Mosaic
Running Meta-Blockchains to Scale Decentralised Applications

B. Bollen, P. Gevorgyan, S. Khedar,
D. Kumar Nath, M. Schenck*
for OpenST Foundation
working draft

OpenST Mosaic v0.10 - September 2018

Abstract

This is a working draft summary of notes and whiteboard sessions as we have been iterating towards the implementation for Mosaic v0.10. This draft is shared for review and comments by the community. This is not the version intended for publication, and you are invited to review critically all content. The code is work in progress at `github.com/openstfoundation/mosaic-contracts`.

1 Introduction

Mosaic is a parallelisation schema for decentralised applications. Mosaic introduces a set of Byzantine fault-tolerant (BFT) consensus rules to compose heterogeneous blockchain systems into one another. Decentralised application can use Mosaic to compute over a composed network of multiple blockchain systems in parallel.

A decentralised application is an application for which the computation is requested, performed, and paid for by independent actors. To ensure correct execution of decentralised computations first an order on the inputs of execution must be determinable. To date Ethereum is the leading networkof nodes that enables developers to write and deploy code which can be called by independent users to collectively construct a shared state of the application.

*corresponding author: martin@ost.com

The Ethereum network achieves agreement on the collective state by constructing a blockchain. A blockchain derives correctness from full replication of the computation by nodes and for Ethereum today the chain is appended through Proof-of-Work consensus.Proof-of-Work (PoW) introduces a block reward incentive for nodes to keep producing blocks and thereby securing the chain of historical blocks. However, PoW produces a serial execution model and nodes cannot be collectively rewarded for the computation they all have to perform, which renders the system computationally inefficient[1].

Active research and implementation work is ongoing to scale Ethereum's transaction throughput by dividing the verification work into multiple sections, also referred to as *shards*. At the same time Ethereum is committed to moving from the probabilistic Proof-of-Work consensus engine to a BFT based Proof-of-Stake (PoS) consensus engine.The outset of a BFT consensus engine is to collectively produce a blockchain which provably - under certain honesty assumptions - cannot finalise conflicting blocks.

Thus far a vision for a decentralised web has been a major driver of innovation. However, to power global networks of billions of users and decentralised computation on vast decentralised data stores, it is reasonable to assume no single blockchain network will suffice as a backbone for all types of applications. Much like the internet is a composed network of networks, we can conceive decentralised applications to transcend any single blockchain and execute across a network of blockchain networks.

In this work we detail two protocols. A first layer-2 protocol constructs *meta-blockchains* on top of existing blockchain networks to extend their state space and transaction throughput capacity. A second protocol, called a *gateway*, acts as a message bus to send typed messages atomically between the underlying blockchain and a meta-blockchain running on top of it.

Together these two building blocks can be used by decentralised applications to construct a parallelisation schema to increase computational capacity at lower transaction cost. In the simplest form a decentralised application can offload transaction processing to a single meta-blockchain. More advanced applications can deploy logic across multiple meta-blockchains and process transactions in parallel while asynchronous messages can be sent between meta-blockchains.

2 Composing auxiliary systems into an origin

A meta-blockchain composes transactions executed on an auxiliary blockchain system into an origin blockchain, such that the capacity of the origin blockchain can be extended and the security properties of the origin blockchain can be inherited by the auxiliary blockchain.

To this end consider two blockchain systems, an origin blockchain O with state transition rules t_O which progress the state Σ_O

$$t_O : \Sigma_O \to \Sigma'_O$$

and similarly consider an auxiliary system A with state transition rules t_A such that $t_A : \Sigma_A \to \Sigma'_A$.

Transition rules for the origin and auxiliary system can be similar but do not have to be. In our discussion, when we need an example, we will reference to the origin blockchain as Ethereum (Byzantium), running Proof-of-Work, and for the auxiliary system we consider Go-Ethereum, running "clique" Proof-of-Authority (PoA). We deliberately use PoA for our considerations for the auxiliary system to emphasise that the security for the composed auxiliary system is not derived from the consensus engine of the auxiliary system. Rather the security properties of the composed system must rely only on the security properties of the consensus rules of the origin blockchain and on the Mosaic BFT consensus rules composing the auxiliary system into the origin system[1].

