Throttling and Rate Limiting in System Design

What is Rate Limiting?

Rate limiting is a technique used to control the amount of traffic allowed to access a system or network within a specific time period. It helps us to prevent overuse or abuse of resources by limiting the rate at which events can occur. This can be used to optimize system performance and ensure that resources are distributed fairly among users.

For example, a website might use rate limiting to prevent someone from repeatedly trying to log in to an account with the wrong password. If user attempts to log in too often within a certain time window, website might block their access or slow down their requests. This helps prevent hackers or malicious actors from overburdening the system and causing attacks like Denial of Service (DoS).

In general, rate limiting is implemented as a protective mechanism to restrict the excessive use of shared services and preserve their availability. It is critical for ensuring the scalability and performance of distributed systems by limiting consumption and minimizing end-to-end latency on both the client and server sides. Clients must be developed with rate limitations in mind to ensure that the system functions well and avoids cascade failure.

How rate limiting works?

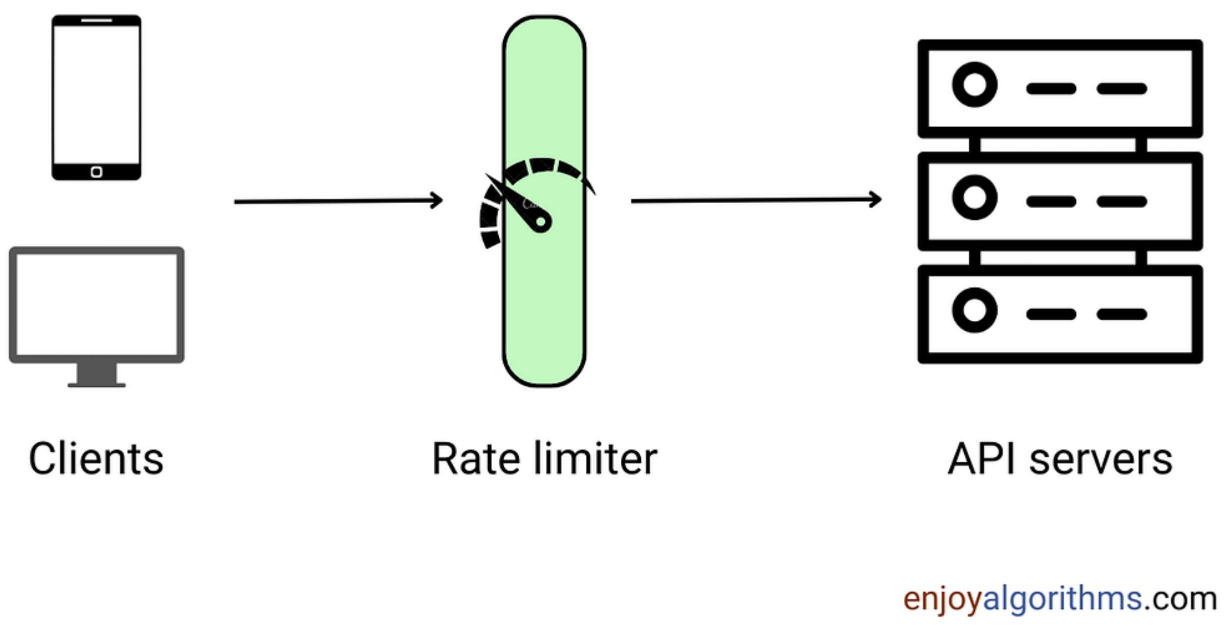
Rate limiting works by tracking IP addresses of incoming requests and time elapsed between them. If a single IP address makes too many requests within a specified timeframe, rate-limiting solution will throttle the IP address and not fulfil its requests for a certain period of time. This way, a rate-limited application acts like a virtual traffic cop, telling unique users who are making requests rapidly to slow down. This is similar to a police officer pulling over a driver for speeding.

What Is API rate limiting?

Granting unlimited access to an API can be risky, as it allows anyone to use the API as much as they want. While it may be helpful for individuals to use the API but open access can also lower its value and limit the growth of the company.

So rate limiting can also protect APIs (Application Programming Interfaces) from being overwhelmed by too many requests. An API that uses rate limiting might throttle or temporarily block any client that tries to make too many API calls. This ensure that legitimate requests will be processed without affecting the overall performance.

To ensure that an API remains scalable and valuable, it is important to implement rate limiting. This involves restricting the number of requests or data that clients can consume. One common unit of measurement for API owners is Transactions Per Second (TPS).



