Selection of A Suitable Algorithm for the Implementation of Rate-Limiter Based on Bucket4j

https://doi.org/10.3991/ijoe.v18i04.25641

Maxim Bartkov[1](✉), Dmitry Borovikov[2]
[1] RooX Solutions Java Team Lead, Khakov, Ukraine
[2] Oceanic and Space Sciences and Scientific Computing, University of Michigan, New York, USA
accessibility@umich.edu

Abstract—In shared network services, rate limiting is essential as it controls the requests of the users or requesters in a specific amount of time. Due to rate limiting, the service or API stays protected from overuse, malicious attack, DDoS attack, data traffic spikes, etc. Bucket4j is a java library that has been demonstrated to be effective in rate limiting. While Bucket4j is mainly based on token bucket algorithm, rate limiting processes can be based on various effective algorithms. Selecting the most suitable algorithm for rate limiting is an essential problem. To address it, we have done a detailed analysis of rate-limiting algorithms based on various factors. The factors we have considered are easy implementation, proper handling of data traffic, data starvation, memory usage, etc. We have found out that for different set of requirements, different algorithms are preferable.

Keywords—rate-limiting, Bucket4j, token-bucket algorithm, leaky-bucket algorithm, fixed window algorithm, sliding window algorithm

1 Introduction

Controlling traffic is a very important problem in shared network services and APIs. Large traffic can be intended or unintended by users, but it's essential to keep services stable and available. A rate limiter solves this problem by limiting access to services, APIs, etc. In an API, a rate limiter restricts the number of requests a client or a user can make within a specific time. Thus the service stays protected from malicious and unintentional overuses [1]. Rate limiters use various techniques such as bandwidth control modules, to restrict attack traffics at source ends [2]. Also this technology controls the traffic rate for HTTP. A network has a limit considering its energy consumption, and a key target is to reduce it [3]. Rate limiters can control energy consumption. Allocating an optimal maximum rate is necessary, and for instance, it can be done for packet communications in wireless networks [4]. GitHub restricts authenticated API requests to 5000 per hour and 60 unauthenticated requests per hour. In IoT applications, there has been a wide range of research on reducing data rates [5] which indicates the importance

of rate limiting. Bucket4j is a library in java for rate limiting. This library mainly follows the token bucket algorithm. It's a thread-safe library and is effective for both clustered environments and standalone JVM applications. Distributed or in-memory caching via JSR107 or JCache specification is also available in Bucket4j. In a clustered environment, controlling data traffic, energy efficiency, energy consumption, etc., are the main challenges [6]. Bucket4j provides a solution in these cases. Rate limiters are based on various rate-limiting algorithms. Different rate limiters follow different algorithms based on their purposes. There are various requirements to meet in rate limiting processes. Based on different requirements, it's important to select a suitable algorithm for the rate limiter. As an example, in wireless sensor networks, by using algorithms like the PSO-BP algorithm, it's simpler to solve problems, such as high energy consumption, low efficiency, etc. [7]. In the present paper, we analyze several rate limiting algorithms. We indicate strengths and weaknesses of algorithms as a guideline for selecting a suitable algorithm based on a developer's requirements and/or limitations.

The next sections provide a detailed discussion and analysis of rate limiting and its algorithms. Section 2 discusses the background of rate limiters and rate limiting algorithms. Section 3 provides a list of related works and applications of rate limiters and algorithms. Section 4 discusses the factors based on which suitable algorithms can be selected for rate limiting. Section 5 provides the overall analysis. Lastly, section 6 concludes the analysis.

2 Background

Implementation of rate limiting is based on various algorithms. The most widely used algorithms are the token bucket algorithm, leaky bucket algorithm, fixed window algorithm, sliding window algorithm, etc. Each of these algorithms is best suited for specific factors and conditions.

2.1 Token bucket algorithm

Token bucket algorithm provides solutions for traffic shaping in packet-switched networks [8]. In token bucket algorithm, when there's a request for an API endpoint access, the bucket will give a token depending on availability to the requester. If tokens are available and the service accepts the request, then the system removes a token from the bucket. If tokens aren't available, the system rejects the request. As tokens are decreasing with each acceptance of the request, the system replenishes the tokens at a fixed rate. Thus the system maintains the bucket's capacity.

