Impact of Rate Limiting Algorithms on Server Energy Efficiency Under Varying Loads

Program: Graduate
Arnav Chahal (chahaa1, arnav.chahal@vanderbilt.edu)
Syed Ali (alisa2, syed.a.ali@vanderbilt.edu)
Zayaan Rahman (rahmanmz, muhammad.z.rahman@vanderbilt.edu)

Dec 4, 2024

Overview. With the prevalence of high-traffic web services, it is becoming increasingly important to consider both request limits and energy usage to minimize the costs of software engineering. This project aims to measure and compare the energy efficiency of HTTP servers implementing different rate limiting algorithms under different load patterns. A client process mimicking different load patterns and a server implementing different rate limiting algorithms will be developed, and the efficiency of each load-algorithm pair will be measured and compared. The goal of this research is to identify algorithms that are particularly energy efficient under relevant load types. Additionally we aim to provide a set of tools that will test the efficiency of rate-limiting algorithms.

Intellectual Merit. Though rate limiting algorithms have been rigorously developed and studied, most research conducted on them aims to optimize their performance in terms of requests served, response time, and CPU/RAM usage [1]. Similarly, there is a large body of research on optimizing the energy efficiency of servers [7]. However, even though almost all modern web servers under high traffic implement robust rate limiting, there is little research on how rate limiting can impact energy usage. With server costs making up a substantial percentage of operational costs for tech companies, it is crucial to identify any areas where energy usage can be allocated more effectively [6].

This study primarily focuses on identifying the behavior of different algorithms in terms of their energy usage under different load patterns. The rate limiting step will constitute a large fraction of the total work done, because the work performed by the requests will be minimal. The simple server-client setup may help highlight the energy usage trade offs of different rate limiting algorithms, providing an additional metric to consider when choosing an algorithm for a real-world web server.

In our proposed experiment, clients will send GET and POST HTTP requests at rates, for fixed periods of time, mimicking the inherent variability in web services. We used an isolated server to limit the CPU and RAM, and we also used it to get an accurate measure of the requests and power usage. The server and client will be written in Golang, for its rising prevalence in distributed systems and simplicity, allowing for clean, zero-dependency implementations of the necessary testing code. By conducting multiple trials for each algorithm and load pattern pair, variation in performance can be weighed and factored into the final analysis.

Broader Impact. This research introduces the idea that rate limiting may be a significant contributor to energy usage in high traffic web servers. Exploring how different rate limiting algorithms scale differently in terms of ratio of requests processed to energy use could highlight opportunities to save costs through reducing power consumption in server centers. Even small percentage differences in different algorithms could translate to substantial savings on a large enough scale.

Additionally, this project will produce a set of simple tools to model clients and servers. A client capable of varying its load pattern could be beneficial in mimicking real behavior in this field of research. The implementation of a simple web server that easily integrates rate limiting as handler functions could also simplify the implementation workflow of future research. Finally, the use of popular distributed service technologies such as Golang can provide a closer representation to real scenarios.

1 Introduction

Rate limiting algorithms are the backbone of modern technology systems, essential for maintaining the stability and security of APIs (Application Programming Interfaces). These algorithms prevent systems from breaking under the strain of heavy traffic or malicious attacks like Distributed Denial-of-Service (DDoS) attacks. These algorithms are extensively used across industries—particularly in cloud services—and their functionality are integral to maintaining usable web services.

At their core, rate limiting algorithms regulate the flow of network requests, ensuring that systems do not become overwhelmed during periods of high traffic. Without rate limiting, APIs would be vulnerable to overload, resulting in server failures and degraded user experiences. Additionally, in the face of cyberattacks, rate limiting serves as a first line of defense by restricting the volume of traffic that can access a server, protecting it from malicious actors. Therefore, they are essential in most modern web applications.

However, despite the crucial role rate limiting algorithms play, there is limited research focusing on their computational costs, specifically regarding energy consumption under different load conditions. This study seeks to fill that gap by examining how various rate limiting algorithms impact power consumption while maintaining system throughput. By evaluating power consumption and throughput under different load scenarios—such as high network traffic and DDoS attacks—this research aims to provide insights into which algorithms are most energy efficient.

