Souvik Sarkar, 2025, 13:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

Decentralized Rate Limiter

Souvik Sarkar
Information Technology, Institute of Engineering and Management
Kolkata, West Bengal, India

Abstract- As distributed systems operate at increasingly high throughput, enforcing fair and efficient request control becomes critical to safeguard service reliability and prevent misuse. This work introduces a decentralized rate limiting mechanism engineered for scalability and resilience without central oversight. The solution integrates Conflict-free Replicated Data Types (CRDTs) for consistent state sharing and employs the libp2p gossip protocol to synchronize nodes through peer-to-peer communication. Each peer manages an LRU-based in-memory cache for frequent users and offloads less active records to disk, balancing performance with persistence. Performance evaluations indicate that each node can independently process up to 3,000 requests per second, maintaining a 99th percentile latency below 2 milliseconds. CRDT-based synchronization across peers shows convergence latencies around 2 ms with compact gossip payloads averaging 3 KB. These findings validate the feasibility of a decentralized architecture for rate limiting, offering a robust alternative to traditional centralized techniques in modern cloud-native systems.

Keywords: distributed systems, libp2p, peer-to-peer,

I. INTRODUCTION

In large-scale distributed environments, regulating client request rates is vital for protecting services from overload, ensuring equitable access, and preserving system responsiveness. Traditional rate limiting methods—often relying on centralized servers using algorithms like Token Bucket or Leaky Bucket, and backed by memory-oriented stores such as Redis or Memcached—have proven effective in controlled, small-scale deployments.

However, as systems evolve toward decentralized and geographically distributed models—including edge computing setups, globally replicated services, and peer-to-peer networks—the limitations of centralized rate limiting become evident. These conventional models often fail to scale effectively, introduce latency due to remote coordination, and act as single points of failure.

The growing adoption of decentralized system paradigms necessitates a shift toward distributed rate limiting solutions that can maintain correctness, fairness, and efficiency without centralized coordination. Emerging technologies like Conflict-free Replicated Data Types (CRDTs) offer strong guarantees for consistency without locking, while libp2p provides robust peer-to-peer communication primitives. Together, they provide a compelling foundation for a reimagined, decentralized approach to rate control.

Problem Statement

Although widely used, conventional rate limiters suffer from two fundamental issues:

- **Centralization and Scalability Limits:** Relying on a single authoritative store hampers the ability to scale horizontally and creates a vulnerability point that can impact system availability and performance in large, distributed deployments.
- **Low Fault Tolerance and Partition Handling:** Centralized systems often struggle under node failures, network partitions, or latency spikes, potentially leading to rate miscalculations—

either under-enforcing or over-enforcing user limits.

Moreover, naive replication strategies in distributed setups may introduce race conditions or inconsistency when concurrent updates occur across multiple nodes.

Aims and Objectives

This research proposes a fully decentralized, CRDT-backed rate limiting system designed to be fault-tolerant, efficient, and scalable. The key objectives are:

- Design a CRDT-driven token bucket algorithm that can handle concurrent updates while ensuring deterministic convergence across distributed peers.
- Implement decentralized state synchronization using a libp2p-based gossip protocol that efficiently disseminates state deltas with minimal overhead.
- Optimize latency for active users through an in-memory LRU cache, while using disk persistence to handle less frequent traffic gracefully.
- Benchmark key performance metrics, including per-node throughput, request latency at various percentiles, CRDT synchronization lag, and gossip message sizes.
- Demonstrate resilience under failure conditions, such as node crashes and partition events, verifying the system's ability to recover and converge without manual intervention.

II. LITERATURE REVIEW

Rate limiting has long been a cornerstone of service reliability, providing mechanisms to regulate client interactions and maintain stability under load. Over time, various strategies and architectures have been developed—each offering distinct advantages and trade-offs depending on deployment scale and fault tolerance requirements.

Centralized Rate Limiting Approaches

Traditional implementations commonly use centralized components to monitor and enforce rate limits. In such architectures, a central
service or API gateway acts as the rate control authority, often backed by fast-access in-memory data stores like Redis or Memcached. Popular platforms, such as Envoy and Kong, support plugin-based configurations of token bucket algorithms within these central stores.