Figure 1 shows the process flow of the token bucket algorithm. Upon receiving a request from a user, the system checks if the bucket holds enough tokens for the incoming data packets. The size of data packets, S, to be equal to or less than the size of available tokens, T. So, if $S \leq T$, the system accepts the request, and the packet conforms. Also, the number of available tokens decreases. If $S > T$, the system rejects the request, and the packet becomes nonconforming.

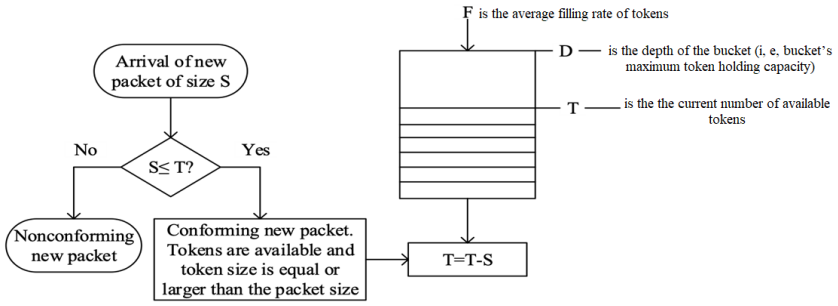

Fig. 1. Process flow of token bucket algorithm

Advantages

1. Token bucket algorithm is very effective in controlling traffic bursts. Requests aren't dropped or leaked as the bucket doesn't allow requests or data packets without the availability of tokens.
2. When request is accepted, there's a guarantee that it will be processed.

Disadvantages

1. Due to handling data traffic by using tokens, its implementation is not so simple as compared to some other methods.

2.2 Leaky bucket algorithm

The leaky bucket algorithm is simple to implement. Here, the system holds the requests using a queue or a bucket. Upon receiving a new request, the system sends it to the queue as long as the queue has a vacancy. Otherwise the request is leaked or dropped and the system notifies requesting user. The system processes the requests at fixed time intervals. It executes the requests in the first-come-first-serve manner. The conventional leaky bucket algorithm mainly needs 2 parameters: bucket size and leaky rate [9].

Figure 2 shows the operation of leaky bucket algorithm. Suppose, for a system, the bucket has a capacity of 3 requests per minute. As shown in Figure 2, the system accepts the first 3 requests but rejects the next 2 requests as the capacity of 3 requests per minute is exceeded.

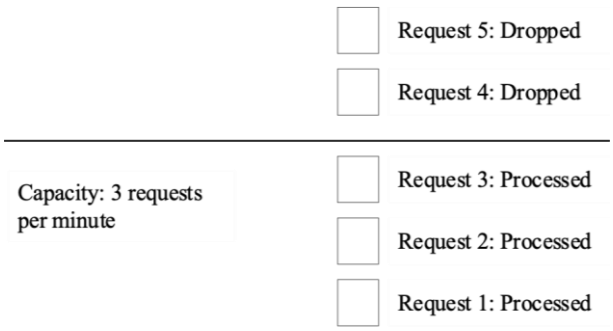

Fig. 2. Working principle of leaky bucket algorithm

Advantages

1. (VVV) Simplicity of implementation.
2. It keeps the bursts of requests smooth by executing them at a constant rate.
3. It's memory efficient as the queue size is constant.

Disadvantages

1. New requests may starve as the queue may fill-up with the traffic of old requests.
2. There's uncertainty whether accepted requests will be processed in a certain amount of time.

2.3 Fixed window algorithm

Fixed window algorithm divides processing timeline into fixed windows. The windows have a fixed time length, such as 1 hour, 1 minute, etc. A counter variable counts the number of requests in the window. If the counter exceeds the predefined limit, all incoming requests before the end of the window are dropped. After every window, the system resets the counter [10]. Figure 3 shows how the fixed window algorithm works. Suppose a service has a limit of 20 requests per hour. As seen in the figure, a new request arrives at 1:55 PM as the window (1:00 PM to 2:00 PM) has 15 requests. Since the limit is 20, the request is accepted. Again at 2:58 PM, another request arrives, but that window (2:00 PM to 3:00 PM) has already reached the limit of 20 requests. Therefore, the request gets rejected.