Benefits of using rate-limiting

There are several benefits to using rate limiting to control network traffic.

- **Avoid resource depletion due to a Denial of Service (DoS) attack**: By limiting the number of excess calls, a rate limiter can prevent DoS attacks, whether intentional or unintentional. For example, Twitter restricts users to 300 tweets every three hours, and Google Docs APIs have a default limit of 300 per user every 60 seconds for reading queries.
- **Reduce expenses:** Limiting excess requests can help reduce the number of servers needed and allocate more resources to high-priority APIs. This is important for companies that use paid third-party APIs, which may be charged on a per-call basis.
- **Ensure that servers are not overburdened:** A rate limiter can filter out extra requests produced by bots or user misconduct to reduce server load. This help to ensure that system remains stable and efficient.

Methods to implement API Rate-Limiting

Request Queues

There are many request queue libraries available that make it easy to implement rate limiting in different programming languages and development environments. These libraries often come with pre-written code and can be easily found in library folders.

For example, there are request rate limiter libraries that limit number of requests per second to a specific number and queue any additional requests. This can be a convenient and easy-to-use solution for API development.

Throttling

Throttling is another common method for implementing rate limiting in practice. It involves establishing a temporary state in which each request is evaluated by the API, allowing API developer to maintain control over how their API is used. If throttle is triggered, a user may be disconnected or have their bandwidth reduced.

Throttling can be done at the application, API, or user level. There are many commercial products available that make it easy for developers to implement throttling, such as the Hybrid Data Pipeline from Progress, which provides throttled API access to various databases and services including IBM DB2, Oracle, SQL Server, MySQL, PostgreSQL, SAP Sybase, Hadoop Hive, Salesforce, and Google Analytics.

Rate-limiting Algorithms

Another technique to make scalable rate-limited APIs is to use algorithms. Many rate-limiting techniques are already available, just like request queue libraries and throttling services.

Fixed Window Technique

The fixed window technique is a rate limiting method that uses an incremental counter to **track the number of incoming requests over a fixed time period**. If number of requests exceeds the specified limit during this time period, any additional requests will be discarded. This technique is used to prevent an API from being overwhelmed with outdated requests, but it can still cause the API to become overloaded if there is a sudden surge of requests while the window is refreshing.

In other words, Fixed-window rate limiting algorithms are used to restrict the number of requests allowed during a specific timeframe, also known as **window**. For example, a server might implement an algorithm that allows up to 300 API requests per minute. This means that server will not serve more than 300 requests between 8:00 and 8:01, and the window will reset at 8:01, allowing another 300 requests until 8:02.

Developers can implement fixed-window algorithm at either server or user level. Implementing algorithm at the user level will restrict each user to a certain number of requests per minute, while server-level algorithm will restrict the number of requests made by all users combined.

Leaky Bucket Technique

Leaky bucket technique is a method for rate limiting that is easy to implement and efficient in terms of memory usage. It converts incoming requests into a **First In First Out (FIFO) queue**, allowing it to process items at a consistent rate. The leaky bucket is effective at smoothing out traffic spikes and is simple to set up on a single server or load balancer.

Leaky bucket rate limiting algorithms differ from fixed-window algorithms because they don't rely on specified timeframes. Instead, they focus on the fixed length of request queues, regardless of time. The server will service requests on a first-come, first-served basis, with new requests joining the back of the queue. **If a new request arrives when the queue is full, it will be dropped.**

This approach ensures that the server receives API requests constantly, as requests are forwarded to the server one by one, and there are no sudden bursts of requests. However, it's not perfect. Because the queue is static, there is a higher chance of starvation, where new requests may remain unattended for an extended period of time.

Sliding Window Technique

Sliding-window rate limiting algorithms are time-based and similar to fixed-window algorithms, but they differ in the starting point of each time window. With sliding-window rate limiting, the timeframe starts when a user makes a new request rather than at a predetermined time. For example, if the first request arrives at 7:00:18 am (and the rate limit is 200 per minute), the server will allow up to 200 requests until 7:01:18.

Sliding-window algorithms help solve the issues faced by fixed-window rate limiting, such as starvation, by providing more flexibility. They also mitigate the starvation issue of leaky bucket rate limiting by starting a new time window whenever a request is made.

The sliding window technique is suitable for processing large requests while being lightweight and fast to run, thanks to the limited number of data points needed to assess each request.

Thanks to Navtosh for his contribution in creating the first version of this content. If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!