Mosaic
RUNNING META-BLOCKCHAINS TO SCALE DECENTRALIZED APPLICATIONS

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

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 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 decentralized applications. Mosaic introduces a set of Byzantine fault-tolerant (BFT) consensus rules to compose heterogeneous blockchain systems into one another. Decentralized applications can use Mosaic to compute over a composed network of multiple blockchain systems in parallel.

A decentralized application is an application for which the computation is requested, performed, and paid for by independent actors. To ensure correct execution of decentralized computations first an order on the inputs of execution must be determinable. To date, Ethereum is the leading network of 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.

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[1]. The outset of a BFT consensus engine is to collectively produce a blockchain which provably — under certain honesty assumptions — cannot finalize conflicting blocks.

Thus far a vision for a decentralized web has been a major driver of innovation. However, to power global networks of billions of users and decentralized computation on vast decentralized 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 decentralized applications to transcend any single blockchain and execute across a network of blockchain networks.

We detail two protocols in this work. The first, a layer-2 protocol, constructs *meta-blockchains* on top of existing blockchain networks to extend their state space and transaction throughput capacity. The second

---

[1]`https://github.com/ethereum/eth2.0-specs/blob/master/specs/casper_sharding_v2.1.md`

protocol, called a *gateway*, acts as a message bus to send typed messages atomically between the underlying layer-1 blockchain and the meta-blockchain running on top of it.

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

2. Composing Meta-Blockchains

A meta-blockchain composes transactions executed on an auxiliary blockchain system into an origin blockchain, such that the capacity of the origin blockchain extends and the auxiliary blockchain inherits the security properties of the origin blockchain.

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

$$t_O : \Sigma_O \to \Sigma'_O \tag{1}$$

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 refer 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 emphasize 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.[2]

We define a meta-blockchain $\mathcal{B}$ with a staked validator set $\mathcal{V}_\mathcal{B}$ on the origin blockchain. The blocks B_i of $\mathcal{B}$ are committed to a *core* contract on origin O. For a given history of O,[3] 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 of the meta-blockchain is committed with a seal S when a $+2/3$ supermajority of the weighted votes has been verified.[4] We will detail later how the meta-block is constructed, but first we will describe the vote messages which, combined, 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. This section briefly (and incompletely) summarizes 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 finalize a single branch of the underlying network. It does so by allowing staked validators to cast votes which together can construct a *justified chain*.

---

[2]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.

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

[4]While a vote counted on a BFT system itself is already BFT for a simple 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 finalize transactions on the auxiliary system before they are committed to origin.

A justified chain is formed by a sequence of *supermajority links*, where a link identifies a *source* blockhash s and its height h_s and a *target* blockhash t and its height h_t. A vote message in Casper FFG is a link $\langle s, t, h_s, h_t \rangle_v$ signed by a validator v. A $+2/3$ supermajority signing of the same link makes it a supermajority link which justifies the target block if the source block is itself already justified.[5]

In the work it is shown that checkpoints along a justified chain can additionally be considered *finalized*, if and only if they are justified themselves and their direct child checkpoint is justified. It is then shown that, should validators finalize a checkpoint on a contradicting branch of the history of the underlying blockchain, minimally $1/3$ 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 finalize a new checkpoint, without being required to violate the slashing conditions, given that the block proposers propose blocks to append to the finalized fork, i.e. follow the fork selection rule of the overlay network.

2.2. Finalizing Auxiliary Systems

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

$$\langle \tilde{T}, s, t, h_s, h_t \rangle_v \tag{2}$$

where $\tilde{T}$ is the keccak256 hash of the transition object T.

We introduce a transition object hash in the vote message to externalize 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 finalized 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 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 finalized checkpoint on a justified chain constructed by such *externalized vote messages* (2), if we extend the slashing conditions to accommodate for the transition object hash.

A validator $v \in \mathcal{V}_\mathcal{B}$ must never sign two externalized 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 externalized 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 finalize an independent (auxiliary) system with a validator set whose stake is held on an external (origin) blockchain.

---

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

2.3. Observing Origin

Per construction, the finalization of the auxiliary system by the Mosaic validators economically binds the block proposers of the auxiliary system to the Mosaic fork selection. By finalizing 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. This is required so that message can pass bidirectionally between the chains.

To achieve it, the Mosaic validators construct a justified chain and finalize 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 incentivized (nor should it be) to follow the fork selection rule of how a meta-blockchain's validator set finalized its observation of the origin blockchain.

In case the origin blockchain is probabilistically finalized, then it is always possible that the Mosaic validators of a given meta-blockchain running on top of the origin blockchain would economically finalize 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 finalized 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 finalized 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 finalization of the observation of the origin blockchain T^O,

$$T_i^A = (O_f^j, \dots),$$

where $O_f^j = f(T^O)$ is the function that returns the highest finalized block number j and blockhash of the origin system as observed by the Mosaic validators for this meta-blockchain.

The origin blockchain can inspect T^A to assert that the highest finalized 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 finalized observation, there was no checkpoint finalized 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 finalized observation.

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

Upon success, the challenger is rewarded from the stake of the offending validators and the core contract must declare the meta-blockchain halted.

2.4. Calculating the Transition Objects

We construct a transition object T^A to coarse-grain a vast amount of transactions processed on the auxiliary system under the state transition rules t_A into a single, abstracted state transition that can be validated by the core contract on the origin blockchain (under the state transition rules of the origin blockchain t_O, eq. 1).

On the auxiliary chain with every block running parameters are tracked and these consist of the latest finalized observation of the origin blockchain $O_f^j = f(T^O)$, the *accumulated transaction root* r_i, the *accumulated gas* g_i consumed, and the current *dynasty number* d on auxiliary. The transition object T^A, however, is updated at every justified checkpoint s with height h_s and we write for a given meta-block height n:

$$T_{n,d}^A(s) = (d, O_f^{j_d}, r_d, g_d, \tilde{K}_n)$$

where $\tilde{K}_n$ is the keccak256 hash of the kernel K_n.

A smart contract calculates the parameters that go into T^A for every block. Therefore, the validators have to report the block header of every block to the smart contract. If the reported block is within the most recent

256 blocks of the auxiliary blockchain,[6] 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 valid if they build a chain that reconnects to a block hash within the most recent 256 blocks of the current branch.

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 at block height i is keccak256(r_{i-1}, R_i), where R_i is the transaction root of the block at height i. The accumulated gas consumed g_i at block height i is $g_{i-1} + G_i$, where G_i is the gas consumed in the block at height i. The dynasty number equals the number of finalized checkpoints in the chain from the root checkpoint to the parent block, carrying over the definition of dynasty number as in Casper FFG [2].

Further, we introduce 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 in the transaction object, it can be included as a constant in the signing string for vote messages to ensure vote messages are valid only about the intended meta-blockchain.

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 per definition.

