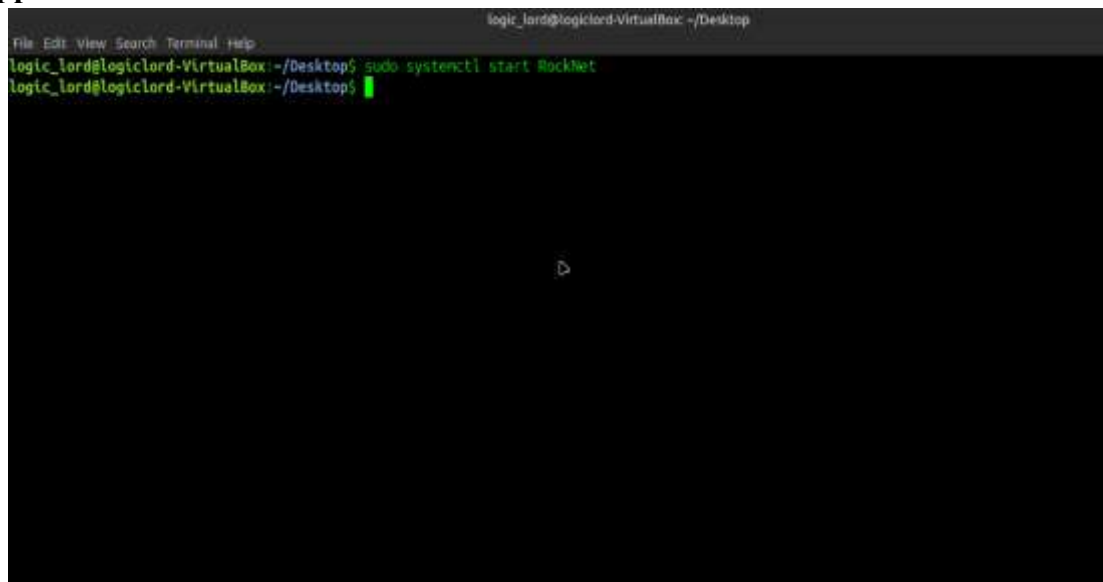
This paper introduces RockNet, a new web server developed using the Rust programming language, diverging from the traditional use of C in prominent industry web servers. Rust's memory safety, scalabil-

ity, and speed enhance RockNet's robustness, making its performance comparable to or exceeding its counterparts. Leveraging Tokio's asynchronous, event-driven, and work-stealing architecture, RockNet maximizes concurrency while efficiently managing system resources. To simplify server setup for beginners, RockNet incorporates a Graphical User Interface (GUI), enabling easy configuration within development environments and allowing users to focus on website creation.

The implementation of the Common Gateway Interface (CGI) facilitates PHP integration, enabling the delivery of dynamic content over the web. Additionally, utilizing the `io_uring` subsystem, introduced in Linux Kernel 5.1, allows for highly efficient, non-blocking asynchronous disk operations, improving concurrency and throughput. While Windows does not yet support `io_uring`, its implementation is expected in version 21H2, as referenced in [12].

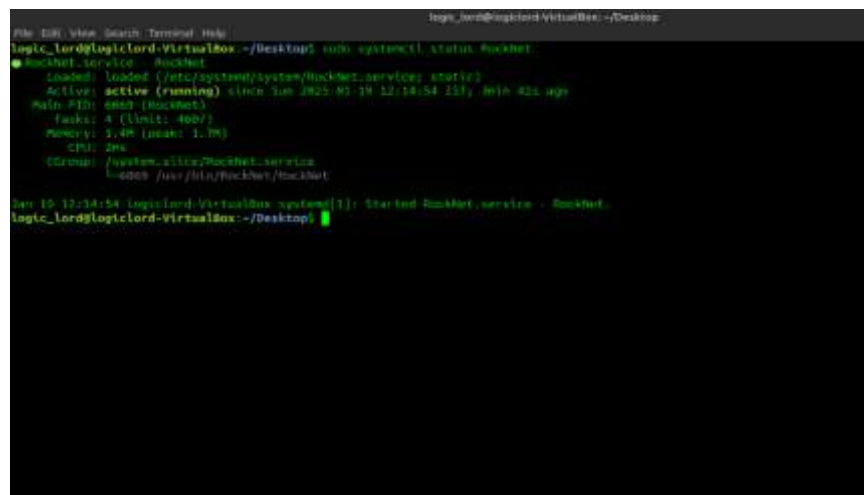
RockNet also supports HTTP/3, the upcoming version of the HTTP protocol, which replaces TCP with QUIC-UDP at the transport layer. This transition significantly enhances communication performance between clients and servers, reducing response times and latency.

