This Is How Stripe Does Rate Limiting tc Build Scalable APIs

#9: Read Now - Awesome Rate Limiter (4 minutes)

NEO KIM
SEP 28, 2023

37 3 2

This post outlines how Stripe does rate limiting. If you want to learn more, scroll bottom and find the references.

- *Share this post & I'll send you some rewards for the referrals.*

Rate Limiter

A rate limiter is important to building a scalable API. Because it prevents bad use from abusing the API.

A rate limiter keeps a *counter* on the number of requests received. And reject a request if the threshold exceeds. Requests are rate-limited at the user or IP addr level.

And it is a good choice if a change in the pace of the requests *doesn't* affect the experience.

Other potential *reasons* to rate limit are:

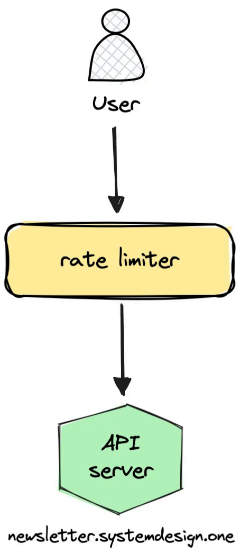
- Prevent low-priority traffic from affecting high-priority traffic
- Prevent service degradation

Rejecting low-priority requests under heavy load is called *load shedding*.

This post outlines how Stripe scales their API with the rate limiter. Consider shar this post with someone who wants to study scalability patterns.

Rate Limiter Workflow

The rules for rate limiting are predefined. And here is the rate limiter workflow:

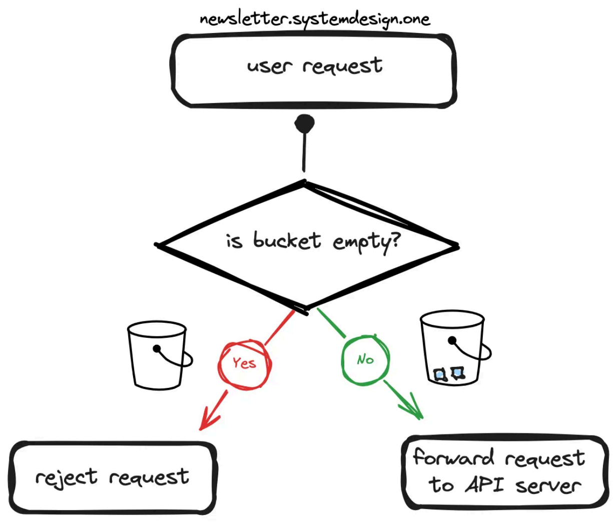


Rate limiter

1. Check rate limiter rule
2. Reject a request if the threshold exceeds
3. Otherwise let the request pass-through

Rate Limiter Implementation

Stripe uses the token bucket algorithm to do rate limiting. Here is a quick overvie this algorithm:



Token bucket algorithm

Imagine there is a bucket filled with tokens. Every request must pick a token fron bucket to pass through.

No requests get a token if the bucket is empty. So, further requests get rejected.

And tokens get refilled at a steady pace.

Other *popular* rate-limiting algorithms are sliding windows and leaky buckets.

They used Redis to build the rate limiter. Because it is in-memory and provides lo latency.

Things they considered when implementing the rate limiter are:

- Quality check rate limiter logic and allow bypass on failures
- Show a clear response to the user: status code 429 - too many requests or service unavailable
- Enable panic mode on the rate limiter. This allowed switching it off on failure
- Set up alerts and monitoring
- Tune rate limiter to match traffic patterns

It's difficult to *rate-limit a distributed system*. Because each request from a singl might not hit the same server. I don't know how Stripe solved this problem. But h a potential solution.

Redirect the traffic from an IP address to the same data center using DNS. And c
an isolated rate limiter in each data center.

Yet a new TCP connection might hit a different server within a data center.