The kernel K_n is a tuple $(n, \tilde{p}_n, \Delta\mathcal{V}_{\mathcal{B}_n}, gt_n)$ 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* $\tilde{p}_n$, the *updated validator weights* $\Delta\mathcal{V}_{\mathcal{B}_n}$ to declare new validators joining and existing validators logging out.

gas target Lastly gt_n is the *gas target* which introduces a forcing function on the total amount of gas that can be consumed in the meta-block B_n. The earliest finalized checkpoint at which the gas target has been consumed by the meta-block must be committed to the origin blockchain by the validators. We highlight that this is not a maximum gas limit, because it cannot be guaranteed that a checkpoint can be finalized before a hard gas limit would be surpassed. Rather validators would gradually lose part of their stake as a function of the number of finalized checkpoints they failed to commit after the gas target has been surpassed.

To show that validators can be held accountable, assume the finalized checkpoint s_d has been committed which surpassed the gas target and has dynasty number d. Now assume there exists a finalized checkpoint $s_{d'}$ which also has surpassed the gas target but has a lower dynasty number $d' < d$, but was not committed before s_d. Any honest node can present the $s_{d'}$ and the core contract can assert the gas target had been consumed at a lower dynasty number than what is committed and reward the challenger.

committing a meta-block 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. The transition object T_n^A must reference the correct kernel $\tilde{K}_n$. It must be checked that the latest finalized 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 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 $T_{n,d}^A$ can be submitted to the core contract for different dynasty numbers d. These proposals are no longer contradicting versions of the history, rather they are ordered, sequenced snapshots of possible state transitions $B_{n,d_n}(\Sigma_{A,d_{n-1}})$ that can be committed onto origin as a new meta-block.

To commit a meta-block then requires selecting among the valid proposals a dynasty number d_n for which the meta-block will be closed. This selection occurs automatically by sequentially verifying the vote messages

---

[6]This is specific to the Ethereum virtual machine, but the logic can be applied for other blockchain systems as well.
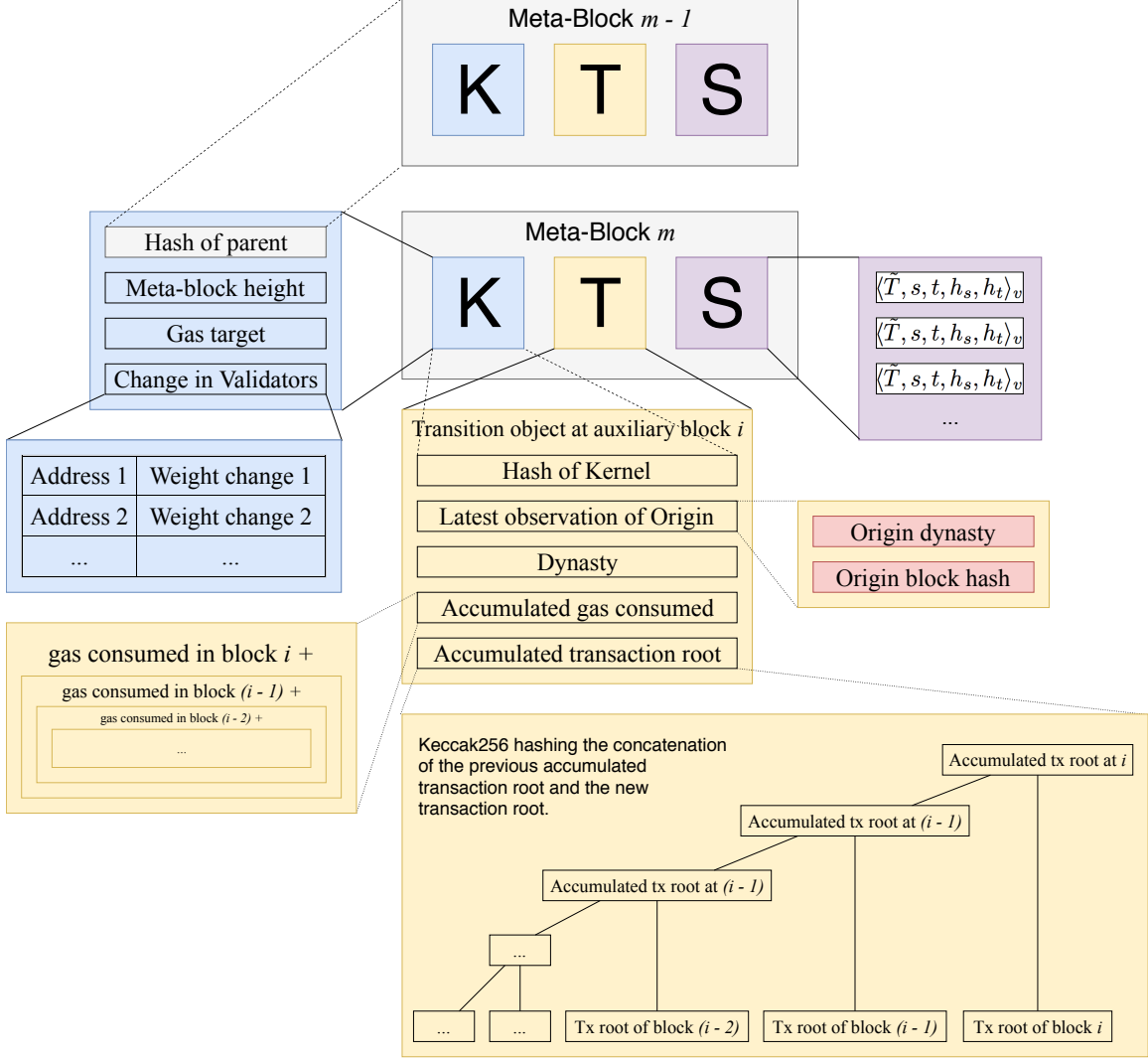
Figure 1: **Anatomy of a meta-block.** The meta-block consists of a kernel, a transition object, and a seal.

on the origin blockchain. The meta-block $B_n(K_n, T_n^A, S_n)$ will be committed with the seal S_n which first asserts a $+2/3$ supermajority of the voting weight $\mathcal{V}_{\mathcal{B}_n}$ for a valid proposal in the set of valid proposals

$$\{S_{n,d}\} = \left\{ \left\{ \langle \tilde{T}_{n,d}^A, s, t, h_s, h_t \rangle_v \right\} \right\}$$

where it is required that the supermajority link finalizes the source s. As such we require that $h_t - h_s = 1$ and the source s must have been justified.

Following the same reasoning as presented in section (2.3) we can assert whether or not the finalized checkpoint s was obtained through an unbroken justified chain leading up to the finalized checkpoint, by allowing for a challenge[7] to be raised against a proposed transition object T^A included in a vote message $\langle \tilde{T}^A, s, t, h_s, h_t \rangle_v$.

