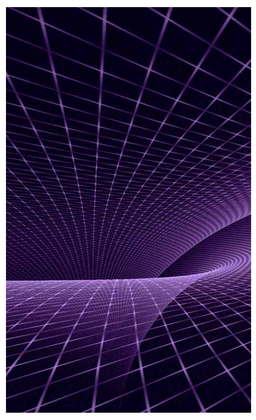


Setchain Algorithms for Blockchain Scalability

Arivarasan Karmegam
IMDEA Networks Institute
Universidad Carlos III de Madrid,
Madrid, Spain

Gabina Luz Bianchi
Universidad Nacional de Rosario
Rosario, Argentina

Margarita Capretto
IMDEA Software Institute
Universidad Politécnica de Madrid
Madrid, Spain

Martín Ceresa
IMDEA Software Institute
Madrid, Spain

Antonio Fernández Anta
IMDEA Software Institute
IMDEA Networks Institute
Madrid, Spain

César Sánchez
IMDEA Software Institute
Madrid, Spain

Introduction

- Blockchain scalability remains a major bottleneck due to strict transaction ordering, limiting throughput and increasing latency.
- Traditional blockchains suffer from low throughput, requiring scalability solutions like Layer 2 rollups, sharding, and alternative consensus mechanisms.
- Setchain, a scalable, reliable distributed object, addresses these challenges by relaxing total ordering while ensuring security and consistency.

Setchain

- **Epoch-Based Structure:** Client elements are grouped into sets, called *epochs,* where elements within an epoch remain unordered, enabling parallel validation and improved throughput. Epochs follow a strict sequential order.
- **Operations:**

→ $add(e)$: Client submits transactions.

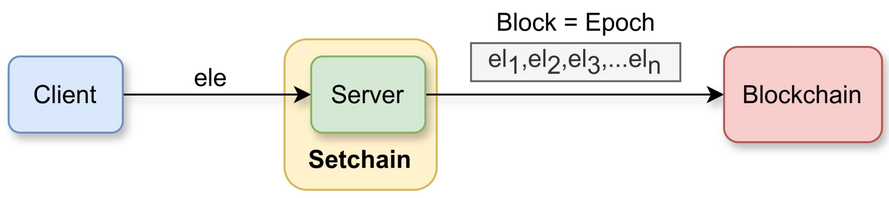
→ $get()$: Clients retrieve the current state of Setchain.

- **Epoch Proofs:** Cryptographically signed proofs for each epoch, allowing light clients to validate epochs by contacting just one honest server.

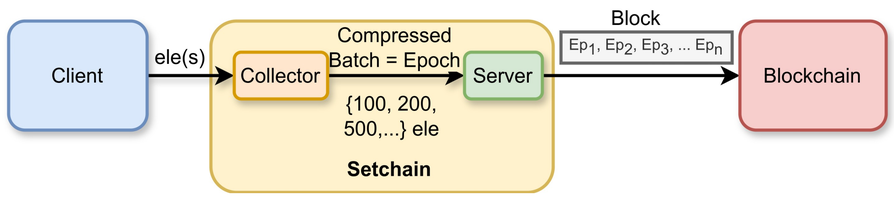
Setchain Algorithms

The Setchain algorithms proposed here use a block-based ledger, which is a Byzantine-tolerant distributed object that maintains a sequence of blocks, each containing a sequence of transactions.

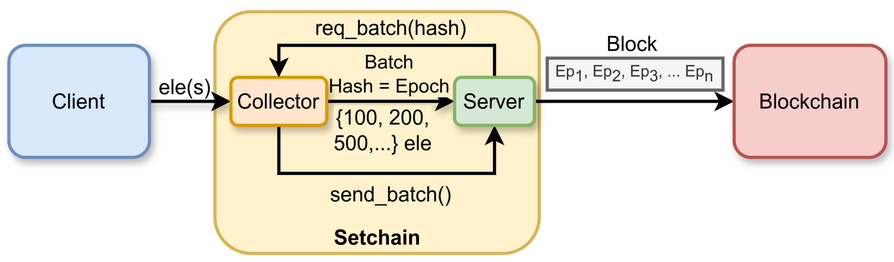
Vanilla: A naive implementation of Setchain, where each client element is a Ledger transaction, and blocks are epochs.