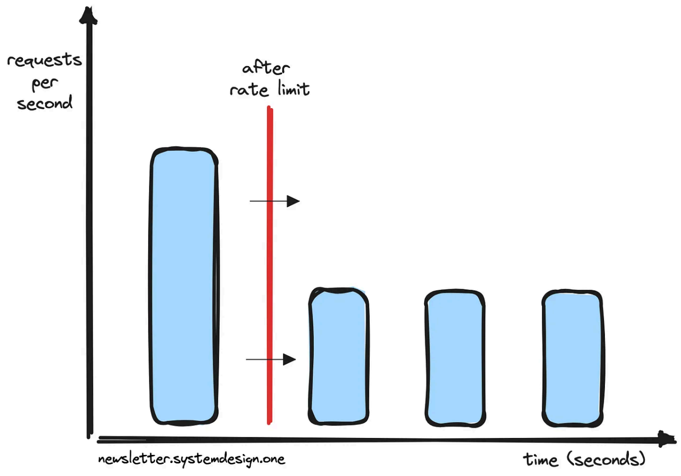
So set up a caching proxy (twemproxy) in each data center. Because it allows to
state across many servers. Put another way, many servers share a single cache:
limiter.

And use consistent hashing to reduce key redistribution on changing load (clust
resize).

Rate Limiting Types

Stripe categorizes rate limiting into 4 types:

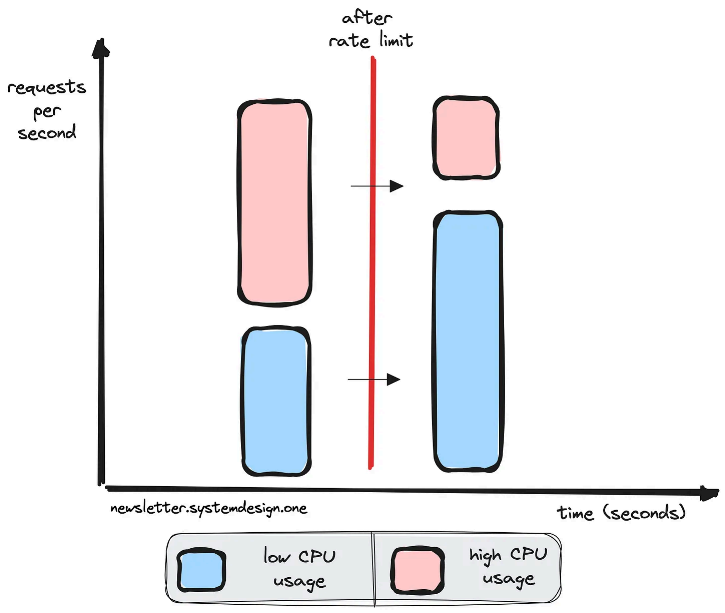
1. Request Rate Limiter



Request Rate Limiter

Each user gets *n* requests per second. This rate limiter type acts as the first line
defense for an API. And it is the most popular type.

