



# Blockchain Made Lightweight: A Median Rule for State Machine Replication

Christian Cachin <sup>1</sup>, Jinfeng Dou <sup>2</sup>, Christian Scheideler <sup>2</sup>, and Philipp Schneider <sup>1</sup>

**Abstract:** We present a lightweight solution for state machine replication. Specifically, we show how a simple median rule for the stabilizing consensus problem [Do11] can be adapted to obtain a low-latency solution for state machine replication. Further, in our solution servers only need to maintain the part of the state machine pertaining to uncommitted commands, and clients hold certificates for each committed command. Our approach remains resilient even under denial-of-service (DoS) attacks on the servers. The protocol guarantees liveness as long as DoS attacks remain below a certain threshold but has the ability to quickly recover after a massive DoS attack.



**Keywords:** Blockchain, consensus, state machine replication



Since the rise of Bitcoins, blockchains have attracted significant attention in both theory and practice. Excluding security concerns, a blockchain can be viewed as a solution to the classical *state machine replication* (SMR) problem [Sc90]. In the SMR problem there is a set of servers that are supposed to replicate a state machine in a consistent way by agreeing on an order in which commands from clients are to be executed on it. More specifically, the servers must reach a solution that satisfies the following two conditions:

- **Safety:** At all times, for any pair of correct servers, the sequence of committed commands on one server is a prefix of the sequence on the other server.
- **Liveness:** Every command submitted by a client will eventually be committed on all correct servers.

State machine replication (SMR) requires consensus on the sequence of committed commands across all servers. To achieve this, we adapt a simple median rule in which each server, in every round, contacts a constant number of random servers and sets its new view to the median of the received views. We show that this approach enables servers to agree on a command's position in the sequence of commands in near-constant<sup>3</sup> time. After agreeing on a command's position, a server can commit the command, ensuring that all servers reach the same state and thereby satisfy safety and liveness.

When commands are serialized on a publicly known blockchain, the shared state does not need to be explicitly maintained. However, this can cause the blockchain to grow very large (as seen with Bitcoin). We propose an alternative approach in which servers only store commands in a blockchain when it is not yet safe to assume that consensus on their position has been reached, while all remaining commands are executed on the shared state.

<sup>1</sup> University of Bern, Bern, Switzerland, christian.cachin@unibe.ch,  <https://orcid.org/0000-0001-8967-9213>; philipp.schneider2@unibe.ch,  <https://orcid.org/0000-0001-9660-1270>

<sup>2</sup> Paderborn University, Paderborn, Germany, jfdou@mail.upb.de,  <https://orcid.org/0009-0004-8852-5182>; scheideler@upb.de,  <https://orcid.org/0000-0002-5278-528X>

<sup>3</sup> By near-constant, we mean a function that grows at most logarithmically in  $n$ .

A drawback of this method is that servers and clients lacking the complete history cannot prove that their commands have indeed been committed. To address this, we give a procedure that provides clients with a certificate for each committed command, that allows them to prove to others that it was indeed committed. Notably, our procedure does not rely on message authentication via asymmetric cryptography, where the private key might expire, be stolen, or even be broken over time. Consequently, small certificates for committed commands cannot simply come from a single server's signed statement. Instead, we use a collision-resistant one-way hash function to store compressed information about the sequence of committed logs in a Merkle hash forest.

The approaches described so far are robust against any 1-late  $\beta$ -blocking adversary with a constant  $\beta > 0$ . Such an adversary can block up to a  $\beta$ -fraction of servers in any communication round, using complete knowledge of each server's state up to the beginning of the previous round. This models denial-of-service attacks or network failures, accounts for partial asynchrony, and covers the crash-recovery model, in which the adversary controls both crashes and subsequent recoveries. Although our methods withstand a  $\beta$ -blocking adversary, our goal is to enable the protocol to recover safely from periods where an arbitrary number of servers may be blocked for arbitrarily long. In particular, we want to ensure that upon recovery, servers arrive at a state containing all previously committed commands. In summary, our solution has the following properties (for further details, see [DS24].):

- **Near-constant communication overhead** per server per client command.
- **Near-constant latency** from receiving a client-issued command by some server to the commitment of the command on all servers.
- **Small memory requirement per server and client**, where servers only have to store near-constant information about the shared state.
- **Near-constant memory** for command certificates that allow a client to prove to a server that its command was committed.
- **Resilience** against a 1-late  $\beta$ -blocking adversary.
- **A recovery procedure** that ensures safe continuation of the protocol, even after an *arbitrary* number of servers was blocked for *any* period of time.

## Bibliography

- [Do11] Doerr, Benjamin; Goldberg, Leslie Ann; Minder, Lorenz; Sauerwald, Thomas; Scheideler, Christian: Stabilizing consensus with the power of two choices. In: Proc. of ACM SPAA 2011. pp. 149–158, 2011.
- [DS24] Dou, Jinfeng; Scheideler, Christian: Invited Paper: Blockchains made Lightweight: A Median Rule for State Machine Replication. In: ApPLIED 2024, Nantes, France, 17 June 2024. ACM, pp. 1–5, 2024.
- [Sc90] Schneider, Fred B.: Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Computing Surveys, 22(4):299–319, 1990.