These methods generally work well in tightly controlled or small-to-medium-sized environments. However, as system complexity grows, they encounter scaling limitations. The central point of coordination becomes a bottleneck, and round-trip latency to a central store—especially in geo-distributed deployments—can severely degrade performance and responsiveness.

Partitioned-Tolerant and Sharded Solutions

Some modern designs attempt to overcome the scalability issues of centralized systems through horizontal sharding. In these setups, rate limit states for users are distributed across multiple nodes. Although this method improves throughput and resource utilization, it remains vulnerable to network partitioning. When clients move between nodes or when network links fail, these systems often suffer from duplication or loss of state updates, leading to inaccurate rate enforcement.

Efforts to enhance resilience have involved techniques like leader election or the use of strongly consistent distributed databases such as Etcd and Consul. However, these come at the cost of high coordination overhead and reduced availability—an inherent consequence of the CAP theorem.

Conflict-Free Replicated Data Types (CRDTs)

CRDTs have emerged as a promising solution for building coordination-free distributed systems. They provide mathematically guaranteed convergence of replicated state, even under concurrent modifications, without requiring global synchronization.

Originally developed for collaborative editing platforms (e.g., Yjs, Automerge) and distributed counters, CRDTs have since been extended to support more sophisticated constructs such as bounded counters—ideal for modeling token consumption with refill mechanisms. Their ability to guarantee idempotent, associative, and

commutative operations makes them well-suited for rate limiting in distributed environments.

Gossip Protocols for State Dissemination

Peer-to-peer synchronization methods based on gossip protocols—such as those used in libp2p, Serf, and Amazon's Dynamo—offer a scalable and fault-tolerant means of sharing state information across nodes. Gossip-based dissemination randomly exchanges updates between peers in a cluster, ensuring rapid and redundant propagation with minimal need for centralized oversight.

When combined with delta-based CRDTs, these protocols can share only the changes (deltas) instead of full state snapshots, minimizing network traffic and improving convergence speed. This synergy forms the foundation for effective decentralized rate limiting.

Identified Gaps in Existing Work

Despite the availability of distributed key-value stores and pub/sub architectures, there is a notable absence of purpose-built systems that combine CRDTs with gossip protocols for decentralized rate control. Existing literature and tooling often either retain centralized assumptions or trade off availability for consistency.

This research addresses this gap by introducing a system that leverages a CRDT-enhanced token bucket algorithm in conjunction with libp2p-based delta gossiping. The aim is to provide an autonomous, eventually consistent enforcement mechanism that can thrive under real-world distributed conditions.

III. METHODOLOGY

This section describes the architecture, components, algorithms, and testing methodology used to design and evaluate the decentralized rate limiting system. The focus is on building a highly available and scalable solution using peer-to-peer synchronization and CRDT-based rate limiters.

System Overview

The proposed system comprises a distributed set of nodes that collaboratively enforce rate limits without relying on a central authority. Each node independently processes incoming requests using a local token bucket algorithm and synchronizes state updates with peers to maintain global consistency.

The system's core components are:

- **Token Bucket Limiter:** Manages per-user request quotas locally at each node.
- **LRU Cache:** Stores active user states in memory for rapid access.
- **Disk Store:** Holds evicted or infrequent user states for long-term durability.
- **Delta Store:** Temporarily tracks recent changes for gossip propagation.
- **Gossip Layer:** Periodically synchronizes delta updates with neighboring nodes via libp2p.

Figure. 1: System Design

This design ensures responsiveness along the hot path while maintaining convergence and fault tolerance across nodes.

CRDT-Compatible Token Bucket

Each node enforces user-specific limits through a bounded token bucket. The structure tracks token consumption and refill rates, but in a format that enables CRDT-style merging. This ensures safe and conflict-free convergence across peers.

The main operations include:

- **Consume():** Deducts a token if available.
- **Refill():** Restores tokens at a configured rate based on elapsed time.
- **Merge():** Reconciles bucket states from remote peers using deterministic merge logic.