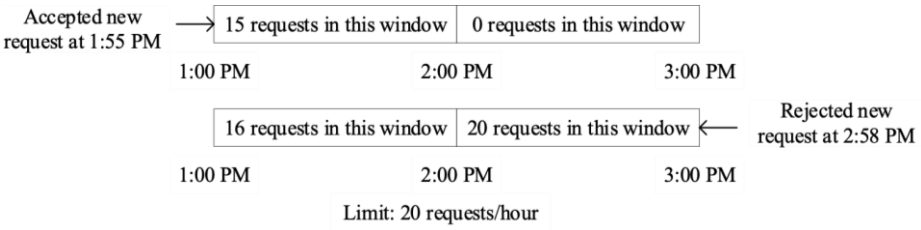

Fig. 3. Working principle of fixed window algorithm

Advantages

1. Simplicity of implementation.
2. Low memory load: the system only needs to store the request count.
3. More new requests are executed as the system resets the counter at the end of each window.

Disadvantages

1. On both sides of a boundary of two windows, a single traffic burst can result in more requests than the limit of requests per hour. Suppose, at 1:59 PM, there are 12 new requests, and at 2:02 PM, there are 11 new requests. So, there are 23 new requests in a few minutes, which is more than the request limit, which is 20 per hour. But as all of these requests aren't in the same window, these requests get accepted.
2. During peak hours, consumers may exploit counter variable reset thus causing a problem in the server.

2.4 Sliding window algorithm

The sliding window algorithm has similarities with the fixed window algorithm. But this algorithm provides solutions to some of the limitations of the fixed window algorithm. Here, the counter doesn't get reset after each window, but it uses the previous window's information and estimates the current window's number of allowable requests. So, in the sliding window algorithm, the windows can smooth traffic bursts much better than the fixed windows.

As shown in Figure 4, 80 requests were accepted in the first window (1:00 PM to 2:00 PM), and 50 requests have been accepted in the first 30 minutes of the second window (2:00 PM to 3:00 PM). Then, there's a request at 2:30 PM. Now, the counter will make the decision whether to accept an incoming request based on the information from the previous window and the current window. The request acceptance is 100 requests per hour. From 2:30 PM, the counter goes behind 1 hour and considers a 1-hour window from 1:30 PM to 2:30 PM.

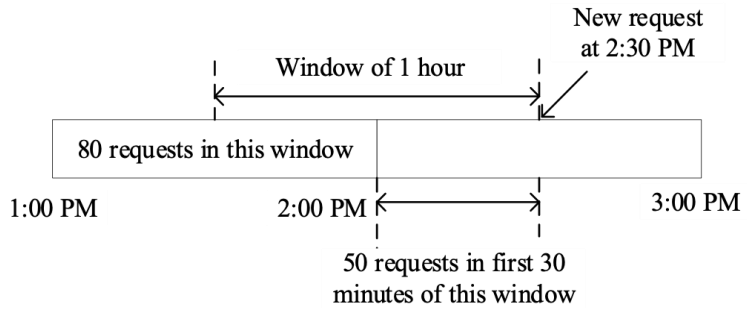

Fig. 4. Working principle of sliding window algorithm

Requests accepted in the last 30 minutes of the first window = 80 * (30/60) = 40.

Requests accepted in the first 30 minutes of the second window = 50.

Total requests accepted in the 1-hour window from 1:30 PM to 2:30 PM = 40+50 = 90.

As total requests accepted = 90 < 100, so the request at 1:30 PM will be accepted.

Advantages

1. It handles the request spikes better than the fixed window algorithm.
2. Its request handling process is very accurate. The number of wrongly allowed requests is very low.

Disadvantages

1. Memory footprint is high as the system has to maintain all the request timestamps for an entire window.
2. Time complexity is high as the system has to remove older timestamps [11].

2.5 Rate limiting using Bucket4j

Rate limiting processes involve tracking API addresses, using API access tokens or keys, etc. When a client reaches the limit, the rate limiter queues the request, or rejects the request, or allows the request with an extra charge. Bucket4j library in java works very effectively in rate limiting processes.