However, recall that in the case of verifying observations of origin (2.3), origin itself can act as an arbiter because origin can rely on its own history. Our concern now is to assert that the state transition to s from the last committed state B_{n-1} is a valid state transition according to the state transition rules t_A of the auxiliary system.

proof Assume first a contradictory finalisation s' of the auxiliary system relative to s exists. This implies that more than $1/3$ of the validator weight can be slashed. The core contract on origin must declare the

---

[7]A challenger must always put forward a sufficient stake to underwrite the cost implied by her challenge.

meta-blockchain halted when it can slash more than the safety threshold of $\mathcal{V}_{\mathcal{B}_i}$ at a single meta-blockchain height i.[8]

Alternatively, no contradictory finalisation s' of the auxiliary system relative to s exists. Under the security model assuming a $+2/3$ supermajority of the validator weight is honest, it follows that the proposed state transition to s has been obtained through honest verification of all transactions under the state transition rules t_A.

Before continuing we summarize that the proposal mechanism to commit a meta-block is asynchronous, requires no leadership-selection and is proven to be Byzantine fault-tolerant under the security model of an honest supermajority.

2.6. Beyond An Honest Supermajority

While the origin blockchain protects assets from being double-spent, users in a meta-blockchain could see their funds stolen if more than a supermajority of the open validator set would collude to violate the transition rules t_A.

We identify at least three active research topics which can allow to push the security model radically further in an efficient way; namely that (up to) a *single* honest observer can hold all validators accountable when a proposal for committing a meta-block to the origin blockchain would violate the state transition rules t_A. It is outside the scope of the current work to explore in-depth, but they are worth summarising below as future work paths.

observer network   In *A Guide to 99% Fault Tolerant Consensus*[3] V. Buterin recaptures an existing algorithm by L. Lamport (1982) to describe how a *latency-dependent* network of observers can be retrofitted on top of a *threshold-dependent* consensus algorithm - in casu our here work as derived from Casper FFG - to introduce a *strong-finality* upon the finalized history constructed by the Mosaic validators arbitrarily pushing up the fault-tolerance at the cost of requiring more observers and a longer time to eventually commit (a strongly finalized proposal).

truebit   A second worthwhile path is to apply the *TrueBit*[4] protocol to challenge a state transition T^A proposed by a subset of the validators $\mathcal{V}_{\mathcal{B}_i}$. The validators $\{v\}$ who have signed vote messages $\langle \tilde{T}^A, s, t, h_s, h_t \rangle_v$ can be seen as the (collective) *solvers* of the off-(of-origin-)chain computation to transition the state B_{n-1} to the proposed solution s given the set of transactions specified by the accumulated transaction root r_{h_s}. The computation at hand is the execution of the virtual machine that implements the state transition rules t_A.

If less than $1/3$ of the validator weight loses the verification game to a challenger, their stake can be taken as the reward for the challenger, on top of the shared interest of all participants in the meta-blockchain for only correct proposals to be committed to the origin chain - removing the need for a jackpot and a *forced-error*. The verification game can be seen as an extension to the slashing conditions strengthening the accountable safety of the meta-blockchain to hold a minority of the validators accountable for violating the state transition rules t_A. This is made possible because the validator stake has been externalized on the origin blockchain with an independent (assumed correct) consensus mechanism.

If more than $2/3$ of the validator weight signed the vote message which was challenged and lost the verification game, their stake is the reward for the challenger, and the meta-blockchain must be declared halted by the core contract.

argument of knowledge   A third strategy can be constructed by requiring any of the challenged validators of a proposal T^A to present an *argument of knowledge*, eg. using zkSNARKs generated after being challenged, to cryptographically prove that the block s has been obtained by correctly applying the state transition rules t_A recursively on each parent block building up from the committed block found in B_{n-1}.

The costs both in time and expense to generate this proof by a validator scale at first approximation linearly with the gas consumed in the proposed meta-block B_n. The transition object T^A to commit this meta-block tracks the gas consumed, so the core contract can require that the challenger puts forward a sufficient stake to substantiate her claim.

---

[8]When a meta-blockchain has been halted, assets and stateful objects can be recovered on the origin blockchain at any later time with Merkle proofs against the last committed state root of the meta-blockchain. We will detail the conditions under which a meta-blockchain must halt and how the recovery processes can work later in this writing.

There are open considerations regarding the (economic) feasibility of requiring all validators to be able to generate such a proof on commodity hardware. The task of generating these proofs could be designated to a specialized node, but an incentive problem arises when it is economically unattractive to run such a node considering it is unlikely that a proposal would be challenged.

We have gone beyond the assumption of an honest supermajority of validators by considering three strategies discussed in the field to radically improve the security model; up to the point where a single honest observer can cost-effectively challenge the validator set on the origin blockchain.

2.7. Dynamic Set of Validators

The set of validators for the meta-blockchain $\mathcal{B}$ must be dynamic to allow new validators to join and for validators to safely exit from their responsibilities. It is defined for each meta-block height n and we note $\mathcal{V}_{\mathcal{B}_n}$.

A validator v is a defined as an address which has a strictly-positive validator weight w_v associated with it on the origin blockchain. A validator can join the set of validators by sending a *deposit message* to the origin blockchain and putting forward a stake. A validator can initiate log out by sending a *logout message* to origin.

The weight of a validator w_v at meta-block height n is the product of the stake of the validator at that height $s_v(n)$ multiplied with a *reputation function $R(n_0, n; v)$*

$$w_v(n) = s_v(n) \cdot R(n_0, n; v) \tag{3}$$

where the reputation function R has values in the range $[0, 1]$ and n_0 is the meta-block height where the validator v sent the deposit message.

The reputation function allows for different security models to mitigate collusion attempts of malicious actors within the validator set of the meta-blockchain. As examples, the reputation function can require initial work by the validator before obtaining a non-zero reputation. The reputation function can also enforce a finite (probabilistic) lifetime for all validators, honest and dishonest alike. Inducing churn in the (honest) validator set adds an important stochastic tool to prevent collusion of more than $1/3$ of the validator weight.

Changes to the validator weights $w_v(n)$ are included in the kernel K_n as $\Delta\mathcal{V}_{\mathcal{B}_n}$. When tallying votes both the origin blockchain and the auxiliary system must use the *effective weight $W_v(n)$*,

$$W_v(n) = w_v(n) \cdot \theta_v$$

where θ_v is initially 1 for all validators, but irreversibly and immediately set to 0 when a validator v is shown to have violated any slashing condition. The effective weight allows for the voting power of an offending validator to be revoked on both blockchains instantaneously without waiting for the meta-block to be committed.

The stake $s_v(n)$ of a validator can only be preserved or decrease for increasing n when the validator does not fulfill its responsibilities. The *loss function L* recalculates the stake of a validator for the new meta-block height

$$s_v(n) = L(s_v(n-1); v). \tag{4}$$

The loss function reduces the actual value the validator can possibly withdraw in the future. The loss function can reward challengers for presenting proofs of bad behaviour by the validator from his stake and it can burn the remaining reduction.

