API Rate Limiting – Technical Specification

Document Version: 2.1
Date: April 18, 2025
Author: [Your Name]
Classification: Technical Documentation

Overview

This document provides detailed technical specifications for the rate limiting implementation in our API infrastructure. Rate limiting is implemented using a token bucket algorithm that balances consistent throughput with the ability to handle temporary bursts.

Rate Limit Specifications

User API Rate Limits

Tier	Requests Per Minute	Burst Capacity	Concurrent Connections
Standard	100	20 req/sec	5
Premium	300	50 req/sec	15
Enterprise	1000	100 req/sec	50
Custom	Negotiable	Negotiable	Negotiable

Standard tier users are limited to 100 requests per minute per API token. A burst capacity of 20 requests per second is allowed. Exceeding these limits will result in a `429 Too Many Requests` HTTP status code.

Rate Limit Headers

All API responses include the following rate limit headers:

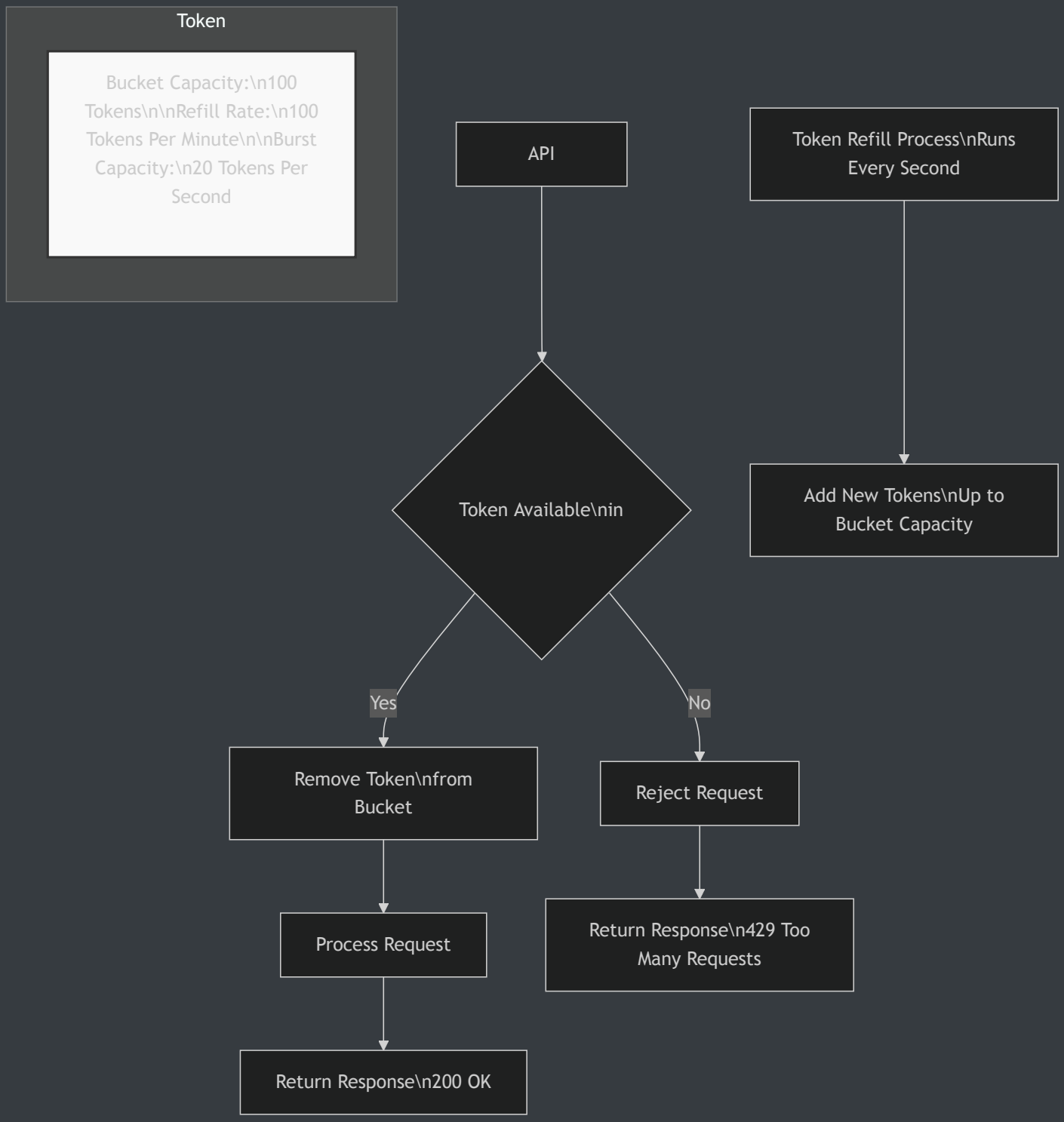
```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 43
X-RateLimit-Reset: 1619084520
```

- `X-RateLimit-Limit` : Total number of requests allowed within the window
- `X-RateLimit-Remaining` : Number of requests remaining in the current window
- `X-RateLimit-Reset` : Unix timestamp when the rate limit window resets

Token Bucket Throttling Algorithm

Our rate limiting implementation uses a token bucket algorithm. This provides a balance between consistent throughput and handling burst traffic effectively.

Figure 3: Token Bucket Throttling Algorithm



Algorithm Pseudo-code

```
function checkRateLimit(userToken):
    bucket = getUserBucket(userToken)
```

```
        if bucket.availableTokens >= 1:
            bucket.availableTokens -= 1
            return ALLOW_REQUEST
        else:
            return REJECT_REQUEST_429

    function refillBuckets():
        // Run this every second
        for each bucket in activeBuckets:
            tokensToAdd = bucket.refillRate / 60  // per second
            bucket.availableTokens = min(bucket.capacity,
    bucket.availableTokens + tokensToAdd)
```

Client Retry Strategy

When receiving a 429 response, clients should implement an exponential backoff strategy:

1. Wait for a base time of 100ms
2. For each consecutive failure, multiply wait time by 2 (100ms, 200ms, 400ms, etc.)
3. Add a small random jitter (±10%) to prevent synchronized retries
4. Cap maximum retry wait time at 30 seconds
5. Abandon after 5 consecutive failures

Monitoring and Alerts

Operations teams receive automated alerts when:

- A user exceeds 90% of their rate limit for 5 consecutive minutes
- System-wide rate limit rejections exceed 5% of total traffic
- Any individual IP address generates more than 1000 rate limit errors in an hour

Implementation Notes

Rate limiting is implemented at two levels:

1. Edge level (Cloudfront/CloudFlare) for coarse IP-based rate limiting
2. API Gateway level for fine-grained token-based rate limiting

The rate limit data store uses Redis with a 1-second precision counter implementation.

Document Approvals:

Role	Name	Date	Signature
API Product Manager			
Platform Engineering Lead			
Security Architect			