Compresschain: Groups elements into batches. Each batch is compressed and appended to the ledger as a single transaction. Each batch is an epoch.



Hashchain: Instead of appending the (compressed) batch, the hash of the batch is appended to the ledger as a single transaction. A distributed hash-reversal mechanism retrieves the original batch contents when needed.



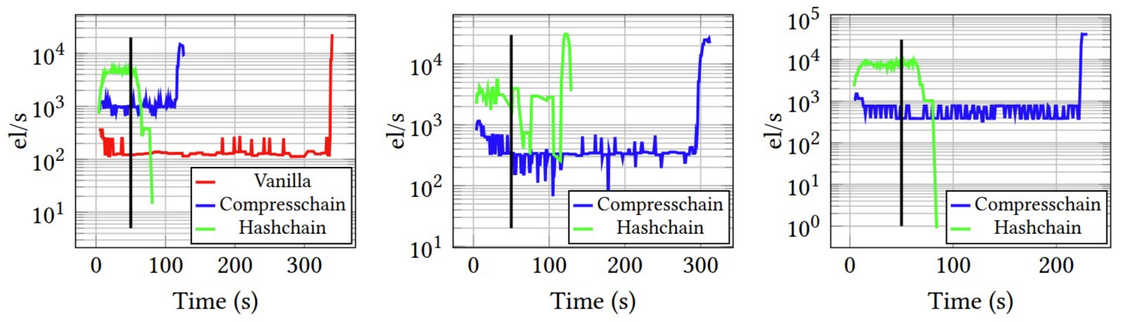
Implementation and Experimental Setup

- Implemented in Golang on the CometBFT blockchain framework.
- Docker-based cluster running on machines each with an Intel(R) Xeon(R) E-2186G CPU @ 3.80GHz with 12 cores, 32GB RAM, and running Debian GNU/Linux 11 (bullseye).
- Uses SHA512 for hashing, Ed25519 for signing, and Brotli for compression.

Results

Results show that Hashchain consistently outperforms Vanilla and Compresschain across all performance metrics. Hashchain achieves the highest throughput, efficiently handling large transaction loads.

Throughput



(a) Sending rate = 5,000 el/s, Collector size = 100

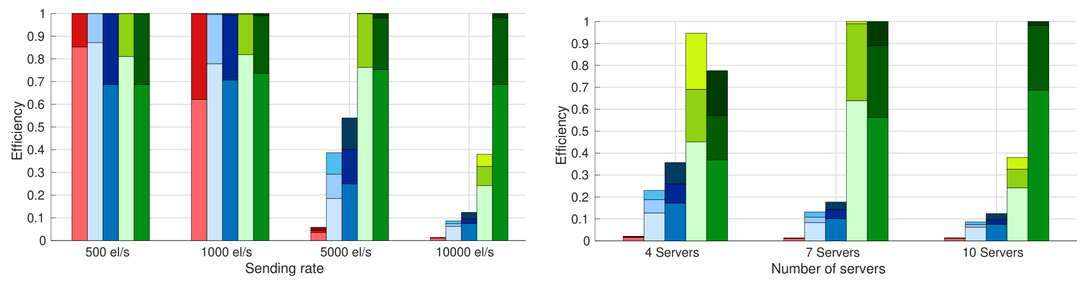
(b) Sending rate = 10,000 el/s, Collector size = 100

(c) Sending rate = 10,000 el/s, Collector size = 500

Figure 1: Throughput over time of the Setchain algorithms for different sending rates. Solid lines plot the rolling average number of elements committed in 9 seconds. The vertical bar marks the time clients add the last element (roughly after 50 s).

