PoS Finality Informal Specification

March 29, 2021

1 Introduction

In the early stage of a public chain's ecological development, the total amount of computing power is relatively insufficient. This results in a low attack cost to launch 51% attacks and steal the on-chain funds by double-spending. In the last year, Ethereum Classic, Grin and Verge sufferers 51% attack successively. To mitigate the attacker's threat, Conflux introduces PoS finality, which forms a staking committee that recommends confirmed pivot blocks that the PoW nodes should follow.

2 Background

Before introducing the design of the PoS Chain, we remark on some important designs in the current Conflux consensus protocol. This article supposes the readers are familiar with the major concepts in Conflux consensus: including tree-graph, GHOST rule, epoch, sub-tree and past-set.

Deferred Execution Unlike Ethereum, the state root in the block header corresponds to the execution result of 5 epochs before, instead of the current block.

Blaming Conflux does not invalidate a block because of the incorrectness of its state root. But each block will vote for the correctness of the state roots in its pivot chain. If a pivot block receives too many negative votes, it is *blamed* and will not receive the mining reward.

3 Overview

The PoS Chain adopts the Hotstuff consensus protocol. So we can reuse the code in the Tree-Graph Consortium Blockchain. We adopt the same terminologies as Hotstuff [1].

PoS Chain Conflux will allow staking to become PoS nodes, elect PoS committee, and maintains a PoS Chain under Hotstuff Consensus Protocols.

View number The view number acts as a clock in the PoS consensus. In the following part, the term "60 views" represents a time interval in which the view number increased by 60.

A randomly picked committee It could be thousands of nodes in PoS consensus. Conflux picks a random subset of PoS nodes as the committee and switches committee members on time. Each committee member serves 360 continuous views. The terms are staggered like the US Senate. Approximately one-sixth of committee seats are up for election every 60 views.

The election period of a term is started 120 views before the beginning and lasts for 90 views. The first term to the eighth term are not open for election. During the election period, the qualified PoS nodes can submit the output of the VRF function. At the end of the election period, the PoS nodes with minimum output will be elected.

Join and withdraw the PoS consensus Conflux introduces a new contract to receive the staking and assign the eligibility to become PoS nodes. It should support the following functions.

1. Access the context related to the PoS chain. E.g., the view number of the most recent *CommitQC*.
2. Allow an applicant to stake and register public key.
3. Allow fishers to dispute by submitting evidence for the misbehavior of PoS nodes.

The lifetime of a PoS node contains several statuses:

- *Deposit.* The applicant stakes and specifies its public key in PoS consensus.
- *Accepted.* When the PoS chain acknowledges a ledger state in which an applicant stakes, the applicant is accepted as a *PoS node*.
- *Retire.* The PoS node retires if it submits an *retire* transaction in PoS chain. Then it will wait for disputes about proof of misbehavior in 20160 views. A PoS node cannot submit a *retire* transaction if participating in a committee election or being a committee member.
- *Unlock.* If there are no disputes in 20160 views after retiring, the PoS node becomes *unlocked*. Now this PoS node can withdraw its staking.

The PoS nodes are identified by the hash value of their public keys. If a PoS node enters unlock status, it can never participate in the PoS consensus with the same identification.

When a new node is accepted by PoS consensus, it needs to wait for at least 240 views before being a PoS committee member. So a node has enough time to catch up blocks from the latest snapshot. For example, if a PoS node is accepted in view 301, it cannot become a PoS committee member until view 541, so it can participate in election as early as view 421.

Reaching a consensus for pivot block The PoS chain should reach a consensus for a pivot block in the current PoW chain.

1. Each PoS node should run a Conflux full node locally. If the PoS node confirms a block for 240 seconds [1], it marks this block as *pivot candidate*. Each time a PoS node has a new pivot candidate, it should sign it and broadcast the message. In the presence of a PoW attacker with a large amount of computing power, a confirmed block may be reverted, and a PoS node may have a conflict pivot candidate. Our design works well in this case.
2. A pivot candidate with combined signatures from enough nodes (specified later) forms a *pivot decision*, which will be included in the PoS blocks as a transaction. A PoS block can include at most one pivot decision. The pivot decision in a PoS block must extend the pivot decision in the earlier blocks.
3. A replica (committee member) should always sign a *prepareQC* including a valid pivot decision, even if it does not regard pivot decision as pivot candidate or even has a conflict pivot candidate.
4. When a PoS block including a pivot decision, it should also indicate the latest pivot block with a correct state root in its pivot ancestors (including itself). This resembles the blaming process in the PoW chain. The latest correct state is regarded as *state decision*, which implies that this PoS block and its descendants acknowledge it. If the PoS leader does not fill the state decision correctly, the replicas (committee members) should refuse to sign.

Make PoW miners follow the pivot decision Each PoW block should include the block header of the newest PoS block with COMMIT votes called *PoS reference*. When a PoW miner constructs a new PoW block, its PoS reference should extend (or equal to) all the PoS references in its parent block and reference blocks. A PoW block should always be in the subtree of the pivot decision in its PoS reference. A PoW block becomes invalid if it breaks these rules.

Besides, an era genesis block can become stable only if a PoS reference acknowledges it.

[1] Which confirmation rule should be adopted here has not been decided.

Handle the conflict PoS blocks If conflict PoS blocks appear, the existing PoW node should insist on their choice and only accept the PoS block extending the latest accepted one. A new PoW node should accept the PoS reference of the last confirmed blocks. We have more security discussion in section 7.