On the origin blockchain we define a meta-blockchain $\mathcal{B}$ with a staked validator set $\mathcal{V}_\mathcal{B}$. The blocks B_i of $\mathcal{B}$ are committed to a *core* contract on origin O. For a given history of O[2] the meta-blockchain cannot fork if we enforce that block B_{n+1} can only be (proposed and) committed after block B_n has been committed.

We define a meta-block $B_i(K, T, S) : \Sigma_A \to \Sigma'_A$ to progress the state of the auxiliary system A, where we call K the *kernel*, T the *transition object* and S the *seal*. A block B_i at height i is committed with a seal S when a $+\frac{2}{3}$ supermajority of the weighted votes has been verified[3]. We will detail

[1]The block proposers of the auxiliary system do have the ability to censor transactions from the auxiliary system if they have an ability to collude.

[2]The origin blockchain O might be probabilistically finalised, in which case it is easy to observe that a transaction to a contract can always assert that it is valid only on the intended branch of history by referencing a historical blockhash.

[3]While a vote counted on a BFT system itself is already BFT for a normal majority, we still require a supermajority, because the votes to commit a meta-block will be collected on the auxiliary system for which there is no assumption it is BFT, and we want Mosaic to be able to finalise transactions on the auxiliary system before they are committed to origin.

later how the meta-block is constructed, but first we will describe the vote messages that together can combine to seal a meta-block.

2.1 Checkpoint finalisation - Casper FFG

The reader is assumed at this point to be familiar with the work presented in *Casper the Friendly Finality Gadget*[2] (FFG), as we will build on the logic and proofs presented there. We will intend to align with definitions and notations where possible and highlight deviations where appropriate. In this section we briefly (and incompletely) summarise core concepts from the paper, so that the shared concepts are established for the reader.

Casper FFG presents an algorithm to build up an overlay network of vote messages presented to a smart contract about the blocks already produced by the underlying network of nodes running the blockchain. The overlay network aims to repeatedly economically finalise a single branch of the underlying network. It does so by allowing staked validators to cast votes which together can construct a *justified chain*.

A justified chain is formed by a sequence of *supermajority links*, where a link identifies a *source* blockhash s and height h_s, and a *target* blockhash t and height h_t. A vote message in Casper FFG is a signed link $\langle s, t, h_s, h_t \rangle_v$ by a validator v. A $+\frac{2}{3}$ supermajority of the same link signed is a supermajority link which justifies the target block if the source block is itself already justified[4].

In the work it is shown that checkpoints along a justified chain can additionally be considered *finalised*, if and only if they are justified themselves and their direct child is justified. It is then shown that should validators finalise a checkpoint on a contradicting branch of the history of the underlying blockchain, then minimally $\frac{1}{3}$ of the validator weight must have signed vote messages that violate either one of two rules, *the Casper slashing conditions*.

These slashing conditions can be intrinsically validated given two signed vote messages from a validator v. As such a validator must never sign two vote messages $\langle s_1, t_1, h_{s_1}, h_{t_1} \rangle_v$ and $\langle s_2, t_2, h_{s_2}, h_{t_2} \rangle_v$ for which either $h_{t_1} = h_{t_2}$ or $h_{s_1} < h_{s_2} < h_{t_2} < h_{t_1}$ holds.

It is additionally shown that validators can always finalise a new checkpoint, without being required to violate the slashing conditions given that the block proposers propose blocks to append to the finalised fork, ie. follow the fork selection rule of the overlay network.

[4]The genesis block of the auxiliary blockchain is considered justified by definition.

2.2 Finalising auxiliary systems

Given a validator set $\mathcal{V}_\mathcal{B}$ staked for the meta-blockchain on origin, validators v can submit vote messages on the auxiliary system about the auxiliary system of the form

$$\langle \tilde{T}, s, t, h_s, h_t \rangle_v \tag{1}$$

where $\tilde{T}$ is the keccak256 hash of the transition object T. We introduce a transition object hash in the vote message to externalise otherwise intrinsic properties of the auxiliary system. This transition object allows the Mosaic validators to coarse-grain and abstract the auxiliary system into a meta-blockchain when proposing meta-blocks on origin.

