

Offscale’s consensus algorithm

Vladimir Komendantskiy

2nd October 2019

Abstract

Using prior work on Lachesis consensus [3] and following informal implementation notes by Offscale [9], I am aiming at defining a gossip-inspired consensus algorithm in a way amenable to incremental implementation and formal verification.

1 Introduction

Replicated state machines underpin blockshain technology. Blockchains whose state contains a unique chain of blocks follow replicated state machine design to achieve replication of the same chain across the network.

It should be noted that Offscale’s design is one of a *blockchain* since, as Maxim Zakharov put it in [9] exactly, “tasks/problems the consensus layer solves: ...creating a *linear order of all events* of a frame based on lamport timestamps of the events and synchronisation patterns”. To put the quote in context, it should be added that frames are linearly ordered too. Hence all events that are members of frames are linearly ordered. Therefore an Offscale blockchain pursues the same fundamental goal as most existing blockchains such as Bitcoin or Ethereum. However the design of the replicated state machine used by such an Offscale blockchain is different due to choices made to address inherent problems in today’s decentralised mainstream blockchains: low transaction rate, low transaction confirmation speed and strong synchrony assumptions.

By increasing the rate at which transactions can be submitted to the network, and submitted transactions confirmed, an Offscale blockchain aims to solve a part of the scalability problem which limits the use of blockchain technology to the “store of value” use cases, and to apply thus improved blockchain technology in domains that strongly rely on high transaction throughput such as large-scale heterogenous computational networks commonly referred to as the Internet of Things.

By lifting synchrony assumptions an Offscale blockchain addresses security concerns of synchronous protocols, Bitcoin included, which require setting an interval between blocks to at least the time greater than the maximum message delay on the network [6]. The minimum interval requirement poses a straightforward problem for such protocols: what if the expected maximum message delay does get exceeded for a majority of participants making them unable to produce or receive blocks, thereby making it possible for a minority of participants construct the longest (and therefore valid) chain in the meantime? This is an attack vector on a synchronous blockchain which can be exploited by a maliciously conspiring adversary. In an Offscale blockchain and in an asynchronous blockchain in general, this kind of a minority adversary will not be able to mutate the global state because, even if the majority experiences a network split, a majority vote is still required in order to mutate the global state, which is why the minority will not be able to make progress without completely leaving the original network.

2 Related work

Gossip-based consensus algorithms gained wide recognition thanks to cryptocurrency projects such as IOTA or Hashgraph [1]. MaidSafe’s Parsec [2] consensus also uses a gossip protocol as a base layer on top of which a Byzantine binary agreement protocol is used to construct a chain of blocks. Parsec makes weak synchrony assumptions to guarantee eventual message delivery. Wavelet [5] seems to follow Parsec on the use of Byzantine binary agreement to construct a chain of events from a gossip graph, in addition featuring a designated network management component responsible for keeping participant membership up to date, that is, adding or removing participants, and managing peer-to-peer connections between those.

Out of all the protocols listed above, Hashgraph has received the best coverage with regards to formal verification [4] and scrutiny of the implementation [8]. This has led to the underlying directed acyclic graph (DAG) data structure spreading over to other algorithms [2, 3]. However it should be noted that the choice of the same DAG data structure is not a requirement for a gossip consensus protocol. More specifically, even if a gossip protocol uses a DAG to store events and child-parent relations between those, the number of parents for non-genesis events does not have to be exactly two as in Hashgraph, and there can be various reasons to prefer a greater number of parents.

3 Model assumptions

Any protocol performs up to its theoretic assumptions. Protocols with stronger assumptions — such as assumptions of linear message ordering and eventual delivery — may be easier to define and prove statements about. When the assumptions don’t hold, the statements that rely on those assumptions don’t hold either and instances of the protocol may exhibit arbitrary behaviour. Therefore having strong assumptions may help in proving statements but those statements will be weak and would cover only a small number of cases.

Offscale’s consensus model assumptions are as weak as it is possible for stating protocol properties. This allows to capture real-world scenarios involving communication protocols with unreliable message delivery. In the model, I assume total asynchrony between participants: messages from a single sender can be reordered and an arbitrary number of messages can be dropped. Therefore a failed message sender is indistinguishable from a still operating message sender whose communication links are down. The only assumption is that a message cannot be received before being sent.

A required protocol property is Byzantine fault tolerance. For a consensus network of size N , I allow the maximum theoretically possible number of Byzantine participants t where $3t < N$.

The permissionless setting is assumed. Any participant is allowed to join or leave anytime, without requiring an authority for that. The security properties hold for all current participants in the consensus network. Participants are allowed to have no information about the exact number of participants in the network.

The protocol satisfies the *consistency property*: at any point in the execution, any two honest participants have consistent chains. That is, either both their chains are identical, or one is a prefix of the other.

Due to asynchrony, the protocol does not satisfy the *liveness property*. This property amounts to any honest protocol participant being able propose to add a transaction to the chain, and if this transaction then gets added to some honest participant’s chain, it is guaranteed to be eventually incorporated into the chains of all honest participants. Clearly the liveness property is unattainable in the presence of arbitrary — possibly infinite — message delays. Participants may stall and fail to progress if messages to or

from them are delayed or dropped. However this problem can be dealt with independently of the asynchronous consensus algorithm, for example, by designing a messaging layer that eliminates arbitrary message delays by detecting and reporting those as failures.