Bucket4j works by following the token bucket algorithm. It starts with a Maven configuration. Also, it offers various features in rate limiting. It's a thread-safe library. It's mainly a cluster environment or independent JVM application [12].

Features of Bucket4j. The implementation of Bucket4j is lock-free and effective [13]. It ensures completely non-compromise precision. It performs the calculations in integer arithmetic rather than with doubles or floats. Thus end-users stay protected from adverse effects of rounding errors.

Bucket4j needs only 2 lines of code to move from JVM to cluster. It can limit something in JVMs' cluster. Any GRID solution compatible with JSR 107 or JCache API is available in Bucket4j.

Multiple bandwidth specification per bucket is another feature of Bucket4j. It supports both synchronous API and asynchronous API. Buckets can also work as a scheduler with Bucket4j.

Terminologies of Bucket4j. There are some terminologies related to Bucket4j. The 'Bucket' interface denotes the bucket with the highest capacity of tokens. For using tokens, there are methods, such as 'try Consume', 'try Consume And Return Remaining'. When the system uses the token by letting the request conform to the limits, these methods or commands return the consumption result as 'true'. A bucket's key building block is the 'Bandwidth' class. It defines the bucket's limits. Bandwidth is necessary to configure the bucket's capacity and refill rate. The 'Refill' class helps to define the fixed filling rate of tokens onto the bucket. 'Consumption Probe' is another class containing the consumption result, the bucket's status, such as remaining tokens, remaining time until the availability of the requested tokens into the bucket again, etc.

Basic operations of Bucket4j. Rate limiting has some basic patterns. When the rate limit is 15 requests per minute, the bucket's capacity will be 15, and the token refill rate will be 15 per minute. Figure 5 shows the code of setting the refilling of tokens. The command 'Refill. intervally' refills tokens into the bucket at the time of window's beginning. As for this example, 15 tokens are injected at the start of the window.

```
Refill ref = Refill.intervally(15, Duration.ofMinutes(1));
Bandwidth lim = Bandwidth.classic(15, ref);
Bucket buck = Bucket4j.builder().addLimit(lim).build();
for (int j = 1; j <= 15; j++) {
   assertTrue(buck.tryConsume(1));
}
assertFalse(buck.tryConsume(1));
```

Fig. 5. Code for refilling rate of tokens

To avoid spikes that can exhaust the tokens, Bucket4j is effective. It can set many limits on the bucket. Suppose, in the first 7 seconds, a spike can exhaust all the tokens. Figure 6 shows the code that allows 7 requests in a time window of 25 seconds.

```
Bucket buck = Bucket4j.builder().addLimit(Bandwidth.classic(15, Refill.intervally
   (15, Duration.ofMinutes(1)))).addLimit(Bandwidth.classic(7, Refill.intervally(7,
   Duration.ofSeconds(25)))).build();

for (int j = 1; j <= 7; j++) {
   assertTrue(buck.tryConsume(1));
}
assertFalse(buck.tryConsume(1));
```

Fig. 6. Code of Bucket4j to avoid spikes

3 Related works and applications

Bucket4j has a wide range of applications as a rate-limiting tool. By using Bucket4j, it's possible to rate limit Spring MVC endpoints. Bucket4j has also been successfully used together with In Memory Data Grid Hazlecast [14]. Many applications implement rate limiting using different rate limiting algorithms. Researchers have modified the token bucket algorithm into a bi-direction adjustable algorithm to control data traffic in network systems in the enterprise and residential sector. They have ensured a guarantee of multi-class bandwidth and sharing [15]. In the Tor project, improving the Tor network's performance has been a challenge. The token bucket algorithm has been successful in ensuring that packets aren't lost, when users send data packets in the form of requests, and the bandwidth limit set by the system is respected [16]. The algorithm works more effectively with optimal values for refill intervals of tokens for the Tor