The bucket is represented as:

type TokenBucket struct { Capacity float64

- RefillRate float64
- UsedTokens float64
- UsableTokens float64
- LastRefilled time.Time SyncIssuedAt time.Time }

All counter fields (UsedTokens, UsableTokens, etc.) are modeled as grow-only to ensure eventual consistency. Merges respect token capacity constraints, and time fields guide accurate token refilling and synchronization.

Delta Tracking and Caching Strategy

To reduce bandwidth and avoid full state sharing, only modified buckets are tracked in a temporary delta map. Two levels of caching are used:

- **Hot Path**: Frequently accessed user data is kept in an in-memory LRU cache for sub-millisecond access.
- **Cold Path:** Evicted or infrequent users are persisted to disk and loaded only when required. Mutexes protect cache access and prevent race conditions. Snapshotting for metrics or logs uses copy-on-write strategies to minimize disruption to request processing.

Peer Communication via libp2p Gossip

Nodes form a mesh network using libp2p's gossip protocol. State synchronization proceeds in cycles:

- Local deltas are collected using toMessage().
- These deltas are transmitted to randomly selected peers.
- Received deltas are merged into the local CRDT state.

The gossip interval is configurable (e.g., every 100ms or after a certain number of updates), balancing convergence speed with network efficiency. Messages are compactly serialized (typically ~1.5 KB) for low-overhead propagation.

Metrics and Instrumentation

The system includes a lightweight instrumentation module to capture:

- **Request Throughput:** Total requests processed per node.
- **Response Time:** p50, p95, p99 latency statistics.
- **CRDT Sync Latency:** Time from delta creation to merge.
- **Message Size:** Gossip message payload distribution.

Metrics are logged periodically (every 10 seconds) and can be toggled at runtime using an environment flag. This avoids dependency on external tools like Prometheus, keeping the system portable and lightweight.

Experimental Setup

The benchmarking environment was configured as follows:

- **Cluster:** 3 peer nodes running as Docker containers.
- **Load Balancer:** NGINX with round-robin request distribution.
- **Load Generator:** Vegeta to simulate real-world usage patterns.
- **Test Parameters:**
- 1,000 simulated users.
- Load ramped from 1,000 to 3,000 RPS.
- Metrics collected: response codes, per-node latency, CRDT convergence times.

This setup was designed to mimic production-like conditions and evaluate the system's performance under varied loads and failure scenarios.

IV. RESULTS AND ANALYSIS

This section presents the empirical performance results obtained through controlled experiments on the decentralized rate limiting system. The analysis focuses on request throughput, response latency, CRDT synchronization efficiency, bandwidth usage, cache behavior, and fault tolerance.

Request Throughput and Latency

Using a 3-node libp2p-based cluster and Vegeta for traffic generation, the system was subjected to increasing request loads ranging from 1,000 to 3,000 RPS per node.

- **Sustained Throughput:** Each node independently handled ~3,000 requests per second, leading to an aggregate cluster throughput of ~9,000 RPS.
- **Latency:** The 99th percentile response time (p99) consistently stayed under 2 milliseconds, which includes token validation, locking, and any necessary caching operations.

Latency distributions remained stable across nodes, indicating the architecture's ability to maintain performance under pressure without significant variability.

Metric	Value
Peak Throughput	3,000 RPS / node
Average Response Time	0.6 ms
p95 Response Time	1.1 ms
p99 Response Time	2 ms
429 Response Ratio	Matches configured rate limits (±2%)

Synchronization Latency of CRDTs

The time taken to propagate and integrate CRDT deltas across peers was also measured.

Metric	Value
Average Sync Latency	0.4 ms
p95 Sync Latency	0.7 ms
p99 Sync Latency	2 ms
Convergence Failures	0 (across 10M ops)

These values reflect high efficiency in state convergence, demonstrating that the gossip mechanism enables near real-time synchronization while maintaining consistency.