2 Background and Related Work

The primary research in the field of rate limiting algorithms has focused on controlling API traffic and balancing loads during high traffic conditions. Algorithms such as Token Bucket, Leaky Bucket, Fixed Window, and Sliding Window help manage traffic, particularly during surges or malicious attacks. However, limited research explores the computational efficiency of these algorithms, especially in terms of power consumption and resource usage, which this study seeks to address.

Constant traffic typically occurs in systems with steady user engagement, such as streaming services or web applications. The traffic remains relatively stable for most periods of time. This scenario is mostly seen in data centers that handle predictable, ongoing workloads.

Burst traffic happens when there is a sudden surge in network requests. These bursts can be triggered by events like flash sales on e-commerce platforms, viral content, or Distributed Denial-of-Service (DDoS) attacks. Bursts often occur due to network protocol behaviors such as TCP's slow start or congestion recovery mechanisms, where segments of data are rapidly sent after congestion is detected and resolved [3].

Gradually increasing network traffic occurs when the traffic volume slowly grows over time, often as user activity ramps up, such as during peak hours on a network. Systems that experience this kind of load include online services, where usage increases as more users log in.

This study focuses on several key rate-limiting algorithms. The Token Bucket algorithm processes a fixed number of requests based on available tokens. When a request is processed, a token is consumed, and once tokens run out, further requests are blocked until more tokens become available. This mechanism handles bursts of traffic within a set limit. The Leaky Bucket algorithm processes requests at a constant rate, discarding any exceeding the allowed rate to ensure consistent traffic [5]. The Fixed Window algorithm permits a set number of requests within a fixed time interval, rejecting any additional requests during that period. Lastly, the Sliding Window algorithm allows a fixed number of requests within a moving time window, dynamically adjusting as new requests arrive and older ones "slide out" [2]. This helps maintain a steady traffic flow without overloading the server.

One of the motivations for this proposal is the expenses for powering data centers and servers. The energy cost to power a single server rack annually in a U.S. data center can reach $30,000 per year [6]. As data centers grow and adopt higher power densities, the cost increases significantly. For example, a data center with 100 racks can incur over $3 million annually in energy expenses. Managing energy efficiently is critical to reducing operational costs and ensuring sustainability [6].

3 Problem Statement

We aim to identify which algorithms maximize energy efficiency (measured as requests processed divided by power used over a time period) under different load patterns. We will use our framework to implement the client and server software components, and will analyze the resulting efficiency values to determine whether they are statistically different.

4 Proposed Experiments and Metrics

In this section, we discuss our proposed methodologies for identifying the energy efficiency of rate limiting algorithms under various traffic patterns. We will develop a basic server with an API that uses these different rate limiting algorithms and a client process to send requests to our server API, emulating different load types. As mentioned above, our primary goal is to find the most energy efficient rate limiting algorithm under different load types. We then use this design to answer the Research Questions in Section 5.

4.1 Experiment Setup

To conduct our experiments, as illustrated in Figure 1, we utilized the following hardware and software components:

- **Raspberry Pi 3:** Used as the server hardware to process incoming requests and enforce rate limiting algorithms.
- **Client Machine:** Configured to simulate traffic by sending requests to the server at predefined load levels.
- **UM25C Meter Tester:** Provided voltage, current, and power measurements at 1-second intervals to capture the server's energy usage during each trial.

Figure 1: Experimental Setup with the Raspberry Pi server, client machine, and UM25C Meter Tester.

4.2 Experiment Parameters

For each traffic pattern (low, medium, high), we will define the following parameters:

- **Request Rate:** (Measured in Requests per Second) The requests per second our client process sends to our server. This is determined by the given traffic pattern.
- **Rate Limit:** (Measured in Requests per Second) The maximum number of requests our server can handle per second. Based on our server's measured capacity under different request conditions, we will define the rate limit, and implement it using the algorithms being studied.

- **Duration:** For each traffic pattern and rate limiting algorithm, we will conduct 45 trials, each lasting 60 seconds, to ensure consistent and reliable results.

