Mosaic

RUNNING META-BLOCKCHAINS TO SCALE DAPPS

Benjamin Bollen, Martin Schenck
for OpenST Foundation
working draft

Abstract

Mosaic is a set of Byzantine fault tolerant (BFT) meta-blockchains defined in smart contracts on Ethereum. A meta-blockchain has an open set of validators staked on Ethereum, and has a smart-contract based consensus engine. Each block is committed by a super-majority vote. Mosaic introduces BFT consensus rules that allow validators to leverage more efficient auxiliary systems running in parallel for block production, while millions of transactions can be finalised before the header is asynchronously committed to Ethereum. Ethereum remains the sole source of truth by locking value tokens on Ethereum while minting a utility token representation on the auxiliary systems. Validators get rewarded collaboratively based on gas consumed thereby addressing the verifiers dilemma that affects proof-of-work. As a result Mosaic can finalise thousands of transactions per second on Ethereum at a lower gas price. We have built an atomic gateway protocol that Ethereum DApps can deploy to leverage Mosaic to scale.

1. Introduction

Both Ethereum and Bitcoin have established over several years that "proof-of-work" (PoW) blockchains provide robust security and liveliness properties, albeit at a high transaction cost and with limited transaction throughput capacity. It has been understood that for PoW mining to be secure, it must be inefficient as formulated by the Verifiers dilemma. This follows from the reward structure where a miner is rewarded for mining a block, but miners are not rewarded for verifying transactions within a mined block.

While PoW mining is far from perfect (and in the long term unsustainable) it currently supports Ethereum to repeatedly reach global consensus about the state of a shared ledger of accounts and their storage. It additionally has spurred work on how to replace PoW with more efficient solutions to reach global consensus on an open network. A significant amount of this work is captured under the category of "proof-of-stake" (PoS) based blockchain technology such that security is not derived from expended energy, but rather on the amount of (self-referential) stake an agent has in the system to influence the production and commitment of new blocks.

In this paper we set out to build on the existing work of Byzantine fault-tolerant (BFT) based proof-of-stake blockchains. We present Mosaic as a layer 2 protocol to extend the transaction throughput capacity of the underlying, *origin* blockchain. Mosaic allows to preserve the native interface of the origin blockchain. In the case of Ethereum, this means Mosaic enables smart contract developers to write code for the same Ethereum Virtual Machine (EVM) interface while benefitting from an increased transaction throughput from adopting Mosaic.

[continue]

2. A Mosaic of Cores

A blockchain relies on each full node replicating all transactions to obtain and verify independently the correctness of the current state. Full replication at the same time impedes scalability.

Mosaic sets out to leverage the security and liveliness properties of the origin blockchain. Mosaic allows to create a new state space and compose it into the origin state space so that it suffices to verify the origin state space and the extended state space the node is

Mosaic is a framework to increase the transaction throughput of an origin blockchain (e.g. Ethereum) by finalising transactions off of the origin blockchain. Finalised transactions can be committed asynchronously in bulk onto the origin blockchain.

3. The Dry Stuff

3.1. Origin

the origin blockchain is the assumed BFT blockchain for upon which Mosaic cores are deployed to extend the origin state space.

3.2. Mosaic Consensus Rules

Build on Casper FFG.

Validator can sign a vote message

$$\langle T, s, t, h_s, h_t \rangle_v$$

slashing conditions for a validator :

1. $h_t \neq h_{t'}$
2. inclusion not allowed $h_{s_1} < h_{s_2} < h(t_2) < h_(t_1)$
3. $s_1 = s_2 \wedge T_1 \neq T_2$

3.3. Stake

Stake defines the validator list and their voting weight derived from their deposited stake.

3.4. Core

Core is a meta-blockchain defined on origin. On core a meta-block can be proposed and committed. A meta-blockchain cannot fork (assuming the origin blockchain has not forked).

3.5. Meta-Block

A meta-block defines a block transition for the core c

$$B : S_c \rightarrow S'_c$$

and a meta-block is composed out of three components. A meta-block has a `kernel` K, a `transition` T and a `seal` . The kernel is determined by the `core` contract on the origin chain. The transition is generated in the finalisation process by the validators on the auxiliary chain. Lastly the seal is generated by the core contract on origin as the validators verify their vote messages. The meta-block's hash is derived from the kernel and the transition (K, T).

The kernel K has five fields. `height` counts the number of committed parent meta-blocks. The genesis meta-block has height zero. `parentHash` refers to the meta-block hash of its parent. The kernel further specifies the `gasPrice` the proposers are willing to pay the validators for committing the meta-block. Lastly the kernel contains an update-insert list of validators and their updated weights. Validators who's weight is set to zero are removed from the validator list.