Gossip Message Size and Bandwidth Utilization

To evaluate the communication overhead of the gossip protocol, the system tracked the size of outgoing delta messages:

Metric	Value
Avg Message Size	1.5 KB
p95 Message Size	2 KB
p99 Message Size	3 KB
Max Message Size	7 KB

Because each message includes only modified user buckets, the overall bandwidth usage remains low, even during load spikes. This supports horizontal scalability and efficient operation in bandwidth-constrained environments.

Hot vs. Cold Path Performance

The system categorizes requests based on whether the corresponding user bucket is in memory (hot) or requires a disk read (cold).

Testing with 10,000 unique users yielded the following behavior:

- Hot Path: Served ~75–80% of all requests, with response times averaging ~1.2 ms at p99.
- Cold Path: Comprised ~20–25% of requests, with higher latencies in the range of 12–18 ms due to disk access.

Path Type	p99 Latency	Ratio
Hot Path	1.2 ms	~80%
Cold Path	12–18 ms	~20%

This illustrates that the caching strategy effectively accelerates the majority of traffic while providing fallback support for infrequent users.

Fault Tolerance and Node Recovery

To test resilience, various failure scenarios were simulated:

- **Node Crash:** When a peer was stopped, the remaining nodes continued processing and synchronizing without impact.
- **Node Rejoin**: Upon rejoining, the previously failed node merged missed deltas within 200 milliseconds, ensuring a consistent state without conflicts.
- The system consistently demonstrated:
- No loss in rate-limiting accuracy during node absence.
- Smooth reintegration upon recovery.
- Zero convergence failures across 10 million operations.

These results confirm the system's robustness against common distributed failure conditions, reinforcing its suitability for high-availability deployments.

V. DISCUSSION AND CONCLUSION

Discussion

The experimental results validate that decentralized rate limiting, when constructed with CRDTs and gossip-based synchronization, is not only viable but highly effective for high-throughput distributed systems.

By using local token buckets with CRDT semantics, the system achieves fast, low-latency request handling without sacrificing consistency. The LRU-based caching architecture ensures that frequently accessed user states remain in memory, while less active data is persisted efficiently. The gossip protocol, driven by libp2p, enables fault-tolerant and lightweight state propagation between peers.

The key takeaway is that the system balances local responsiveness and global convergence without needing centralized orchestration, making it ideal for modern edge, mesh, and microservice environments.

Design Trade-offs

One deliberate design decision is the adoption of eventual consistency instead of enforcing strict global consistency. While this means that short-term burst violations are possible before state convergence, CRDTs ensure such deviations are eventually reconciled. This trade-off suits applications where soft limits are acceptable—like API gateways or per-user throttling—but may not be suitable for use cases that demand precise, hard quotas (e.g., billing systems).

Another critical trade-off involves caching strategy. The combination of in-memory LRU caching and disk persistence allows the system to scale beyond memory-bound limits. However, it introduces higher latency for cold requests. Tuning the eviction policy or preloading key users might be necessary in latency-sensitive applications.

Limitations

While the system performs well under simulated stress and fault conditions, several areas are yet to be addressed:

- **Strict Global Quotas:** The current system cannot guarantee tight quotas across all peers within strict time intervals.
- Topology Awareness: There's no logic to route requests based on proximity or load (e.g., sending traffic to a "hot" node).
- **Tenant-Aware Rate Limiting:** Multi-tenant support with variable limits and priority classes has not yet been implemented.
- **Clock Dependency:** Although logical timestamps are used, high degrees of clock drift between nodes could impact sync latency metrics—though not correctness.

VI. CONCLUSION

This research presents a fully decentralized rate limiting system that is scalable, fault-tolerant, and efficient. By combining CRDTs with libp2p-powered gossip, the architecture eliminates central coordination while ensuring eventual consistency and high throughput.

The system consistently handles thousands of requests per second per node with sub-2ms latency, all while maintaining delta-based synchronization and compact bandwidth usage. It recovers seamlessly from node failures, offers hybrid memory-disk caching, and supports consistent peer merging without conflicts.