When a validator violates any slashing condition the loss function returns the validator stake to zero.

Validators can see their stake reduced when they become inactive in the voting process (*inactivity leak*[9]), or when they fail to commit a meta-block back to origin according to the gas target requirements (section 2.5).

When a validator successfully logs out the reputation is set to zero, but the stake remains locked in the contract for a fixed amount of time to protect against long range revision attacks.[10] After this time the validator can withdraw his stake (and remaining rewards).

---

[9]The inactivity leak forces validators to participate in the voting process and be available. The stake of inactive validators can gradually be diluted if they fail to verify their vote on a committed meta-block. We credit the term to Casper FFG[2] for the original argument.

[10]By holding on to the stake the validator can still be held accountable for evidence of wrong doing that may surface later or new attempts of revising the history. We follow Casper FFG's proposal of approximating four months delay before withdrawal is allowed.
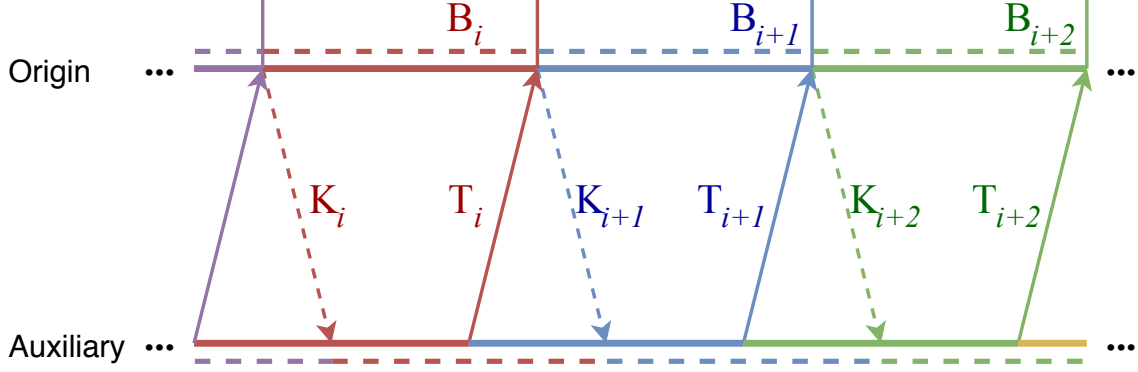
Figure 2: **Asynchronous composition.** When a proposed transition object T_i is committed as meta-block B_i on origin (solid red line), the new kernel K_{i+1} is opened on origin. The validator set $\mathcal{V}_{\mathcal{B}_i}$ casts votes on the auxiliary system to finalise both the checkpoints of the auxiliary system and its observation of the origin blockchain on the auxiliary system (dashed red line on aux). The commit of B_i and opening of K_{i+1} is observed by $\mathcal{V}_{\mathcal{B}_i}$ and K_{i+1} is confirmed on the auxiliary system transitioning to the new validator set $\mathcal{V}_{\mathcal{B}_{i+1}}$ (dashed blue line on aux). On the auxiliary system all processes continue uninterrupted. At each confirmation of a new kernel K_j the changes to the validator set $\Delta\mathcal{V}_{\mathcal{B}_j}$, and the new gas target gt_j are enacted on the auxiliary system.

asynchronous composition of blockchains   We refresh for the reader (section 2.5) that a meta-block B_i is committed on origin with a valid[11] transition object $T_i^A(\tilde{K}_i, \cdots)$ for which a seal S_i was created on origin by verifying a supermajority of the voting weight $\mathcal{V}_{\mathcal{B}_i}$ on vote messages of the form $\langle \tilde{T}_i^A, s, t, h_s, h_t \rangle_v$. As the hash of the kernel $\tilde{K}_i$ is included in the transition object, and as the hash of the transition object $\tilde{T}_i^A$ is included in the vote messages, the order to construct a meta-block $B_i(K_i, T_i^A, S_i)$ is logically enforced.

When a proposed transition object T_i^A is committed as meta-block $B_i(K_i T_i^A, S_i)$ then the state root of the auxiliary system is immutably anchored into the origin blockchain under the blockhash s[12]. It follows that the state of the meta-blockchain is first knowable on the auxiliary system and causally passed to the origin blockchain upon committing B_i (see solid lines on figure 2).

A new kernel K_{i+1} is defined by the committed meta-block B_i, the deposit, logout and slashing messages[13] that were included on origin during height i[14] and a new gas target gt_{i+1} for closing B_{i+1}. These messages combined with the recalculation of the validator weights (eq. 3) update the validator set

$$\mathcal{V}_{\mathcal{B}_{i+1}} = \Delta\mathcal{V}_{\mathcal{B}_{i+1}}(\mathcal{V}_{\mathcal{B}_i}) \tag{5}$$

which is included in K_{i+1} as the operator $\Delta\mathcal{V}_{\mathcal{B}_{i+1}}$.

The validator set $\mathcal{V}_{\mathcal{B}_i}$ is activated on the auxiliary system when the kernel K_i is *confirmed* on the auxiliary system. We define the confirmation of a kernel to require two conditions to be met. First, a Merkle proof that K_i exists on the origin blockchain must be presented against a finalised observation of (the state of) the origin blockchain as finalised by the validators set $\mathcal{V}_{\mathcal{B}_{i-1}}$. Second, if this Merkle proof of the kernel K_i was included on the auxiliary system at dynasty d then we consider K_i confirmed when this branch of history reaches dynasty $d+2$. The dashed lines on figure 2 indicate the activity period of $\mathcal{V}_{\mathcal{B}_i}$ and start at opening of K_i on origin and the confirmation of K_i on the auxiliary system.

---

[11] A valid proposal is a transition object T^A which satisfies the validity checks and has been unchallenged, or has been successfully proven upon being challenged.

[12] We assume here without loss of generality that the blockheader of the auxiliary system includes the state root of the system at height h_s and that the blockhash is the hash of an encoding of the blockheader.

[13] Slashing messages can concern both violations of slashing conditions reducing the stake of a validator to zero, or a proof of inactivity gradually reducing the validator stake.

[14] We say a transaction on origin is included during height i if it is included in the history of origin before B_i is committed and after B_{i-1} was committed.

9

For the auxiliary system all other processes continue uninterrupted, such as the calculation of accumulated gas consumed and the accumulated transaction root. At each confirmation of a new kernel K_j the changes to the validator set $\Delta \mathcal{V}_{\mathcal{B}_j}$, and the new gas target gt_j are enacted on the auxiliary system.

