Rate limiting using the Token Bucket algorithm

Source: DEV Community

Author: Amir Keshavarz

Published: December 17, 2020

Tags: Python, Computer Science, Tutorial, Programming

Saved: February 2, 2026

Rate Limiting

The term "rate-limiting" is known to almost everyone who has worked with web servers. I think it's one of those subjects that we often just use and don't think about it. But I think knowing how rate limiting works is useful if you have ever used it.

As you may know, "rate-limiting" is basically monitoring the incoming requests and making sure they don't violate a threshold. Basically, we define a rate of requests per time-unit and discard packets of a stream that is sending more packets than the one defined in our rate.

You probably noticed that I used the word "discard" when our threshold is reached. That's not always the case. In a lot of cases, we put the packets in a queue to keep the packets at a constant output rate. (And packets won't be discarded.)

What we'll be discussing doesn't require a queue and I dare say it's probably the simplest way to implement rate-limiting.

Famous Rate Limiting Algorithms

The most famous ways of implementing rate-limiting (Traffic Shaping) are:

1. **Token Bucket**
2. **Leaky Bucket**
3. **(r, t) Traffic Shaping**

The Leaky Bucket is somewhat similar to the Token Bucket but right now we don't care about the constant output rate and Token Bucket is the only algorithm we will talk about.

Token Bucket

The Token Bucket analogy is very simple. It's all about a bucket and tokens in it. Let's discuss it step by step.

1. Picture a bucket in your mind.
2. Fill the buckets with tokens at a constant rate.
3. When a packet arrives, check if there is any token in the bucket.
4. If there was any token left, remove one from the bucket and forward the packet. If the bucket was empty, simply drop the packet.

That's it. Wasn't it simple?

Token Bucket Visual Representation:

Bucket → Tokens added at constant rate → Packet arrives → Check for token → Forward or Drop

Implementation

Enough talking. Let's write a simple Token Bucket throttler in Python.

We start by defining a class with 4 arguments when it's being instantiated:

1. `tokens` : number of tokens added to the bucket in each time unit.
2. `time_unit` : the tokens are added in this time frame.
3. `forward_callback` : this function is called when the packet is being forwarded.
4. `drop_callback` : this function is called when the packet should be dropped.

We also prefill the bucket with the given tokens. This allows for a burst of packets when the throttler first starts working.

`last_check` is the timestamp that we previously handled a packet. For initialization, we set the time when we created the bucket.

```
import time

class TokenBucket:

    def __init__(self, tokens, time_unit, forward_callback, drop_callback):
        self.tokens = tokens
        self.time_unit = time_unit
        self.forward_callback = forward_callback
        self.drop_callback = drop_callback
        self.bucket = tokens
        self.last_check = time.time()
```

Then, we define `handle()` where the magic happens.

```
def handle(self, packet):  #1
    current = time.time()  #2
    time_passed = current - self.last_check  #3
    self.last_check = current  #4

    self.bucket = self.bucket + \
        time_passed * (self.tokens / self.time_unit)  #5

    if (self.bucket > self.tokens):  #6
        self.bucket = self.tokens

    if (self.bucket < 1):  #7
        self.drop_callback(packet)
    else:
        self.bucket = self.bucket - 1  #8
        self.forward_callback(packet)
```

Code Explanation

1. `handle` accepts only 1 parameter: the packet.
2. The current timestamp is put into `current` .
3. The time passed between now and the previous `handle` call is put into `time_passed` .

4. We update the `last_check` to the current timestamp.
5. **Most important part:** By multiplying the `time_passed` and our rate (which is `tokens / time_unit`) we find out how many tokens need to be added to the bucket.
6. Reset the bucket if it has more tokens than the default value.
7. If the bucket doesn't have enough token, drop the packet.
8. Otherwise, remove one token and forward the packet.

Complete Working Code

That's all. This is the final working version:

```
import time

class TokenBucket:

    def __init__(self, tokens, time_unit, forward_callback, drop_callback):
        self.tokens = tokens
        self.time_unit = time_unit
        self.forward_callback = forward_callback
        self.drop_callback = drop_callback
        self.bucket = tokens
        self.last_check = time.time()

    def handle(self, packet):
        current = time.time()
        time_passed = current - self.last_check
        self.last_check = current

        self.bucket = self.bucket + \
            time_passed * (self.tokens / self.time_unit)

        if (self.bucket > self.tokens):
            self.bucket = self.tokens

        if (self.bucket < 1):
            self.drop_callback(packet)
        else:
            self.bucket = self.bucket - 1
            self.forward_callback(packet)


def forward(packet):
    print("Packet Forwarded: " + str(packet))


def drop(packet):
    print("Packet Dropped: " + str(packet))


throttle = TokenBucket(1, 1, forward, drop)

packet = 0

while True:
    time.sleep(0.2)
    throttle.handle(packet)
    packet += 1
```

Expected Output

You should be getting something like this:

```
Packet Forwarded: 0
Packet Dropped: 1
Packet Dropped: 2
Packet Dropped: 3
Packet Dropped: 4
Packet Forwarded: 5
Packet Dropped: 6
Packet Dropped: 7
Packet Dropped: 8
Packet Dropped: 9
Packet Forwarded: 10
Packet Dropped: 11
Packet Dropped: 12
Packet Dropped: 13
Packet Dropped: 14
Packet Forwarded: 15
```

As you can see in the code, our rate is **1 packet per second**. We also send a packet every **0.2 seconds**. When we first send a packet it quickly empties the bucket and it takes some time for the bucket to be filled again and that's why 4 packets are dropped between each successful forward.

Conclusion

Rate limiting and traffic policing, in general, is a very vast subject so if you liked what you just read, there are many materials available online about this subject that you can use to have a much deeper understanding of traffic policing.

Thank you for your time.

Key Takeaways

- Token Bucket is one of the simplest rate limiting algorithms
- Tokens are added to the bucket at a constant rate
- Each packet consumes one token
- If no tokens are available, packets are dropped
- The bucket has a maximum capacity to prevent unlimited bursting
- Implementation requires tracking time between packet arrivals
- Works well for controlling request rates in APIs and web services

Related Articles in Series

1. **Rate limiting using the Token Bucket algorithm** (This article)
2. Rate limiting using the Fixed Window algorithm
3. Rate limiting using the Sliding Window algorithm

For the full article and discussions, visit: dev.to