Ultimately, this work demonstrates that robust rate limiting can be achieved in a decentralized way—

paving the path for scalable, self-healing infrastructure suitable for the next generation of cloud-native and edge computing platforms.

FUTURE WORK

Adaptive Gossip Scheduling

Presently, the gossip synchronization interval is fixed or event-triggered based on a threshold of state changes. Future iterations could benefit from dynamic gossip scheduling, where the frequency of synchronization adapts in real-time based on traffic patterns, observed convergence delays, or cluster stability. Such adaptability would reduce unnecessary traffic under low load while accelerating convergence during bursts or re-joins.

Geo-Aware Peer Coordination

In multi-region deployments, latency between nodes can affect the timeliness of rate limit enforcement. Incorporating geographical awareness into the gossip layer—such as by biasing peer selection based on latency proximity or data center location—could reduce synchronization lag and improve regional consistency. Additionally, sharding the mesh by location could further scale the system without sacrificing convergence speed.

Hybrid Enforcement Models

For high-value or security-critical endpoints, strict rate limits may be required. A possible enhancement is a quorum-based enforcement mechanism, where certain high-sensitivity operations require consensus or strong guarantees from a subset of peers before proceeding. This would blend eventual consistency for general traffic with stronger guarantees for sensitive workflows, offering a flexible consistency model.

Pluggable Storage Backends

The current system uses a native file-based store for persisting inactive user states. In the future, introducing pluggable storage engines like LevelDB, BadgerDB, RocksDB, or even remote object stores (e.g., Amazon S3, IPFS) could offer improved durability, portability, or replication capabilities—particularly for edge deployments or multi-tenant architectures.

These enhancements would expand the system's versatility across a broader set of use cases—ranging from mobile edge networks to cloud-native multi-tenant infrastructures—while pushing the envelope on decentralization, adaptability, and reliability.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Prof. Dr. Sanchita Ghosh for her invaluable guidance, thoughtful critiques, and unwavering encouragement throughout the course of this research. Her expertise and mentorship have been instrumental in shaping both the technical and conceptual aspects of this work.

I am also deeply thankful to Prof. Dr. Moutushi Biswas Singh, Head of the Department of Information Technology, for her consistent support, visionary leadership, and dedication to academic excellence, which have profoundly influenced my learning experience over the past four years.

Finally, I extend heartfelt thanks to the faculty members and my peers in the department for their constructive feedback, stimulating discussions, and continuous motivation that helped bring this project to life.

COMPLIANCE WITH ETHICAL STANDARDS

- **Conflict of Interest**

The author declares that there are no conflicts of interest related to the content or results presented in this work.

- **Data Availability**

All source code and experiment artifacts developed as part of this study are publicly available at the following GitHub repository: https://github.com/souviks22/decentralized-ratelimiter

- **Author Contribution**

This paper is the sole work of the author, including system design, implementation, benchmarking, and documentation.

- **Ethical Approval**

This study does not involve any experiments with human participants or animals.

REFERENCES

1. G. Fairbanks, Just Enough Software Architecture: A Risk-Driven Approach, Marshall & Brainerd, 2010.
2. Envoy Proxy, "Rate Limit Architecture," [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/int ro/arch_overview/other_features/global_rate_limiting
3. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Commun. ACM, vol. 59, no. 5, pp. 50–57, 2016.
4. M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," Stabilization, Safety, and Security of Distributed Systems, vol. 6976, pp. 386–400, 2011.
5. A. Bieniusa et al., "An optimized conflict-free replicated set," arXiv preprint arXiv:1210.3368, 2012.
6. G. DeCandia et al., "Dynamo: Amazon's Highly Available Key-value Store," in Proc. 21st ACM SOSP, 2007, pp. 205–220.
7. Protocol Labs, "libp2p Specification," [Online]. Available: https://libp2p.io
8. T. T. Nguyen and D. T. Tran, "Decentralized Access Control with CRDTs in Edge Computing," in Proc. IEEE ICC, 2020.
9. T. Senart, "Vegeta – HTTP load testing tool," GitHub, [Online]. Available: https://github.com/tsenart/vegeta