4.3 Experiment Methodology

- **Traffic Patterns Choices:** Testing the impact of different traffic patterns on server energy efficiency is inspired by findings from [4], which highlights significant variations in energy consumption based on server workload characteristics. We will test the following constant traffic patterns in our experiment:
  - **Low Traffic:** Traffic flows at a steady and consistent rate, 50% below the rate limit.
  - **Medium Traffic:** Traffic flows at 100% of the rate limit.
  - **High Traffic:** Traffic flows at 150% of the rate limit.

  For our preliminary results, we focused on constant rates. Non-constant rates (e.g., steadily increasing, burst traffic) will be considered in future iterations of this study.
- **Algorithm Choices:** We will be testing the following: no limiting algorithm, token bucket, leaky bucket, fixed window, sliding window. Aligning with the research conducted by [7], our experiment is designed to identify rate limit algorithms which optimize energy without compromising the total requests handled by our server, hence contributing to greener server solutions.
- **Procedure:**
  1. **Server Configuration:** The server is initialized with a specified rate limiting algorithm, token count, and the corresponding algorithm parameters.
  2. **Client Initialization:** The client machine is configured to send requests to the server under three predefined load levels: low, medium, and high.
  3. **Trial Details:** A total of 45 trials are conducted across the 5 rate limiting algorithms tested, with each trial lasting 60 seconds. Each algorithm is tested at low, medium, and high load rates, with three repetitions for each combination.
  4. **Energy and Requests Measurement:** Using a UM25C Meter Tester, we measure the server's power consumption (with the baseline power subtracted) at 1-second intervals during each trial. The total number of requests processed during each trial is recorded.
  5. **Efficiency Calculation:** We calculate the efficiency of each algorithm using the formula:

$$\text{Efficiency} = \frac{\text{Total Requests Processed}}{\text{Total Energy Used}} \tag{1}$$

  6. **Statistical Analysis:** To compare the efficiency metrics across different algorithms and load levels, we will perform statistical tests, including the One-Way Analysis of Variance (ANOVA) test.

5 Preliminary Results

In this section, we answer four research questions based on our experimental design to identify rate limiting algorithms that maximize efficiency for different load sizes:

- RQ1: Which rate limiting algorithms maximize efficiency (total requests served over energy) under a given load?
- RQ2: How does power usage of a server scale (linear, logarithmic, exponential) as load size increases?
- RQ3: How does efficiency of a server scale (linear, logarithmic, exponential) as load size increases?
- RQ4: How does RAM/CPU usage of different algorithms correlate with power consumption of a server?

For our preliminary results, we will be focusing on RQ1, RQ2, and RQ3.

5.1 RQ1: Efficiency Maximization

The first research question aimed to identify whether there were any rate limiting algorithms with statistically significant differences in efficiency when compared to the performance of other algorithms, under each load type. Initial results were collected for low, base, and high constant loads.

Algorithms	Load	Efficiency Mean
No RL	Low	47.74808407
	Med	82.14607015
	High	111.0150545
Token Bucket	Low	47.83896547
	Med	71.19457613
	High	59.51413674
Leaky Bucket	Low	47.18760432
	Med	66.07955214
	High	61.09176376
Fixed Window	Low	47.14954696
	Med	80.52816927
	High	79.08096584
Sliding Window	Low	47.00786907
	Med	79.15943734
	High	79.78074536

Table 1: Efficiency Mean of Algorithms under Different Loads

The trials carried out without rate limit had the highest efficiency ratings because all requests were fulfilled. Because of testing limitations, the server was not overloaded by the maximum request rate. Further studies could use a request rate large enough to overload the server, which is more representative of real-world scenarios.

Under low constant loads, the request rate was always below the set rate limit for all of the algorithms. There were no statistically significant differences between rate limiting algorithm efficiencies. The P-Value of the ANOVA test was 0.4563, much higher than the 0.05 threshold set for significance.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1.723606867	3	0.430901717	0.989552498	0.456337215	3.478049691
Within Groups	4.354510927	8	0.435451093			
Total	6.078117794	11				

Table 2: ANOVA Results Under Low Constant Loads

In these conditions, the algorithms never had to reject any requests; all requests were fulfilled. Because energy usage was relatively similar for all of the algorithms, and the request response ratio was 1, the efficiencies were also similar for all algorithms.

Under medium constant loads, the request rate was set to be equal to the set rate limit. The results indicated that there was a significant statistical difference between the efficiencies of the algorithms. The P value was 4.03301E-05, much lower than the 0.05 threshold.