network's different scenarios. By using the multi-rate token bucket concept, it is possible to explain mathematical models of managing traffic flows in networks. It's also possible to explain the adaptive formation of network traffic flow tuning of control systems with the shaper's structure and control parameters with indirect feedback. Multiple token bucket shapers have shaped traffic streams in the network [17]. The leaky bucket algorithm is also effective in controlling network traffic. Due to congestion, major threats occur in the communication process between nodes in wireless sensor networks. It results in overflowing of the buffer, delaying packets, low energy efficiency, minimization of the throughput, etc. Implementation of the leaky bucket algorithm has made it possible to detect and avoid congestion effectively [18]. The leaky bucket algorithm controls the bucket size, the number of allowable packets, the flow rate to avoid congestion. Researchers have used the sliding window concept in the process of detecting network intrusion in cloud computing platforms [19]. There are major threats while conducting complete real-time active monitoring, defense, and detection. In these processes, the sliding window concept is effective in achieving desired results.

4 Selecting suitable algorithms for rate limiting with Bucket4j

It's a great challenge for tech giants to provide access to their shared services using API properly. There have been several methods for controlling the number of requests from users. Having discussed some of the methods in rate limiting in previous sections, in this section we discuss the factors based on which engineers can select the most suitable rate limiting algorithm. There are many factors to be considered when choosing a rate limiting algorithm. The factors include: easy implementation, handling traffic bursts, memory usage, dropping or leakage of requests, energy consumption, efficiency, etc. Different service providers have different priorities, and thus may prefer different rate limiting algorithms.

4.1 Easy implementation

When service providers look for a simple rate limiting mechanism, leaky bucket and fixed window algorithms are suitable to implement. These two algorithms are the simplest for implementing rate limiting. In the case of the leaky bucket algorithm, whenever a request arrives, the system appends it to the queue. Regardless of what happens next, it's a simpler way to handle data traffic. The fixed window is also an as y to implement method. As long as the counter doesn't reach the limit of requests, the fixed window will keep processing the requests. Though these two algorithms have other limitations, they are preferable regarding ease of use.

4.2 Handling traffic bursts

In a shared network service, there are continuous requests from the users. It's very important to specify how the service providers handle the traffic of requests. There's a wide range of applications and methods for controlling traffic bursts. Bucket4j has its

applications of refilling process and controlling traffic bursts, as discussed earlier. Token bucket algorithm has been a very effective tool for controlling traffic bursts. When there are requests from users, the data packets enter the bucket only when there are enough tokens for the packets. The tokens provide a guarantee for processing the packets if allowed into the bucket. So, there's no possibility of dropping or leakage of packets from the bucket, unlike the leaky bucket algorithm and fixed window algorithm. Bucket4j also works based on the token bucket algorithm; hence it performs so effectively in rate limiting. While implementing the token bucket algorithm in handling traffic bursts, we can consider the case as shown in Table 1. Two users' token bucket is a process to handle traffic bursts in the network. Suppose the working speed of a network output interface is 5 Mb/s. At some time, there are 2 requests where user 1 is in high priority class with 3Mb/s of minimum guaranteed bandwidth and user 2 is in low priority class with 2Mb/s of minimum guaranteed bandwidth. In the case 1, traffic arrived from both users is more than the minimum guaranteed bandwidth. So, each of the users only get respective minimum guaranteed bandwidth. In the case 2, user 1 gets more than minimum guaranteed bandwidth as user 2 needed smaller bandwidth. In the case 4, the total bandwidth used by both users was less than the speed of the network. In this way, the traffic of data packets can be handled.

Table 1. Traffic management of shared networks

Case No.	**Arrived Traffic**		**Passed Traffic**	
	User 1	User 2	User 1	User 2
1	4 Mb/s	3 Mb/s	3 Mb/s	2 Mb/s
2	5 Mb/s	1 Mb/s	4 Mb/s	1 Mb/s
3	3 Mb/s	3 Mb/s	3 Mb/s	2 Mb/s
4	3 Mb/s	1 Mb/s	3 Mb/s	1 Mb/s
5	2 Mb/s	3 Mb/s	2 Mb/s	3 Mb/s

The sliding window algorithm is also effective in handling traffic bursts. As seen in the case of the fixed window algorithm, there can be data traffic bursts in the timeline of a window if there are a huge number of requests on both sides of the boundary of two windows. When such an incident occurs, the number of requests exceeds the limit [21]. The sliding window algorithm avoids such incidents as it keeps the records of the previous window [20]. So, when a request arrives, the system checks the timeline of the sliding window and makes the decision based on this information.