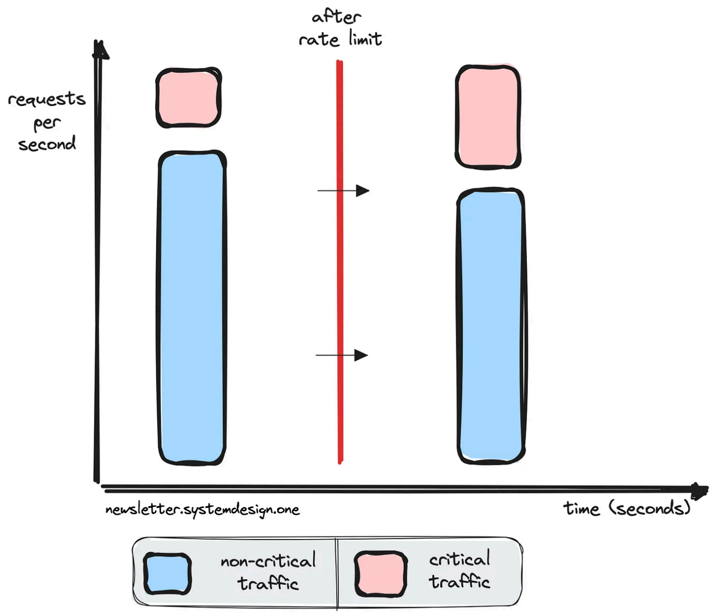
2. Concurrent Requests Rate Limiter



Concurrent Requests Rate Limiter

The number of concurrent requests that are in progress is rate-limited. This prot
resource-intensive API. And prevents resource contention.

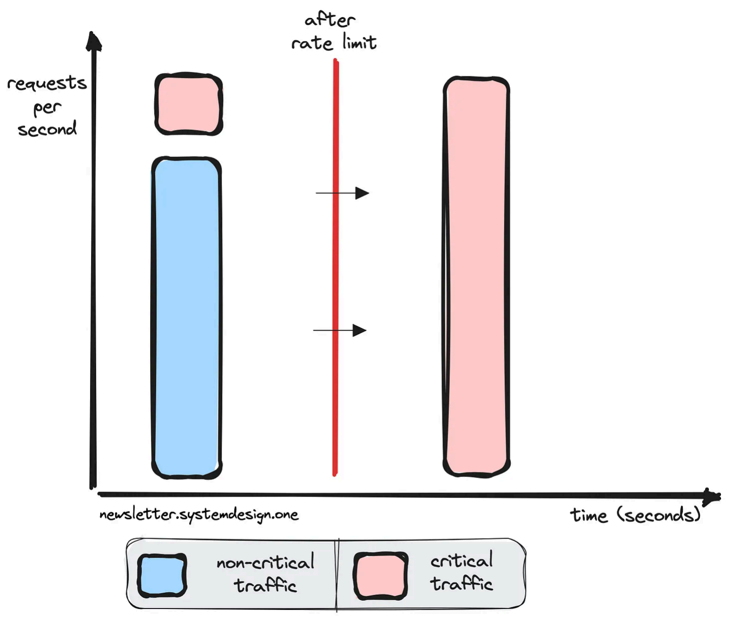
3. Fleet Usage Load Shedder



Fleet Usage Load Shedder

The critical APIs reserve 20% of computing capacity. And requests to *non-critic*
get rejected if the critical API doesn't get 20% of the resources.

4. Worker Utilization Load Shedder



Worker Utilization Load Shedder

The non-critical traffic gets shed on server overload. And it gets re-enabled afte
delay. This rate limiter type is the last line of defense for an API.

👋 PS - Are you unhappy at your current job?

While preparing for system design interviews to get your dream job can be stres

Don't worry, I'm working on content to help you pass the system design intervie
make it easier - you spend only a few minutes each week to go from 0 to 1. Yet p
subscription fees will be higher than current pledge fees.

So pledge now to get access at a lower price.