In these conditions, the algorithms accepted most requests, but some were still rejected. Because each algorithm implemented the set rate limit in a different way, certain cut-offs inherent to the way the algorithm worked could cause some requests to be rejected. The sliding window and fixed window algorithms in particular had noticeably higher request acceptance rates than the token bucket and leaky bucket algorithms.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	418.8283549	3	139.6094516	38.97034291	4.03301E-05	4.066180551
Within Groups	28.65963011	8	3.582453764			
Total	447.487985	11				

Table 3: ANOVA Results Under Medium Constant Loads

The energy usage, though higher overall for all algorithms, was relatively similar between algorithms. Due to these factors, the window algorithms had a significantly higher overall efficiency.

Under high constant loads, the request rate was set above the rate limit. The results indicated that there was a significant statistical difference between the efficiencies of the algorithms. The P value was 0.000252379, much lower than the 0.05 threshold.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1102.098187	3	367.3660622	23.55251621	0.000252379	4.066180551
Within Groups	124.7819329	8	15.59774161			
Total	1226.880119	11				

Table 4: ANOVA Results Under High Constant Loads

The high constant load results followed a similar pattern to the medium constant load trials. The window algorithms had higher efficiencies due to the details mentioned earlier.

5.2 RQ2: Power Scaling

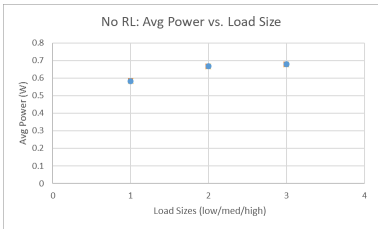

Figure 2: No RL

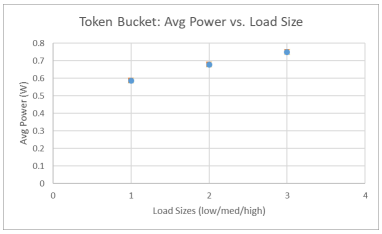

Figure 3: Token Bucket

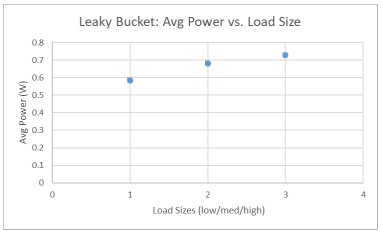

Figure 4: Leaky Bucket

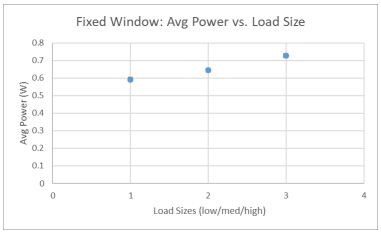

Figure 5: Fixed Window

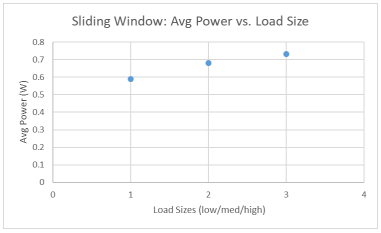

Figure 6: Sliding Window

The second research questions aimed to identify how the power usage of a server scaled with load size. Again, low, medium, and high constant loads were measured and their powers were plotted. Due to the time limitations of the study, only 3 levels were used (high, medium, and low), so the relationships could not fully be identified with preliminary results. However, the graphs provide an initial estimate for how power may scale.

All of the graphs seem to follow a relatively linear pattern, with a steady increase in average power as load size increases. The trend varies among the graphs, however, with some graphs having a more or less steep

increase. It is possible that the curves would actually follow logarithmic, polynomial, or exponential curves with more data points from more load sizes tested. With the limited data from the initial proposal, it is not possible to directly determine function by which the power consumption scales with load size.