6. Appendix



```
logic_lord@logiclord-VirtualBox: ~/Desktop
File Edit View Search Terminal Help
logic_lord@logiclord-VirtualBox:~/Desktop$ sudo systemctl start RockNet
logic_lord@logiclord-VirtualBox:~/Desktop$
```

[STARTING ROCKNET]



```
logic_lord@logiclord-VirtualBox: ~/Desktop
File Edit View Search Terminal Help
logic_lord@logiclord-VirtualBox:~/Desktop$ sudo systemctl status RockNet
● RockNet.service - RockNet
   Loaded: loaded (/etc/systemd/system/RockNet.service; static)
   Active: active (running) since Sun 2025-05-19 12:14:54 IST; 36s ago
     Main PID: 6883 (RockNet)
       Tasks: 4 (limit: 4607)
      Memory: 5.4M (peak: 1.7M)
         CPU: 2ms
    CGroup: /system.slice/RockNet.service
            └─6883 /usr/bin/rocknet/rocknet

Jan 10 12:14:54 logiclord-VirtualBox systemd[1]: Started RockNet.service - RockNet.
logic_lord@logiclord-VirtualBox:~/Desktop$
```

[STATUS OF ROCKNET]

```
logic_lord@logiclord-VirtualBox: ~/Desktop
File Edit View Search Terminal Help
logic_lord@logiclord-VirtualBox:~/Desktop$ sudo mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 0.0.40-0ubuntu0.24.04.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

[MySQL]

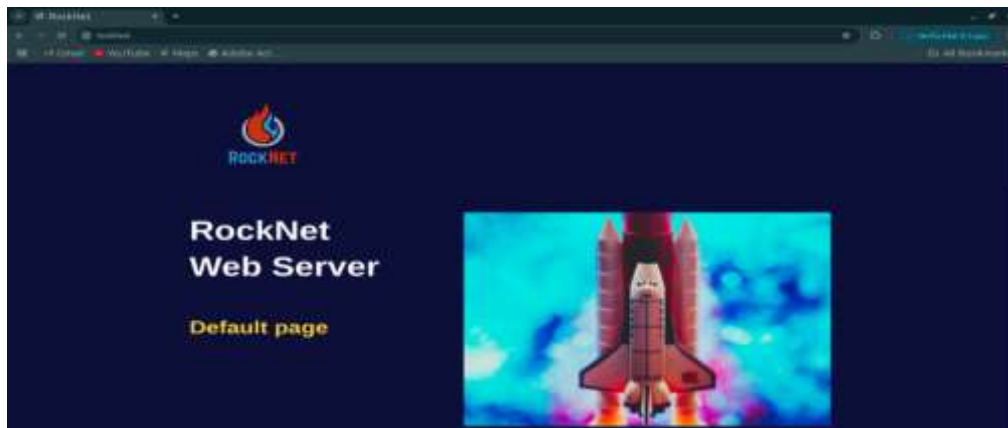
```
logic_lord@logiclord-VirtualBox: ~/Desktop
File Edit View Search Terminal Help
logic_lord@logiclord-VirtualBox:~/Desktop$ sudo systemctl status php8.2-fpm
● php8.2-fpm.service - The PHP 8.2 FastCGI Process Manager
   Loaded: loaded (/usr/lib/systemd/system/php8.2-fpm.service; enabled; preset: enabled)
   Active: active (running) since Sun 2025-01-19 18:37:35 IST; 26min ago
     Docs: man:php-fpm8.2(8)
   Process: 975 ExecStartPost=/usr/lib/php/php-fpm-socket-helper install /run/php/php-fpm.sock /etc/php/8.2/fpm/pool.d/www
 Main PID: 957 (php-fpm8.2)
   Status: "Processes active: 0, idle: 2, Requests: 0, slow: 0, Traffic: 0.00req/sec"
    Tasks: 0 (limit: 4097)
  Memory: 12.7M (peak: 13.4M)
     CPU: 96ms
   CGroup: /system.slice/php8.2-fpm.service
           └─977 "php-fpm: master process (/etc/php/8.2/fpm/php-fpm.conf)"
             └─972 "php-fpm: pool www"
               └─974 "php-fpm: pool www"