We require that any transition object is calculable by a smart contract on the blockchain in question for any finalised checkpoint along a justified chain. For a given link, we define the transition object to refer to the state of the blockchain at source block height h_s. It then follows that for a given source blockhash s $\tilde{T}$ is uniquely determined by s and the same properties of accountable safety and plausible liveliness hold for a finalised checkpoint on a justified chain constructed by such *externalised vote messages* (1), if we trivially extend the slashing conditions to accommodate for the transition object hash.

A validator $v \in \mathcal{V}_\mathcal{B}$ must never sign two externalised vote messages $\langle \tilde{T}_1, s_1, t_1, h_{s_1}, h_{t_1} \rangle_v$ and $\langle \tilde{T}_2, s_2, t_2, h_{s_2}, h_{t_2} \rangle_v$ such that any of the following three conditions holds:

1. $h_{t_1} = h_{t_2}$,
2. $h_{s_1} < h_{s_2} < h_{t_2} < h_{t_1}$,
3. $s_1 = s_2 \wedge T_1 \neq T_2$.

As the slashing conditions can be intrinsically asserted to have been violated given two externalised vote messages by the same validator, there is no communication overhead to assert a possible violation of these conditions on the origin blockchain; even if the justified chain is being constructed on the auxiliary system. As a result the validators in $\mathcal{V}_\mathcal{B}$ can be held accountable on the origin blockchain for their voting actions on the auxiliary system. The slashing condition can be asserted on both systems and there is a clear incentive for the honest validators to assert any violation without delay on both the origin and auxiliary system, naturally synchronising the validator weights when such an event occurs.

We will later in this work address a dynamic validator set $\mathcal{V}_{\mathcal{B}}$, where validators can join and log out on the origin system. However, already with a static validator set, we can observe that the finality gadget first introduced for (re)defining economic finality in layer-2 of Ethereum's PoW - turning miners into mere block proposers and introducing a PoS (partial) consensus engine in the smart contract layer - can also be applied to finalise an independent (auxiliary) system with a validator set whose stake is held on an external (origin) blockchain.

2.3 Observing origin

Per construction the finalisation of the auxiliary system by the Mosaic validators economically binds the block proposers of the auxiliary system to the Mosaic fork selection. By finalising the auxiliary system the Mosaic validators reach consensus about the auxiliary system itself. However, to construct a meta-blockchain we will additionally require the Mosaic validators to reach consensus about their observation of the origin blockchain on the auxiliary system.

To achieve this the Mosaic validators can construct a justified chain and finalise checkpoints along it for reported blocks of the origin blockchain on the auxiliary system. The incentive structure is now reversed and the origin blockchain is in no way incentivised (nor should it be) to follow the fork selection rule of how a meta-blockchain's validator set finalised its observation of the origin blockchain.

In case the origin blockchain is probabilistically finalised, then it is always possible that the Mosaic validators of a given meta-blockchain running on top of the origin blockchain would economically finalise an observation of origin on the auxiliary system which is (later) reverted by the origin blockchain - even if they sufficiently trailed the head of origin. Note that the validators cannot revert their finalised observation, because they would be required to sign vote messages which would violate the slashing conditions.

Under this scenario we must force the meta-blockchain to halt at the highest finalised checkpoint of the auxiliary system which was still consistent with its observation of the (now reverted) history of the origin blockchain. This property can be enforced by including in the transition object of a justified chain of the auxiliary system T^A information about the finalisation of the observation of the origin blockchain T^O,

$$T^A_i = (f(T^O_j), \dots),$$

where $f(T^O)$ is the function that returns the highest finalised block number and blockhash of the origin system as observed on the auxiliary system.

The origin blockchain can inspect T^A to assert that the highest finalised observation of itself on the auxiliary system is within its current history. Should this not be the case, then the origin blockchain must reject meta-blocks containing contradictory observations.

However, origin cannot directly assert that prior to the last finalised observation, there was no checkpoint finalised from a contradictory branch of its history, as the nodes on origin should not fully verify the transactions executed on the auxiliary system. This is resolved by introducing an option to challenge a proposed finalised observation.