5.3 RQ3: Efficiency Scaling

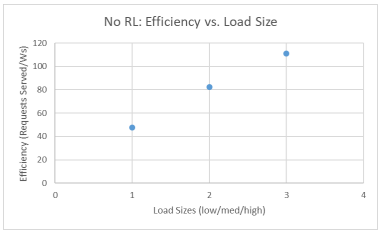

Figure 7: No RL

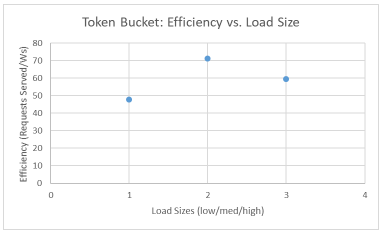

Figure 8: Token Bucket

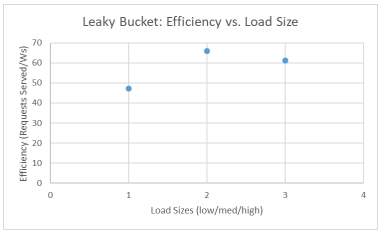

Figure 9: Leaky Bucket

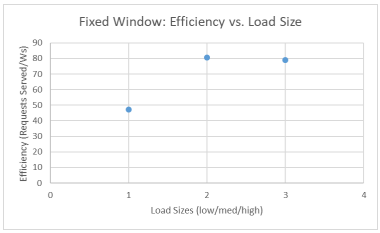

Figure 10: Fixed Window

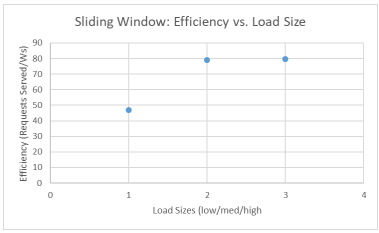

Figure 11: Sliding Window

The third research questions aimed to identify how the efficiency of a server scaled with load size. Just as with RQ2, only 3 levels were used and plotted (high, medium, and low). Unlike the average power measurements, efficiency considers how many requests each algorithm was able to fulfill given its energy usage.

Unlike the average power graphs, efficiency peaks with medium load sizes for almost all of the rate limiting algorithms. The graph representing the trials with no rate limiting algorithms had a constantly increasing efficiency, due to the fact that it was able to respond to all requests with relatively minor increases in energy usage. The sliding window algorithm had an almost equal efficiency under high and medium loads, likely due to its strength at handling steady constant loads like the experiments done so far. Again, the data for the preliminary study was limited, so further patterns may emerge from testing more load sizes/types.

6 Conclusion and Future Work

This study reinforces the importance of understanding the energy efficiency of rate-limiting algorithms under varying load conditions. While our experiments do not confirm that any single algorithm consistently outperforms others across all load types, the findings underscore the potential for optimization tailored to specific traffic patterns.

Future work will expand upon these results by incorporating burst load scenarios, which are common in real-world applications such as e-commerce and media streaming. Additionally, testing on industry-standard servers will provide more generalizable insights, moving beyond the constraints of Raspberry Pi hardware. Another promising avenue for exploration is the implementation of dynamic rate-limiting algorithm switching. By adapting algorithms in real-time to match load conditions—switching, for example, to a more efficient algorithm during low traffic and a more robust one during bursts—systems can further optimize energy use and throughput. These advancements can pave the way for increased operational efficiency and sustainability in large-scale web services.

References

[1] Bays, L. R., and Marcon, D. S. Flow based load balancing: Optimizing web servers resource utilization. *Journal of Applied Computing Research 1* (02 2012).

[2] Elbably, M. Rate limiting: The sliding window algorithm, 2023. Accessed: 20-10-2024.

[3] Georgia Tech. Major causes of source-level bursts. https://sites.cc.gatech.edu. Accessed: 2024-10-20.

[4] Manotas, I., Sahin, C., Clause, J., Pollock, L., and Winbladh, K. Investigating the impacts of web servers on web application energy usage. *2013 2nd International Workshop on Green and Sustainable Software (GREENS) 2* (05 2013).

[5] Medhi, D., and Ramasamy, K. *Network Routing: Algorithms, Protocols, and Architectures*, 2nd ed. Elsevier Science, 09 2017.

[6] Nlyte. How Much Does it Cost to Power One Rack in a Data Center? https://www.nlyte.com/blog/how-much-does-it-cost-to-power-one-rack-in-a-data-center/. Accessed: 2024-10-20.

[7] Varghese, B., Carlsson, N., Jourjon, G., Mahanti, A., and Shenoy, P. Greening web servers: a case for ultra low-power web servers. *International Green Computing Conference* (11 2014).