4.3 Memory usage

Efficient memory usage is another important factor while conducting rate limiting. When data traffic is large, it's also important to ensure efficient memory usage. The fixed window algorithm requires less memory as it doesn't need to handle additional tasks like sliding window does, which has to keep the records of the previous window. Similarly, the leaky bucket algorithm also requires less memory as it's mainly based on handling the queue of requests and processing the requests in first-come-first-serve

manner. So, despite having other limitations, leaky bucket and fixed window algorithms are preferred, when available memory is limited.

4.4 Dropping and leakage of requests

An unexpected incident in a shared network service is dropping or leakage of requests. In leaky bucket and fixed window algorithms, the possibility of requests being dropped is higher. As seen in the leaky bucket mechanism, if bucket capacity is exceeded, requests in the queue get rejected. Such incidents can be avoided using the token bucket and sliding window algorithm. The token bucket mechanism will accept requests data packets into the bucket based on available tokens. So, the system will surely process the data packets from the bucket. So, to avoid dropping or leakage of requests, the token bucket algorithm and sliding window algorithm are more effective.

5 Discussions

Based on the factors discussed in the previous section, selecting a suitable rate limiting algorithm depends on the priority of the service providers. Some service providers prefer easy and economic implementation, and some prefer proper traffic burst handling and high efficiency. We performed analysis of rate limiting algorithms like the token bucket algorithm, leaky bucket algorithm, fixed window algorithm, sliding window algorithm to identify, which are preferred under different conditions. For easy implementation and economic applications, the leaky bucket algorithm and the fixed window algorithm have been found to be efficient. At the same time, these two algorithms may not be the optimal choice, when high efficiency is required. Token bucket algorithm and sliding window algorithm have been effective in handling traffic bursts in networks. These two algorithms have also performed with high efficiency.

Each of the algorithms offers some advantages as well as some limitations. However, it's possible to modify the algorithm; thus, the limitations of the algorithms can be reduced to some extent. So, to improve rate limiting, apart from just selecting the most suitable algorithm, modifying the algorithms may be necessary to overcome some limitations.

6 Conclusion

Rate limiting is an essential task and also a defensive measure for shared network services. Proper rate limiting protects the network from excessive use along with handling the traffic burst efficiently. Researchers are continuously working with rate limiting algorithms to increase the efficiency of rate limiting. The existing algorithms have made it easy for us to handle data traffic bursts in networks efficiently. Also, it has been possible to ensure the simple and easy implementation of rate limiters. While conducting rate limiting, properly considering the discussed factors can make rate limiting more efficient.

7 References

[1] P. Srivastava, "Rate Limiting a Spring API Using Bucket4j," *Baeldung*, 09-Sep-2020. [Online]. Available: https://www.baeldung.com/spring-bucket4j [Accessed: 21-Jul-2021].

[2] R. Y. Patil and L. Ragha, "A dynamic rate limiting mechanism for flooding based distributed denial of service attack," in Fourth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom2012), 2012. https://doi.org/10.1049/cp.2012.2512

[3] S. Ling and Q. D. Yang, "Secure and low energy consumption range query in tiered sensor networks," Int. J. Online Eng., vol. 12, no. 07, p. 4, 2016. https://doi.org/10.3991/ijoe.v12i07.5510

[4] S. Ling and Q. D. Yang, "Secure and low energy consumption range query in tiered sensor networks," Int. J. Online Eng., vol. 12, no. 07, p. 4, 2016. https://doi.org/10.3991/ijoe.v12i07.5510

[5] D. Gao, J. Zhang, F. Zhang, and H. Lin, "Distributed optimal maximum rate allocation based on data aggregation in Rechargeable Wireless Sensor Networks," Int. J. Online Eng., vol. 14, no. 03, p. 172, 2018. https://doi.org/10.3991/ijoe.v14i03.8339

[6] O. H. Yahya, H. Alrikabi, and I. A. Aljazaery, "Reducing the data rate in internet of Things applications by using Wireless Sensor Network," Int. J. Onl. Eng., vol. 16, no. 03, p. 107, 2020. https://doi.org/10.3991/ijoe.v16i03.13021