On the origin blockchain the state for the meta-block B_i is voted on by $\mathcal{V}_{\mathcal{B}_i}$ and the activity period of the validator set $\mathcal{V}_{\mathcal{B}_i}$ coincides with the duration of the meta-block height i on origin. On the auxiliary system the activity period of the validator set $\mathcal{V}_{\mathcal{B}_i}$ is delayed relative to the state that will be included in the meta-block B_i, as T_{i-1} must be committed on origin and K_i confirmed. It is guaranteed that the confirmation of K_i is included in the state transition of B_i if T_i is calculated by a smart contract on the auxiliary system.

forward and rear validator set   To ensure that a major change to the validator set cannot introduce an attack vector, we apply an analogous defence strategy as described in Casper FFG[2] as the "stitching mechanism".

When a validator's deposit message is included on origin in meta-block height i, the *start meta-block height* $n_0(v)$ is equal to $i + 1$ and the validator joins in the next validator set $\mathcal{V}_{\mathcal{B}_{i+1}}$[15]. When a validator v sees its logout message included on origin in meta-block height $j \geq i + 1$, then the *end meta-block height* $n_e(v)$ is equal to $j + 1$ and the validator participates for a last time in the validator set $\mathcal{V}_{\mathcal{B}_{j+1}}$. Before a logout message for a validator is included the end meta-block height is considered infinite.

We use the start and end meta-block height of a validator to define two subsets of the validator set $\mathcal{V}_{\mathcal{B}_n}$. The *forward validator set* $\mathcal{V}_{\mathcal{B}_{f,n}}$ includes all active validators, except those who are in their end meta-block height $n_e(v)$:

$$\mathcal{V}_{\mathcal{B}_{f,n}} = \{v \in \mathcal{V}_{\mathcal{B}_n} : n < n_e(v)\} = \mathcal{V}_{\mathcal{B}_n} \setminus \{v : n = n_e(v)\} \tag{6}$$

and *the rear validator set* $\mathcal{V}_{\mathcal{B}_{r,n}}$ includes all active validators, except those who have commenced in their start meta-block height $n_0(v)$:

$$\mathcal{V}_{\mathcal{B}_{r,n}} = \{v \in \mathcal{V}_{\mathcal{B}_n} : n_0(v) < n\} = \mathcal{V}_{\mathcal{B}_n} \setminus \{v : n = n_0(v)\}. \tag{7}$$

We redefine that a supermajority link over the validator set $\mathcal{V}_{\mathcal{B}_n}$ henceforth means that a $+2/3$ supermajority of validator weight has voted for the link counted over both the rear and the forward validator set separately:

$$\mathcal{S}(\mathcal{V}_{\mathcal{B}_n}) \doteq \mathcal{S}(\mathcal{V}_{\mathcal{B}_{f,n}}) \wedge \mathcal{S}(\mathcal{V}_{\mathcal{B}_{r,n}}). \tag{8}$$
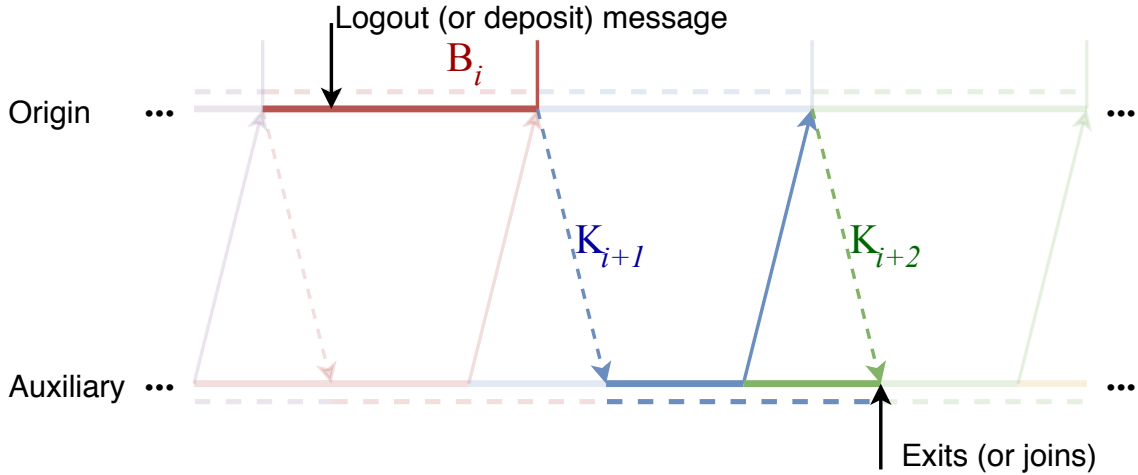
Figure 3: **Dynamic set of validators.** The validator sends a log out message (or a deposit message) in B_i. The change in the set of validators is transferred to auxiliary as part of kernel K_{i+1} that gets confirmed. The validator does its last round in the rear validator set on auxiliary. The validator's weight drops to zero when K_{i+2} is confirmed (or the validator joins in the forward set of validators).

---

[15]We note that in Casper FFG[2] the analogous *start dynasty* of a validator is defined as $d + 2$. The confirmation of the kernel requires a full dynasty increase, i.e. $d + 2$, for it to be confirmed on the auxiliary system, hence it is sufficient for validators to join in the validator set of the next meta-block, i.e. $\mathcal{V}_{\mathcal{B}_{i+1}}$

2.8. Gas Markets

A meta-blockchain has a double gas market. First the known gas market exists where users of the auxiliary system pay gas fees to the block proposers of the auxiliary system for every transaction. A second, new gas market is created that rewards validators for their work.

The meta-blockchain includes transactions from auxiliary. Users of the auxiliary system already paid a gas fee for when the transactions were included in an auxiliary block and cannot be expected to pay another fee later on. Still, it is desired that the validators have an incentive to commit meta-blocks on origin, as it is a pre-requisite for anyone to transfer value from auxiliary to origin.

Mosaic provides two incentives for the validators to do so:

1. The validators receive a gas fee for a committed meta-block, based on the gas that auxiliary consumed for all auxiliary transactions that exist within that meta-block.

2. The validators get rewarded on auxiliary. In order for them to move the rewards to origin, they must first commit a meta-block.

Gas fees. A validators gets a reward for every vote. The amount depends on the relative weight of the validator, the amount of gas that was consumed on auxiliary, and the current gas price. The amount paid out to a single validator address on auxiliary equals

$$\frac{W_v}{w} \cdot g_{st} \cdot gp$$

where W_v is the effective weight of the validator, w is the total weight of all validators, g_{st} is the gas consumed from source to target, and gp is the *current* gas price. Gas fees will be awarded in the base token of the auxiliary chain.

The block proposers decide on the gas price for every checkpoint, but they may not go below a minimum gas price set by the validators. The validators vote on the minimum gas price every n checkpoints, in that way stabilizing the minimum gas price for a foreseeable future. This allows the block proposers to deposit sufficient rewards.

The initial minimum gas price gm_0 is set as part of the genesis block. Every n checkpoints, the validators can vote to either increase the minimum gas price, decrease it, or leave it as is. Based on the relative weights of the votes, the minimum gas price will change within a set limit in change.