Assume an observed checkpoint a was finalised on a contradictory branch of origin relative to the last finalised observed checkpoint b included in T^A. Any honest node can challenge the finalisation of b on origin by presenting the finalisation of a and demonstrating that $a \notin \text{history}(b)$. Note that if validators would want to alter the finalised observation from $o \rightarrow a \rightarrow b$ to $o \rightarrow b$ they would have to produce vote messages violating the slashing conditions given the already existing vote messages.

2.4 Calculating the transition objects

In order to be able to verify the correct observation of auxiliary on origin, the transition object T^A is part of the meta-block. T^A is tracked on the auxiliary chain with every block and consists of the latest observed $f(T^O)$, the accumulated transaction root, the accumulated gas consumed, and the current dynasty number on auxiliary.

It also includes a constant *core identifier*, a 256-bit string where the first 12 bytes are a constant specifying the origin blockchain and the rightmost 20 bytes are determined by the smart contract address of the core contract on the origin blockchain. Rather than storing the core identifier as a constant it can be included in the signing string for vote messages to ensure vote messages are valid only about the intended meta-blockchain.

A smart contract calculates T^A for every block. Therefore, the validators have to report the details of every block header to the smart contract. If the reported block is within the most recent 256 blocks of the observed chain[5], then the smart contract can verify its correctness by accessing the corresponding block hash. If the validators fall behind more than 256 blocks in reporting, they can report more than one block per new block in order to catch up. The smart contract will record all reports, but only mark them as trustworthy if they build a chain that ends within the most recent 256

[5]This is specific to the Ethereum virtual machine, but the logic can be applied for other blockchain systems as well.

blocks.

Tracking of T^A begins at the genesis checkpoint. For the genesis checkpoint, the accumulated transaction root is defined as the transaction root of the block, the accumulated gas consumed equals the gas consumed in the block, and the current dynasty number is 0. For all subsequent blocks, the accumulated transaction root r_i^a at block height i is keccak256(r_{i-1}^a, r_i), where r_i is the transaction root of the block at height i. The accumulated gas consumed g_i^a at block height i is $g_{i-1}^a + g_i$, where g_i is the gas consumed in the block at height i. The dynasty number equals the number of finalised checkpoints in the chain from the root checkpoint to the parent block, carrying over the definition of dynasty number as in Casper FFG [2].

2.5 Proposing meta-blocks on origin

For a meta-blockchain $\mathcal{B}$ the chain can be appended by proposing and committing new meta-blocks $B_n(K_n, T_n^A, S_n)$ in the core contract on the origin blockchain. The genesis meta-block B_0 is considered committed.

The kernel K_n is a tuple $(n, p_n, \Delta\mathcal{V}_{\mathcal{B}_n}, gp_n)$ fully determined by and stored in the core contract on the origin blockchain. On a given branch of the origin blockchain a meta-blockchain cannot fork. The act of committing a meta-block $B_{n-1}(K_{n-1}, T_{n-1}^A, S_{n-1})$ activates the new kernel K_n at height n and *opens* the core contract to accept proposals of the form $B_n(K_n, \cdot, \cdot)$. The kernel further specifies the *parent hash* p_n, the *updated validator weights* $\Delta\mathcal{V}_{\mathcal{B}_n}$ to declare new validators joining and existing validators logging out, and a voted-upon *gas price* gp_n for the gas that will be consumed in the upcoming meta-block B_n as rewarded to the Mosaic validators[6].

Given a committed meta-block B_{n-1} at height $n-1$ a proposal for a meta-block $B_n(K_n, T_n^A, \cdot)$ at height n is a valid proposal if validity assertions for the transition object T_n^A hold. These validity assertions require that the dynasty number and the gas consumed are strictly increasing compared to the transition object committed T_{n-1}^A. It must be checked that the latest finalised observation of origin is within the history of the current state. Furthermore a brief challenge period exists where the transition object can be contested on the grounds that it contains a prior observation of origin that contradicts

[6] A meta-blockchain has a double gas market. First the known gas market exists where users set a gas price in the transaction that gets executed on the auxiliary blockchain and the gas rewards go to the block proposers of the auxiliary system. A second, new gas market is created by requiring the block proposers to pre-deposit gas rewards for several meta-blocks in advance. Mosaic validators are rewarded upon committing a meta-block on the origin blockchain for the total gas consumed in that meta-blockchain at the agreed-upon gas price gp_n for that meta-block B_n. Details in later section.

the current history of origin, as explained above in section (2.3).