[7] X. Yang, "Data clustering method in wireless sensor networks based on residual energy perception," Int. J. Online Eng., vol. 14, no. 06, p. 85, 2018. https://doi.org/10.3991/ijoe.v14i06.8700

[8] S. Sahana and R. Amutha, "Data aggregation in wireless sensor networks," in International Conference on Information Communication and Embedded Systems (ICICES2014), 2014, pp. 1–6. https://doi.org/10.1109/ICICES.2014.7034189

[9] F. Gao and H. Qian, "Efficient, real-world token bucket configuration for residential gateways," IEEE ACM Trans. Netw., vol. 24, no. 1, pp. 462–475, 2016. https://doi.org/10.1109/TNET.2014.2366496

[10] S. Pinaikul and W. Benjapolakul, "Credit token leaky bucket algorithm with fuzzy logic in ATM networks," in Proceedings. Ninth IEEE International Conference on Networks, ICON 2001, 2005, pp. 296–301.

[11] "Design a scalable API rate limiting algorithm - system design," Nlogn.in, 01-May-2020. [Online]. Available: https://nlogn.in/design-a-scalable-rate-limiting-algorithm-system-design/ [Accessed: 21-Jul-2021].

[12] A, "Rate Limiting Algorithm," Wordpress.com, 19-Oct-2019. [Online]. Available: https://preparingforcodinginterview.wordpress.com/2019/10/19/rate-limiting-algorithm-algo/ [Accessed: 21-Jul-2021].

[13] "Using bucket4j to limit the access rate of spring API - baeldung - Java知识," Javamana.com. [Online]. Available: https://javamana.com/2021/04/20210407215435749h.html [Accessed: 21-Jul-2021].

[14] V. Bukhtoyarov, bucket4j: Java rate limiting library based on token/leaky-bucket algorithm.

[15] A. Mohamed, spring-boot-bucket4j-hazelcast-demo.

[16] D. He, W. Zhou, and X. Zhang, "A bi-direction adjustable token bucket mechanism for multi-class bandwidth guarantee and sharing," in 2009 IEEE International Conference on Network Infrastructure and Digital Content, 2009. https://doi.org/10.1109/ICNIDC.2009.5360808

[17] Kiran, A. Rathore, Vignesh, P. D. Shenoy, Venugopal, and V. T. Prabhu, "Optimal Token Bucket Refilling for Tor network," in 2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2018.
[18] Y. Toroshanko, Y. Selepyna, N. Yakymchuk, and V. Cherevyk, "Control of traffic streams with the multi-rate token bucket," in 2019 3rd International Conference on Advanced Information and Communications Technologies (AICT), 2019, pp. 352–355. https://doi.org/10.1109/AIACT.2019.8847860
[19] Srinivas, Gowtham, S. Amith, Chaitanya, Archana, and G. Raja, "Leaky Bucket based congestion control in Wireless Sensor Networks," in 2018 Tenth International Conference on Advanced Computing (ICoAC), 2018, pp. 172–174.
[20] H. X. Ni, "Based sliding window cloud computing platform of network intrusion detection algorithm," in 2014 IEEE International Conference on Computer and Information Technology, 2014, pp. 927–930.
[21] "An alternative approach to rate limiting," Figma.com. [Online]. Available: https://www.figma.com/blog/an-alternative-approach-to-rate-limiting/ [Accessed: 21-Jul-2021].

8 Authors

Bartkov Maxim Vitalievich, RooX Solutions Java Team Lead prospect Petra Grigorenko 16, Khakov, Ukraine, 6100 (email: accessibility@umich.edu, ORCID: https://orcid.org/0000-0003-3527-3642).

Borovikov Dmitry, Ph.D. in Atmospheric, Oceanic and Space Sciences and Scientific Computing, University of Michigan Instabase, Inc. 1480 York ave apt 6F, New York, NY 10075 (email: accessibility@umich.edu, ORCID: https://orcid.org/0000-0002-0151-7437).

Article submitted 2021-07-21. Resubmitted 2021-10-08. Final acceptance 2021-10-09. Final version published as submitted by the authors.