Jan 19 18:37:35 logiclord-VirtualBox systemd[1]: Starting php8.2-fpm.service - The PHP 8.2 FastCGI Process Manager...
Jan 19 18:37:35 logiclord-VirtualBox systemd[1]: Started php8.2-fpm.service - The PHP 8.2 FastCGI Process Manager.
lines 1-17/17 (END)
```

[STATUS OF PHP FPM]



[PHP RENDER]

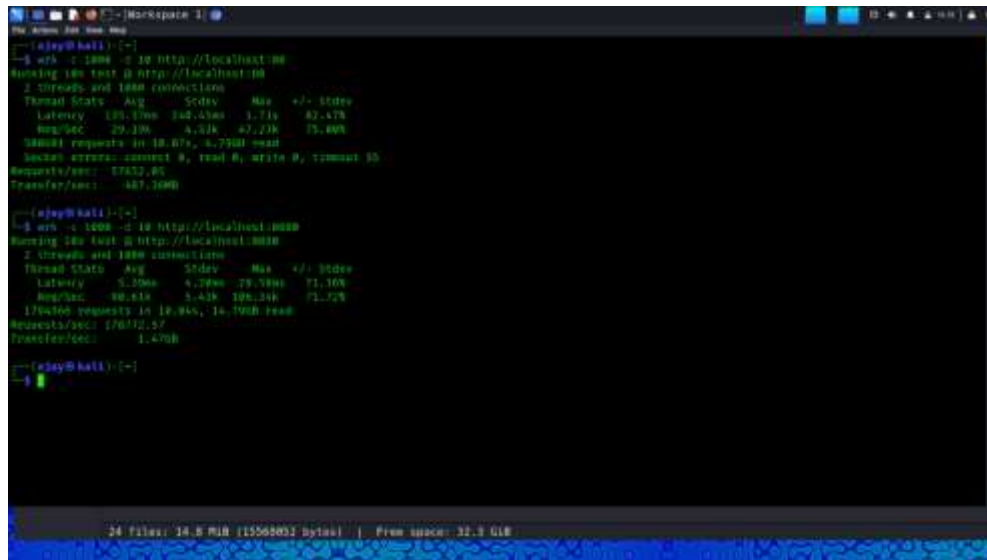


[ROCKNET'S DEFAULT PAGE]



[ROCKNET'S DASHBOARD]





```

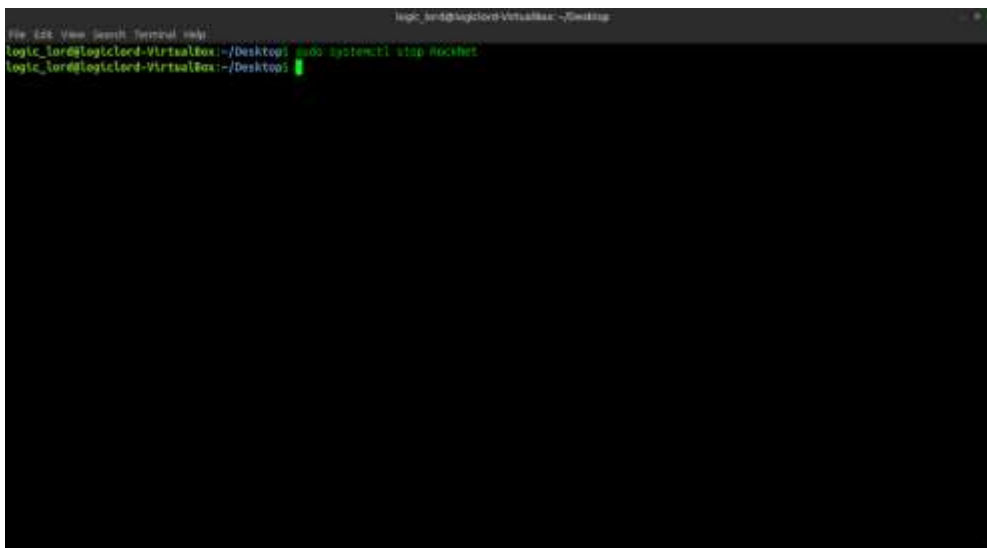
root@kali:~# ab -n 1000 -c 10 http://localhost:8080
Running 100 tests @ http://localhost:8080
2 threads and 1000 connections
Thread Stats Avg Stdev Max +/- Stdev
Latency 139.15ms 142.43ms 3.71k 83.47%
Req/Sec 29.10k 4.31k 47.23k 75.80%
50000 requests in 10.07s, 4.750M read
Socket errors: connect 0, read 0, write 0, timeout 0
Requests/sec: 17642.05
Transfer/sec: 487.10MB