Validator rewards. Whenever someone commits a meta-block on origin, the meta-block closes, a new meta-block opens, and someone confirms the kernel of the newly opened meta-block on auxiliary. The validators would usually carry out this duty, as they want to earn their rewards.

The auxiliary system tracks the total gas consumption per checkpoint and can therefore calculate the amount of gas consumed in the closed meta-block on-chain. The gas consumed of a meta-block is merely a mirror of the gas that all auxiliary transactions within that meta-block consumed on auxiliary.

A smart contract on auxiliary pays out a fee to the validators based on the gas consumed in the closed meta-block. In order for the auxiliary smart contract to be able to do the pay-out, the block proposers of auxiliary must deposit sufficient value for multiple future meta-blocks in said contract. Block proposers have an interest in confirmed meta-blocks on origin. They earn on auxiliary for proposing blocks and they can only transfer their value to origin after a meta-block is committed. Furthermore, an auxiliary system with more frequent meta-block commits could be more interesting to users and therefore produce more gas rewards for the block proposers.

If block proposers would fail to produce a sufficient deposit for validator rewards, validators would most likely switch to a more profitable auxiliary chain.

2.9. Attacks and Defenses

We defend against the two main attacks against proof of stake systems the same way Casper FFG[2] does:

- We defend against *long-range revision* by introducing a *withdrawal delay* and a *maximum source block age* for votes.

- We defend against *catastrophic crashes* by introducing an *inactivity leak*.

However, there are a number of new attack vectors introduced by the concepts of a meta-blockchain and decoupled systems:

- Proposing invalid meta-blocks to origin. E.g. from another auxiliary system.
- Delaying kernel confirmation.
- Reporting invalid blocks to the auxiliary system.
- Validators produce votes to commit on origin without actually voting on auxiliary.
- Validators stop finalizing.

Invalid meta-block proposals.   Say a supermajority of validators uses the same private key on two auxiliary chains A_1 and A_2. When these validators sign a supermajority vote that finalizes a checkpoint on auxiliary chain A_1, an adversary could present a transition object from A_1, including the votes to commit it, to the core contract of A_2 on origin. The A_2 core contract would deem the transition object and the signatures as valid. For that reason we introduced the kernel hash to the transition object. This binds every transition object to a kernel and transitively to a specific core contract.

Kernel delay.   The set of validators could have an interest in delaying the confirmation of a new kernel, e.g. to prevent a new validator from joining the set of validators. However, the system is designed in a way that anyone, even the new validator that is not yet part of the set of validators, can do the Merkle proof to confirm a new kernel on auxiliary. All that is required for that is a finalized checkpoint of origin on auxiliary that includes the state root that includes the commit of the previous kernel. If the validators delay origin block finalization to prevent the kernel opening, they would lose stake as the result of the inactivity leak.

Invalid block reports.   Block reporting to auxiliary is not limited to any set of specific addresses. An adversary could report origin blocks that do not actually exist on origin. There is no way for auxiliary to check the validity of these reports. However, validators would ignore these reports and not vote on them, the adversary is only burning gas fees.

Belated vote production.   Say someone proposes a transition object to origin for which a supermajority vote by the validators does not exist. Validators may produce votes to commit that proposal. There are two cases then to consider:

1. The produced votes never violate any slashing condition.
2. The produced votes violate a slashing condition eventually.

In the first case, the commit on origin is in-line with what is actually observed on auxiliary. In the second case, at least $1/3$ of the validators' stake can be slashed.

Finalization breakdown.   If validators decided to stop finalization, they would lose stake due to the inactivity leak.

3. A Mosaic of Cores

summary: By constructing asynchronous BFT consensus rules to compose a heterogeneous auxiliary blockchain into the origin blockchain, we leverage the security of the origin blockchain to asynchronously finalize the transactions on the auxiliary blockchain. As there are no time constraints or time-outs in the Mosaic consensus rules, the limited processing capacity of the origin blockchain does not undercut the ability to compose many auxiliary systems into the same origin blockchain. This enables Mosaic to run many meta-blockchains in parallel, additively improving the total transaction capacity, but keeping a bound on the cost imposed on the origin blockchain.

4. Gateway

abstract: the gateway protocol allows decentralized applications to deploy a message bus to send asynchronous messages between the state space of the origin blockchain and the state space of the auxiliary system supporting

a specific meta-blockchain. This allows tokens (ERC20, later non-fungible tokens) declared on the origin blockchain (Ethereum) to be locked and moved into the meta-blockchain where transactions can be finalized faster, and at a lower transaction cost.

The gateway protocol is code-complete at github.com/openstfoundation/mosaic-contracts

4.1. Message Bus

A Message Bus is a messaging framework that allows to atomically communicate typed messages between two homogeneous or heterogeneous blockchain system. This communication is the two-phased commit on each blockchain to transpose a message from one system to another. Message bus acts as a state machine where message state is changed from one state to another in response to the two-phased commit. If message M is intended to move from blockchain A to blockchain B, then in this case the A is termed as the source and B is termed as target. The state of message M is stored in outbox in source blockchain and in inbox of target blockchain.
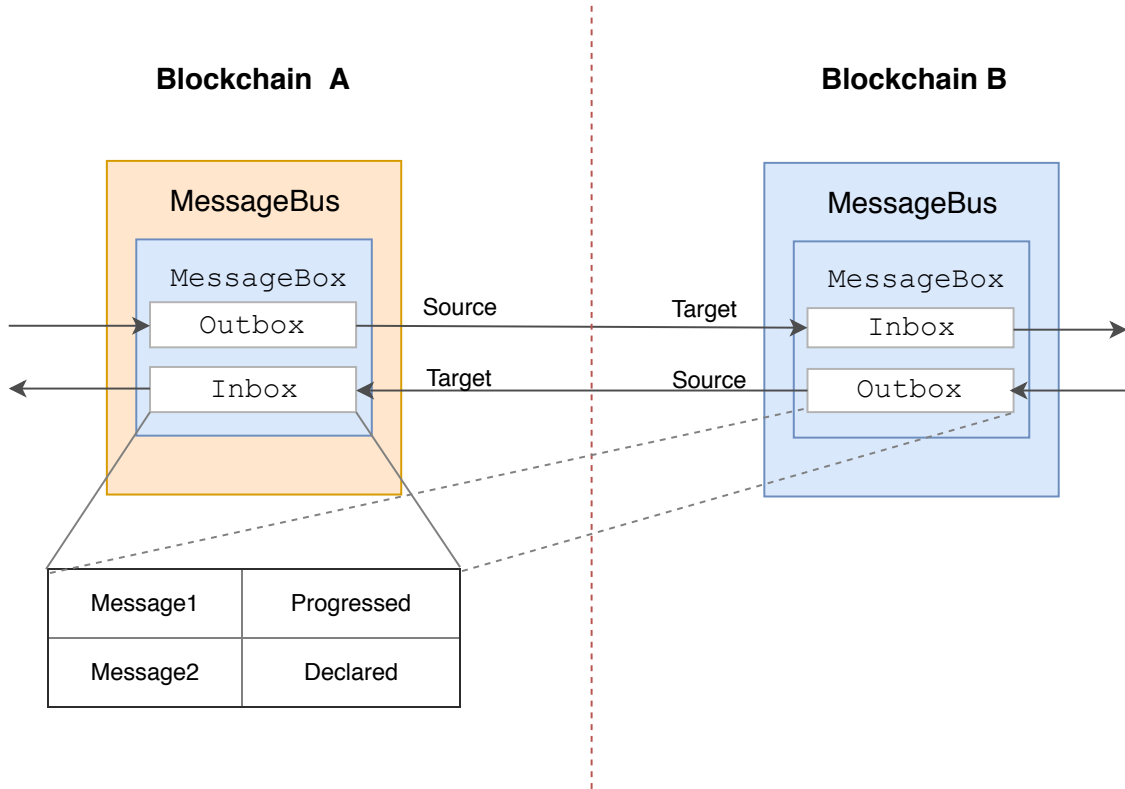
Figure 4: **Message box.**