4 Basic notions

Honest participant: a consensus network participant adhering to the protocol.

Malicious participant: a consensus network participant exhibiting arbitrary, Byzantine behaviour.

Blockchain, or simply **chain**: a list of blocks. The chain is the replicated state of the protocol. All honest protocol participants agree on it in the sense of the consistency property, see Section 3.

Block: an event signed by a subset of network participants.

Gossip event, or simply **event**: a data structure containing information that a participant wants to pass to other participants.

Gossip graph: a graph data structure with nodes being gossip events and with edges being child-parent relations. This graph structure satisfies acyclicity: there is no path from any node leading to itself. In other words, a gossip graph is required to be a DAG and any operations on it are required to preserve this property.

Child-parent relation: a directed acyclic relation between nodes in the gossip graph connecting events to earlier events that led up to them.

Genesis event: a gossip event which is the unique root of the gossip graph. It is the same across all honest participants.

Famous event: a unique event in the gossip graph selected by honest participants according to a secure protocol which is protected from malicious participants, for example, using a verifiable delay function or Byzantine binary agreement.

5 Overview of the protocol

In the overview below I provide a conceptual presentation of the protocol with minimal details about how the protocol might be designed at the functional level. Detailed designs may vary widely in the degree of optimisations or sophistication.

A protocol instance, run by an honest participant, alternates between three tasks indefinitely, starting from a genesis event:

1. Add any pending blocks to the chain.
2. Add pending transactions if there are any as a new event in the gossip graph.
3. Synchronise the gossip graph with another participant chosen at random.

5.1 Construction of the blockchain

To construct blocks, the gossip graph is divided into sections called rounds, starting from round 0 that contains the unique genesis event E_0 with no parents. Every complete round n yields a unique block B_n that is added to the blockchain. Round n is complete when a new unique famous event E_{n+1} is computed by honest participants according to the protocol. That new unique famous event E_{n+1} marks the start of the next round $n + 1$. The new block is then formed from the transactions contained in the famous event E_n at the start of the current round and from gossip events that are both children of E_n and ancestors of E_{n+1} , not including the event E_{n+1} itself. Those transactions are linearly ordered locally by each honest participant using only the information in the events of those transactions and rules of the protocol that are the same for all participants.

5.2 Adding new transactions to the gossip graph

New transactions originating at a participant are included in new events in a manner that guarantees eventual linear ordering of transactions. A trivial example is having one event per transaction given that the new transactions are linearly ordered between themselves. Every new such event has the last event created by the participant as one of its parents.

5.3 Synchronisation of the gossip graph

Participants make and receive randomised synchronisation requests. Requests themselves are events in the gossip graph, as well as actions of receiving a request. A request receipt is related to the request by the child-parent relation. Creating a request receipt is followed by creating a synchronisation response event by the receiving participant, making the receipt event one of the parents of the synchronisation response event, and sending a gossip graph update to the sender of the request.

References

- [1] Leemon Baird. *The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance*. Swirlds Tech Report SWIRLDS-TR-2016-01. 2016.
- [2] Pierre Chevalier, Bartolomiej Kamiński, Fraser Hutchison, Qi Ma, Spandan Sharma. *Protocol for Asynchronous, Reliable, Secure and Efficient Consensus (PARSEC)*. 2018. <https://docs.maidsafe.net/Whitepapers/pdf/PARSEC.pdf>
- [3] Sang-Min Choi, Jiho Park, Quan Nguyen, Andre Cronje. *Fantom: A scalable framework for asynchronous distributed systems*. 8th February, 2019. https://github.com/SamuelMarks/consensus-rough-notes/blob/cd603414cf16526ea8e94ed341d9021eb8e0041f/papers/Fantom__A_scalable_framework_for_asynchronous_distributed_systems.pdf
- [4] Karl Crary. *Hashgraph consensus aBFT proof in Coq*. 23rd October, 2018. <https://swirlds.com/downloads/hashgraph-coq.zip>
- [5] Kenta Iwasaki, Heyang Zhou. *Wavelet: A decentralized, asynchronous, general-purpose proof-of-stake ledger that scales against powerful, adaptive adversaries*. 26th May, 2019. <https://wavelet.perlin.net/whitepaper.pdf>
- [6] Rafael Pass, Elaine Shi. *Rethinking Large-Scale Consensus*. 30th Computer Security Foundations Symposium (CSF), 2017. <https://ieeexplore.ieee.org/document/8049715>
- [7] Team Rocket. *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*. 16th May, 2018. <https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV>
- [8] Eric Wall. *Hedera Hashgraph – Time for some FUD*. 2nd September, 2019. <https://medium.com/@ercwl/hedera-hashgraph-time-for-some-fud-9e6653c11525>
- [9] Maxim Zakharov. *Consensus schema*. 20th September, 2019. <https://github.com/SamuelMarks/consensus-rough-notes/blob/356d9d2df8872c23e57f65154522979f522b068/Consensus-schema.md>