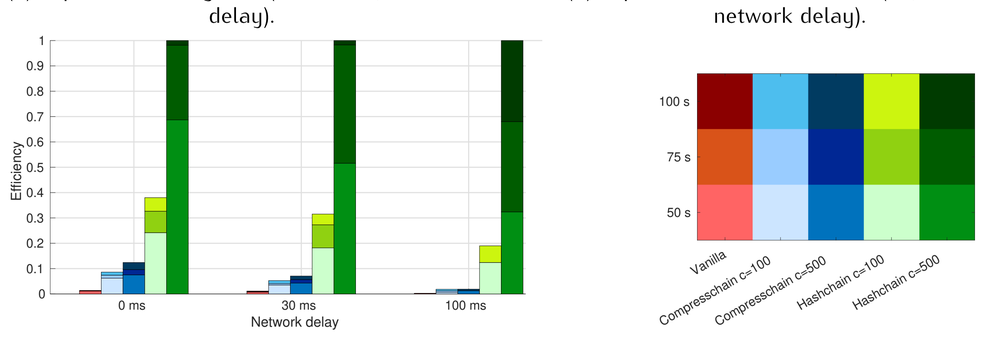
Efficiency

To quantify easily the level of stress of an algorithm we define and use a metric that we call *efficiency*. The efficiency is obtained by dividing the number of elements committed by the total number of elements added.



(a) Impact of sending rate (10 servers, 0 network delay).

(b) Impact of number of servers (10,000 el/s, 0 network delay).

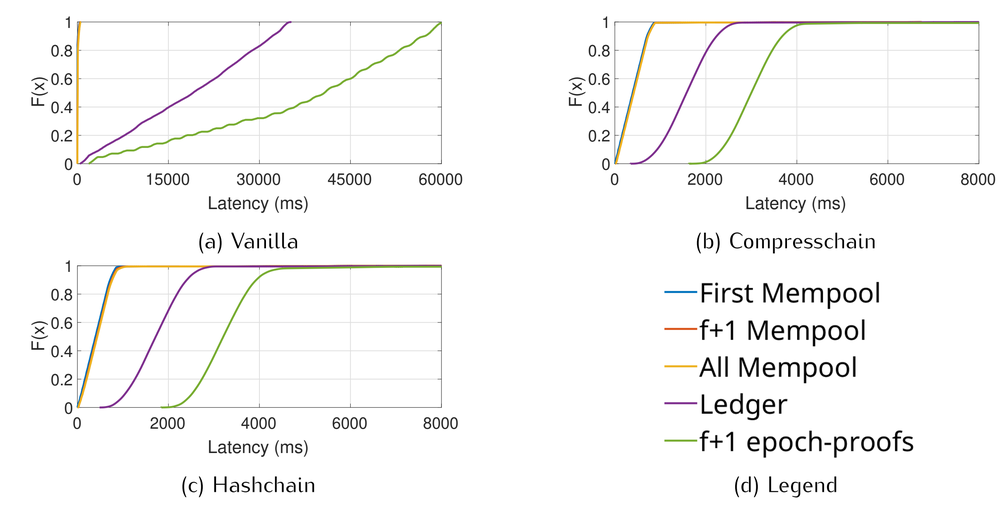


(c) Impact of network delay (10 servers, 10,000 el/s).

(d) Color codes

Figure 2: Efficiency values observed under different scenarios. The base scenario has 10 servers, a sending rate of 10,000 el/s, and no (0) network delay.

Latency



(a) Vanilla

(b) Compresschain

(c) Hashchain

(d) Legend

Figure 3: Cummulative distribution function $F(x)$ of the latency experienced by the elements added to the Setchain to reach several stages in their process. The scenario is with 10 servers, a sending rate of 1,250 el/s, and no network delay.

Acknowledgement

This work is funded in part by a research grant from Nomadic Labs and the Tezos Foundation, and by MICIU/AEI /10.13039/501100011033/, ERDF, and the ESF+ under predoctoral training grant PREP2022-000373, grant DRONAC (PID2022-140560OB-I00), and grant DECO (PID2022-138072OB-I00).