Similarly if the message is intended to move form blockchain B to blockchain A, then the source will be B and target will be A. Throughout the paper the terms source and target blockchain will be used.

Message. We introduce message object M consists of intent hash i, nonce n, gas price g_p, gas limit g_l, sender s, hash lock h_l , where intent hash i represents the context specific message, example staking intent hash in context of stake and mint process, redemption intent hash in context of redeem and unstake process. Nonce n is the count of message moved across blockchain by the sender s. Gas price g_p and gas limit g_l is the price at which the reward is calculated for facilitation of message transfer, refer 4.8. Hash lock h_l is $keccak256$ of a secret key that is set by the facilitator[16], for more details refer progress with hashlock.

$$M = (i, n, g_p, g_l, s, h_l)$$

---

[16]For more details of facilitator please refer section 4.5

Message hash m_h is an unique identifier of a message M. It can be calculated as the $keccak256$ of message type hash[17] t, intent hash i, nonce n, gas price g_p, gas limit g_l and sender s.

$$m_h = keccak256(t, i, n, g_p, g_l, s)$$

The state of the message is in message box is stored against the message hash.

State machine. Message bus acts as a state machine, the transition of state of message is deterministic. The states of the message can be *Undeclared*, *Declared*, *Progressed*, *RevocationDeclared* and *Revoked*. Message M can be moved from source blockchain to target blockchain in a two phased commit transactions. If the current state for M in outbox of source blockchain is *undeclared*, then first commit is to set the message status as *declared* in outbox of source blockchain. Once the state root of the source blockchain is committed to target blockchain, then the message M can be presented to target blockchain along with the merkle proof specifying the existence of message in source blockchain with status as *declared* in outbox, this is generally done by the facilitator f. If the current state of M in inbox of target blockchain is undeclared and merke proof is valid then the state of M is changed to *declared* in the target blockchain. This completed the one of the two phased commit in both the blockchain. The second commit to change the state to progressed can be done in two ways, firstly **progress with proof** or alternatively gas efficient **progress with hashlock**.
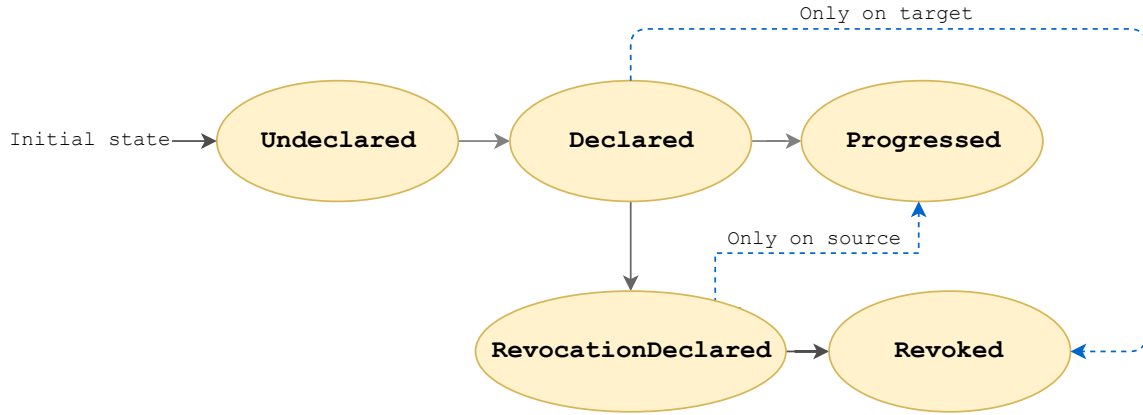

Figure 5: **States of the message.**

Progress with proof. Now the message M exists in both the blockchains in declared state, to complete the atomic communication between the two blockchains it needs to be progressed. The progressed state signifies the completion of transposition of message from source to target. In the source blockchain the merkle proof can be presented for message M specifying the status as *declared* in the inbox of target blockchain. If the merkle proof is valid and current state of message M in outbox of source blockchain is *declared*, then its state is changed to *progressed*. Similarly for the target blockchain, the merkle proof can be presented for message M specifying the status as progressed in the outbox of source blockchain. If the merkle proof is valid and current state of message M in inbox of target blockchain is *declared*, then its state is changed to *progressed*.

Progress with HashLock. The gas consumption in verification of merkle proof is very high, so an alternative mechanism to complete the second commit is to progress with hash lock. While declaring the message M, an hashlock h_l can be provided in the message. Hashlock h_l is $keccak256(secret)$. As this hash lock is part of message this will be available in both the blockchains when its declared. Now by presenting the secret in source blockchain the state of message M can be changed to *progressed* if the current state of message M is declared in outbox. Similarly the same secret can be provided in the target blockchain, the state of message M can be changed to *progressed* if the current state of message M is *declared* in inbox.