It now follows that for a given active kernel K_n multiple proposals $T_n^{A(i)}$ can be submitted to the core contract.

2.6 Dynamic set of validators

The set of Mosaic validators needs to be able to change. A validator can join the set of validators by sending a deposit transaction to the stakes contract on origin. The amount of stake of the validator is based on the amount of value that the validator sends with the deposit transaction.

When a validator wants to leave the set of validators, it sends a logout message to the stake contract on origin. The validator's weight is reduced to zero, therefore the validator leaves the set of validators.

When a validator casts a vote on auxiliary, the vote carries a specific weight. The initial weight of a new validator is equal to the stake on origin. When we say that a vote receives a $\frac{2}{3}$ majority, we mean that at least $\frac{2}{3}$ of the total voting weight on auxiliary have cast that vote. A validator's stake may be reduced to a non-zero value, e.g. by slashing. However, in any slashing case the weight will always go to zero immediately.

We defend against long range revision attacks the same way casper does. Four months after the validator left the set of validators, the validator can withdraw its stake.

Casper FFG[2] defines a stitching mechanism for a changing set of validators. It prevents a case where a significant change in validators from one dynasty to the next could result in conflicting finalised checkpoints on different forks. They define a forward validator set and a rear validator set based on the start dynasty and end dynasty of a validator. In the case of Mosaic, however, validators do not join and leave the set of validators within two dynasties, but rather two meta-blocks. When a validator sends a deposit or logout message at meta-block height i, the change is announced to auxiliary with the opening of meta-block $i + 1$. The auxiliary system will update the validator's weight with the opening of meta-block $i + 2$ and the validator will join or leave the set of validators. We call that meta-block the start meta-block or end meta-block. Following this change, the forward and rear validator sets are based on the start meta-block and end meta-block, respectively, instead of the start dynasty and the end dynasty.

In the same way, an ordered pair of checkpoints (s, t), where t is in meta-block B_i, has a supermajority link if both at least $\frac{2}{3}$ of the forward validator set and the rear validator set of B_i, respectively, have published vote $\langle \tilde{T}, s, t, h_s, h_t \rangle_v$.

The same public key can only deposit and withdraw once. It cannot join the set of validators multiple times or change its stake.

NOTE: weight = stake x reputation

When a meta-block has been committed on origin, the Kernel of the newly opening meta-block can be confirmed on auxiliary. The state root of origin is regularly transferred to auxiliary and finalised. By means of a Merkle proof against that known state root, the content of the Kernel can be made available to auxiliary. Auxiliary will note the update of the validator weights for the meta-block height of the opened meta-block plus one. The validator set changes on auxiliary at the time the kernel opening is confirmed. Even though the kernel opening retroactively opens the meta-block on auxiliary from the first block after the last finalised checkpoint in the meta-block that precedes the opening one, the validator set only changes at the point of the confirmation. When the Kernel at height i is confirmed on auxiliary, the validator set $i + 1$ becomes the active set of validators. The validators that logged out on origin before the opening of K_i will be in the rear validator set of K_{i+1} and leave the validator set when K_{i+2} is confirmed. This way the Casper stitching of dynamic validators works.

3 A mosaic of cores

4 Gateway

5 Outlook

6 Conclusion

7 Appendix

References

[1] Luu, L., Teutsch, J., Kulkarni, R. & Saxena, P. Demystifying incentives in the consensus computer. *IACR Cryptology ePrint Archive* **2015**, 702 (2015).

[2] Buterin, V. & Griffith, V. Casper the friendly finality gadget. *CoRR* **abs/1710.09437** (2017). URL `http://arxiv.org/abs/1710.09437`. `1710.09437`.