root@kali:~# ab -n 1000 -c 10 http://localhost:8080
Running 100 tests @ http://localhost:8080
2 threads and 1000 connections
Thread Stats Avg Stdev Max +/- Stdev
Latency 139.15ms 142.43ms 3.71k 83.47%
Req/Sec 29.10k 4.31k 47.23k 75.80%
50000 requests in 10.07s, 4.750M read
Socket errors: connect 0, read 0, write 0, timeout 0
Requests/sec: 17642.05
Transfer/sec: 487.10MB

root@kali:~#

```

[BENCHMARK OF ROCKNET & APACHE]



```

logic_lord@logiclord-VirtualBox: ~/Desktop
logic_lord@logiclord-VirtualBox:~/Desktop$ sudo systemctl stop rocknet
logic_lord@logiclord-VirtualBox:~/Desktop$

```

[STOPPING ROCKNET]

Book References

1. Fulton, K. R., Chan, A., Votipka, D., Hicks, M., & Mazurek, M. L. (2021). Benefits and Drawbacks of Adopting a Secure Programming Language: Rust as a Case Study. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)* (pp. 597-616). USENIX.
2. Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. (2020). Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. *SIGCOMM Comput. Commun. Rev.*, 50(3), 3–15. DOI: 10.1145/3411740.3411742.
3. Shoemaker, R. B. (2020). RFC 8737 Automated Certificate Management Environment (ACME) TLS Application Layer Protocol Negotiation (ALPN) Challenge Extension.
4. Apache Software Foundation. (2020). Apache HTTP Server Project ([httpd](http://httpd.apache.org)). <https://httpd.apache.org>.
5. Apache Software Foundation. (2020). ab-Apache HTTP server benchmarking tool. <https://httpd.apache.org/docs/2.4/programs/ab.html>.

6. Jonathan, H. (2020, July). The 3 Best Config File Formats. <https://jhall.io/posts/best-config-file-formats>.
7. Zimmermann, T., R  th, J., Wolters, B., & Hohlfeld, O. (2017, June). How HTTP/2 pushes the web: An empirical study of HTTP/2 server push. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops* (pp. 1-9). IEEE.
8. Rosen, S., Han, B., Hao, S., Mao, Z. M., & Qian, F. (2017, April). Push or request: An investigation of HTTP/2 server push for improving mobile performance. In *Proceedings of the 26th International Conference on World Wide Web* (pp. 459-468).
9. Abdullah, S. A., & Ahmad, A. M. (2016). HTTP/2 in Modern Web and Mobile Sensing-based Applications: Analysis, Benchmarks, and Current Issues. *J Electrical & Electronic System*, 5, 193.
10. Pai, V. S., Druschel, P., & Zwaenepoel, W. (1999, June). Flash: An efficient and portable Web server. In *USENIX Annual Technical Conference, General Track* (pp. 199-212).

11. Web References

1. **Tokio Framework -rs.** (2021). *Tokio Framework Runtime*. Retrieved from <https://Tokio Framework.rs>
2. **Actix-rs.** (n.d.). *Actix Documentation*. Retrieved from <https://actix.rs/docs/>
3. **PHP-FPM.** (n.d.). *PHP-FPM Documentation*. Retrieved from <https://php-fpm.org/>
4. **Plotters.** (n.d.). *Plotters Documentation*. Retrieved from <https://plotters-rs.github.io/home/#!/>