3.6. Block Store

Block store is a representation of a blockchain on-chain.

3.7. Polling Place

Polling place tracks the weight of the validator set and collects vote messages about links between checkpoints.

4. Gateway

4.1. MessageBus

Message bus allows abstraction of logic to pass a message from bidirectionally between origin and auxiliary. Declaration, confirmation and progression and optional reverting of messages.

4.2. ERC20-Typed

4.3. Kernel-Typed

5. Running Origin and Auxiliary Chains

We assume without loss of generality that for every validator-address v^O in the validator set $\mathcal{V}^O$ on origin a corresponding, unique validator-address v^A on auxiliary A can be associated and we call the resulting validator set $\mathcal{V}^A$, the validator set on A. A single actor is assumed to control and be responsible for both v^O and the associated v^A. On origin v^O will always be held accountable for any signed message by v^A on A, because and only v^O has stake on origin.

Lemma 1 (Remote Accountability)**.** *A vote on A that violates a Casper Commandment can be punished on O.*

Proof. Given lemma **??**, a violation of either Casper Commandment can be proven on O without the respective transaction residing on O. If a validator ν^A published two distinct votes for the same target height, the relevant votes can be presented on O and the signature can be validated. An offending validator ν^O can be identified according to lemma **??** and its deposit slashed. The same is true if a validator ν^A votes within the span of its other votes. All relevant votes including their signatures can be presented on O. □

Theorem 2 (Partial View)**.** *Given Origin O with blocks b_i^O, checkpoints c_i^O can be reported on Auxiliary A, such that a set of validators $\mathcal{V}^O$ with stake on O can reach finality about the reported checkpoints c_i^O on A as $\mathcal{V}^A$ with accountable safety enforced on O.*

Proof. If you report checkpoints from Origin as $\langle h_c, h(c)\rangle$ where h_c is the hash of any checkpoint c on Origin and $h(c)$ is the height of checkpoint c in Origin's checkpoint tree, then you can vote on Origin's checkpoints on Auxiliary according to the Casper FFG voting rules. A Casper FFG vote consists of two checkpoint hashes, two checkpoint heights, and the validator signature, all of which is now available on Auxiliary.

Accountable safety is enforced according to lemma 1. □

Theorem 3 (Leveraged Security)**.** *Given that the auxiliary chain A follows the Casper fork choice rule and given that the validators staked on Origin O, Casper can be applied to A as a whole; without validators staking on A. It is not necessary to report blocks or block headers from A to O.*

Proof. Validators staked on Origin

Lemma 1 proves that the validators can be slashed on Origin for their actions on Auxiliary. Even though Auxiliary is not necessarily lively, Casper still has plausible liveliness, as Casper assumes that the underlying chain keeps producing blocks. □

Reporting a finalised auxiliary checkpoint to Origin costs gas on Origin. Therefore, validators do not have an incentive to copy the state root from Auxiliary to Origin. We present an asynchronous mechanism that forces validators to copy the state root. We define the amount of consumed gas since the last copied checkpoint g. We define a limit of consumed gas l. The finalised auxiliary checkpoint with the smallest dynasty where $g \geq l$ must be copied to Origin.

$$\neg\exists f : h(f') < h(f) \wedge g(f') \geq l \tag{1}$$

If the validators fail to copy that finalised checkpoint, they are still incentivised to copy a later checkpoint to get rewarded (see below). When a validator copies a state root of a later finalised checkpoint from Auxiliary to Origin, then it can be proven that there exists a finalised checkpoint with a lesser height that is also above the gas limit. To do that, a fisherman could present the earlier finalised checkpoint to Origin after the later one was committed. The height, consumed gas, and validator signatures can be proven. When that happens, all validators in the validator set are punished and the fisherman is rewarded.

We must track the consumed gas on Auxiliary in order to for this proof to work.

Theorem 4 (Honesty Guard)**.** *An honest validator will not be punished by the gas limit rules.*

Proof. All validators, including the honest ones, would be punished if the lowest finalised auxiliary checkpoint above the gas limit were not reported to Origin. If there is at least one honest validator, however, that validator would report the auxiliary checkpoint to Origin. Thus, all validators get only punished if none of them is honest. □

5.1. Gateway

5.2. Reward Structure

5.3. Dynamic Set of Validators

6. Outlook

7. Conclusion

8. Appendix

8.1. Origin and Core Identifiers

Each Origin chain and each Core have a unique identifier such that vote messages can associated with a specific core or origin chain.

References