Revocation. Any declared message M can be revoked if its current state is not already progressed in any of the two blockchains. In the source blockchain the only case to change the state to *revocationDeclared* is that current state in the outbox should be *declared*. For the target blockchain if the current status of the message M in inbox is *declared* then a merkle proof can be presented specifying the status as *revocationDeclared* in outbox of the source blockchain. If the merkle proof is valid then the status can be changed to *revoked* in the inbox of target blockchain. Incase the message M is not *declared* in the inbox of target blockchain then it

---

[17]Details of type hash can be referred to EIP 712.
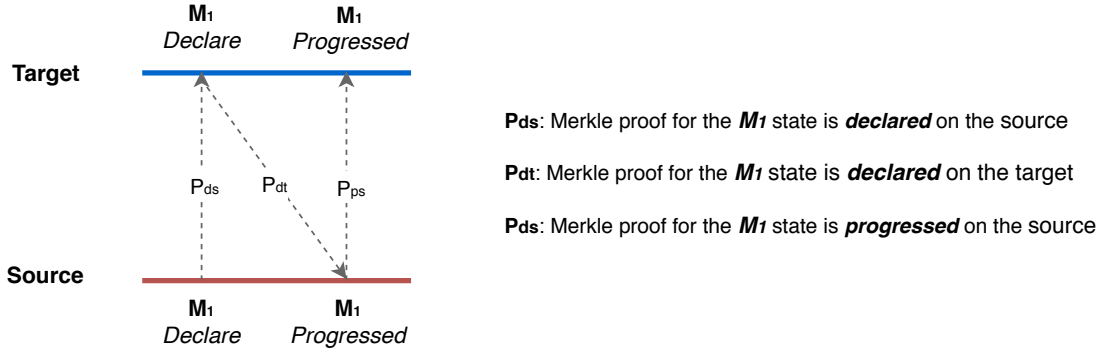
Figure 6: **Progress with proof.**

Pds: Merkle proof for the **M₁** state is **declared** on the source

Pdt: Merkle proof for the **M₁** state is **declared** on the target

Pds: Merkle proof for the **M₁** state is **progressed** on the source
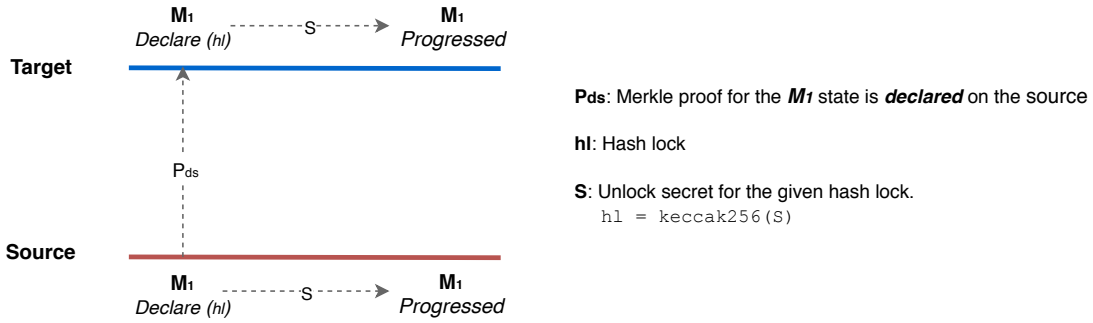
Figure 7: **Progress with hash lock.**

Pds: Merkle proof for the **M₁** state is **declared** on the source

hl: Hash lock

S: Unlock secret for the given hash lock.
```
hl = keccak256(S)
```

needs to be declared first. If the status if message M is already *progressed* in the inbox then state cannot be changed.
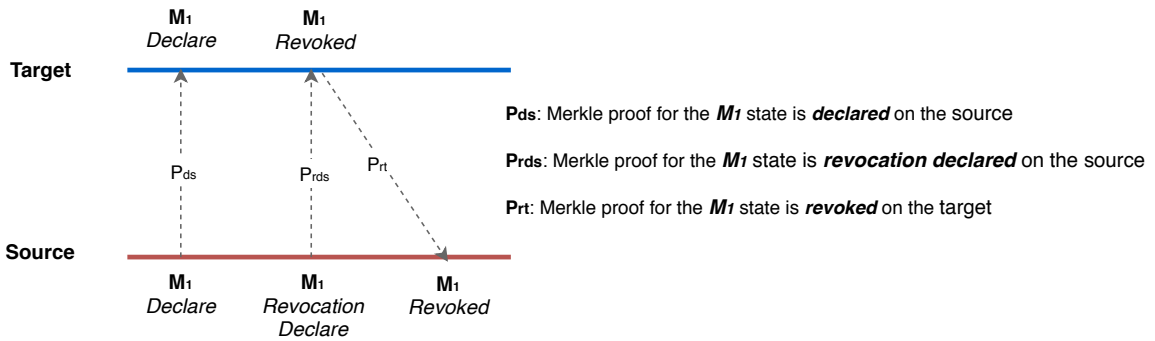
Figure 8: **Revocation of message.**

Pds: Merkle proof for the **M₁** state is **declared** on the source

Prds: Merkle proof for the **M₁** state is **revocation declared** on the source

Prt: Merkle proof for the **M₁** state is **revoked** on the target

On the source blockchain if the message state of message M is revocation declared but the status of message in target blockchain is progressed, in this case the status on source blockchain can be changed only to progressed by presenting the merkle proof for message M specifying the state as progressed in inbox of target chain.

4.2. Linking of Gateway and Co-Gateway.

Gateway and Co-gateway belongs to different blockchain and can co-exists only in pair to be functional. Both the contracts are inactive i.e both the contracts cannot perform any process except the linking. Linking of Gateway and Co-gateway is a process where its cryptographically verified that both contracts have information about one another, are set up to transfer the same token and use the same version of the message bus.

Figure 9: **Progress after revocation declared.**

We define gateway linking intent hash i_{gl} as a hash of gateway address g_a, cogateway address c_a, message bus code hash m_{ch}, ERC20 token name t_n, ERC20 token symbol t_s, ERC20 token decimal t_d, nonce n and token address t_a

$$i_{gl} = keccak256(g_a, c_a, m_{ch}, t_n, t_s, t_d, n, t_a)$$

A gateway linking message M_{gl} can be created with intent hash as i_{gl} as described in section 4.1. This message can be declared in the outbox of source blockchain in order to move the message to target. When the corresponding block of meta-blockchain is committed on target, the message can be presented in target blockchain with merkle proof. The gateway linking intent hash is calculated with the information available with target and it does not use the intent hash provided from the source blockchain. This process will be completed as described in section 4.1

After completion of the process the Gateway and Co-gateway are activated and can perform the other processes like stake & mint or redeem & unstake.

4.3. Bounty

4.4. Reward

4.5. Redeem & Unstake

4.6. Protocol Deactivating

Protocol deactivating is an ability of Gateway and Co-gateway to restrict the movement of messages from source blockchain to target blockchain. If Gateway is deactivated then no new stake and mint process can be initiated, but this does not restrict the redeem and unstake flow. Users can choose to redeem the tokens even if the Gateway is deactivated. Similarly if a Co-gateway is deactivated then no new redemption process can be initiated, but the stake and mint flow can work. If both the Gateway and Co-gateway is deactivated then this will cause the protocol to halt.

5. Outlook

6. Conclusion

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.

[3] Buterin, V. A guide to 99% fault tolerant consensus (2018). URL `https://vitalik.ca/general/2018/08/07/99_fault_tolerant.html`.

[4] Teutsch, J. & Reitwießner, C. A scalable verification solution for blockchains (2016). URL `https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf`.