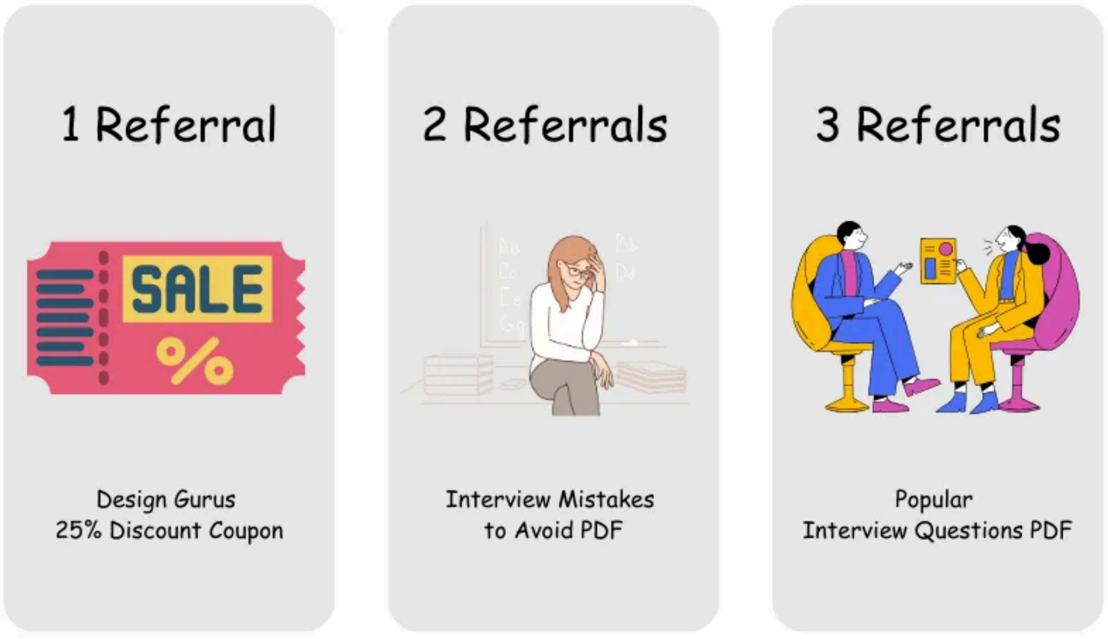
"An excellent newsletter to learn system design through practical case studies."
Franco

Consider subscribing to get simplified case studies delivered straight to your inb

Type your email... Subscribe

Follow me on LinkedIn | YouTube | Threads | Twitter | Instagram | Bluesky

Thank you for supporting this newsletter. Consider sharing this post with your fri
and get rewards. Y'all are the best.



Tech Stack Evolution at Levels.fyi

NK • 24 SEPTEMBER 2023

Read full story →

**11 Reasons Why YouTube Was Able to Support 100 Million Video \
a Day With Only 9 Engineers**

NK • 16 SEPTEMBER 2023

Read full story →

Are you interested in deepening your knowledge of TDD and DDD? Conside
subscribing to [Crafting Tech Teams Newsletter](#) from Denis Čahuk. He has a grea
archive of information in his newsletter.

Word-of-mouth referrals like yours help this community grow - *Thank you*.

Samuel Path @smlpth · 12m

I enjoy the System Design Newsletter (@systemdesign42). Every few days I get a short article explaining what approach some tech company used to overcome a real-world challenge.

It's small and regular. Ideal to slowly grow one's understanding over time.

Feedback from an awesome reader

Get featured in the newsletter: Write your feedback on this post. And tag me on [Twitter](#), [LinkedIn](#), and [Substack Notes](#). Or, you can reply to this email with anony feedback.

References

- https://stripe.com/blog/rate-limiters
- https://www.cloudflare.com/en-gb/learning/bots/what-is-rate-limiting/
- https://blog.cloudflare.com/counting-things-a-lot-of-different-things/

37 Likes · 2 Restacks

Previous

Ne

Discussion about this post

Comments Restacks

Write a comment...

Anton Zaides 28 Sept 2023
Liked by Neo Kim

If you implement the simple request rate limiter - one important part is how to communica
clients. I had 2 cases in the last year where we breached the limit:

1. A free 3rd party api. We had a limit of 500 requests per day (which was not written dow
we didn't pass for a long time. Then we suddenly started to get 403 errors, which immedi
us to think our token was revoked or expired. Only after it started to work at 12:00, did we
understand we breached the limit, and for some reason, they decided to throw unauthoriz
of the standard 429.

2. A paid api, that we heavily use. Recently we reached their limit (100 calls/minute), and s
get 429 responses. The good part, is that in the header we got the time when our limit will
This allowed us to implement internal queueing of the request, without noisy errors. We ha
ones, we stored the requests, and then we continued at the allowed time until we hit it aga
mainly in rare peaks, this solution is perfect for us.

LIKE (6) REPLY

2 replies by Neo Kim and others

2 more comments...