4 Specification

4.1 Tool functions

BLS signature In the BLS signature scheme, an agent obtains a key pair $(\mathsf{BLS.sk}, \mathsf{BLS.pk})$ with a private key $\mathsf{BLS.sk}$ and public key $\mathsf{BLS.pk}$. The sign algorithm $s = \mathsf{BLS.Sign}(\mathsf{BLS.sk}, m)$ signs message m with private key $\mathsf{BLS.sk}$ and outputs the signature s. A verification algorithm $\mathsf{BLS.Ver}(\mathsf{BLS.pk}, m, s)$ verifies the validity of the BLS signature. The BLS signature is aggregatable such that for a series key pairs $\{\mathsf{BLS.sk}_i, \mathsf{BLS.pk}_i\}$, $s = \sum_i \mathsf{BLS.Sign}(\mathsf{BLS.sk}_i, m)$ is a valid signature accepted by $\mathsf{BLS.Ver}(\sum_i \mathsf{BLS.pk}_i, m, s)$.

Verifiable random function In a verifiable random function, an agent obtains a key pair $(\mathsf{VRF.sk}, \mathsf{VRF.pk})$ with a private key $\mathsf{VRF.sk}$ and public key $\mathsf{VRF.pk}$. The compute algorithm $(\mathsf{VRF.val}, \pi) = \mathsf{VRF.Compute}(\mathsf{VRF.sk}, m)$ computes the random value $\mathsf{VRF.val}$ and proof π with seed m. The verification algorithm $\mathsf{VRF.Ver}(\mathsf{VRF.pk}, m, \pi)$ verifies the correctness of random value.

Verifiable delayed function In a verifiable delayed function, the compute algorithm $(\mathsf{VDF.val}, \pi) = \mathsf{VDF.Compute}(m)$ computes the value val and proof π given input m. It should takes a long time. The verification algorithm $\mathsf{VDF.Ver}(m, \mathsf{VDF.val}, \pi)$ verifies the correctness of output.

4.2 Structures

4.2.1 PoS state

The PoS state is a tuple of $(\mathsf{Nodes}, \mathsf{TermList}, \mathsf{Decision})$.

Nodes The Nodes is a mapping from a NodeID to a Node. A Node is a tuple of $(\mathsf{BLS.pk}, \mathsf{VRF.pk}, \mathsf{Status}, \mathsf{StartTime})$. The Status is an element in $\{\mathsf{accepted}, \mathsf{retire}, \mathsf{unlocked}\}$. The $\mathsf{StartTime}$ indicates the view number of the node entering such Status. The NodeID of a node is defined by $\mathsf{KEC}(\mathsf{RLP}(\mathsf{BLS.pk}, \mathsf{VRF.pk}))$, where KEC represents keccak-256 hash function.

Terms Term i refers the view number in $[60i + 1, 60i + 60]$. If a node is elected in term i, it becomes the committee member from term i to term $i + 5$, e.g., six continuous terms. In the initial stage, all the members elected in term k $(1 \leq k \leq 6)$ are also the committee members before term 6.

The $\mathsf{TermList}$ is a list of Term, each Term is a tuple of $(\mathsf{StartView}, \mathsf{Seed}, \mathsf{NodeIDs})$. The $\mathsf{StartView}$ is the view number of this Term.The Seed is used to be the input of VRF. The $\mathsf{NodeIDs}$ is a list of $(\mathsf{NodeID}, \mathsf{VRF.val})$, ordered in ascending of $\mathsf{VRF.val}$. It's length should always no more than 100. All the Terms should be ordered in ascending of the $\mathsf{StartView}$.

Decision The $\mathsf{Decision}$ is a tuple of $(\mathsf{pivot}, \mathsf{state})$ to denote the block header of the pivot decision and the root of state decision in the current PoS node, and their corresponding metadata (e.g., epoch height).

4.2.2 Transactions

There are three types of transactions, *Decision*, *Election* and *Retire*. Each transaction is a tuple of $(\mathsf{type}, \mathsf{body})$.

Decision The body of *Decision* transaction contains the following fields:

- Target term number; (So the others can decide the committee members)
- A block header hash of pivot chain;
- The epoch height of block header;
- A state root of pivot chain;
- The epoch height of state root;
- The aggregated BLS signature, $> 5/6$ committee members of the target term should sign.

Election The body of *Election* transaction contains the following fields:

- The NodeID of sender;
- The start view number of the term that the sender planned to participate in;
- The VDF answer $\mathsf{VDF.val}$ with proof of $\mathsf{VDF.Compute(Seed)}$, where Seed has been introduced in PoS state;
- The VRF output $\mathsf{VRF.val}$ with proof of $\mathsf{VRF.Compute(VRF.sk, VDF.val)}$;

Retire The body of *Retire* transaction contains the following fields:

- The NodeID of sender;

4.2.3 PoS block

A PoS block contains a block header and a block body.

Block header The block header contains the following fields.

- View Number;
- Parent Hash;
- Leader's NodeID;
- The root of PoS state after executing this block; (The rule of computing PoS state root can be decided during implementation.)
- The Decision in PoS state after executing this block;
- The Merkle root of transactions;
- The Merkle root of unlocked NodeID, which should contains the unlocked NodeID in the PoS state;
- The management contract address on PoW chain;

Block body The block body is a list of transactions.

4.2.4 New field in PoW block header

The PoW block header should has one new field:

- The hash value of a *commitQC* of a PoS block.

4.3 PoS consensus

The PoS consensus inherits the pipelined version of Hotstuff, with the following modifications.

Signature mechanism We use the BLS signature scheme to aggregate signatures. The PoS consensus may have hundreds of committee members. For an aggregated BLS signature, including the NodeID of all the nodes contributing to this signature is redundant. Instead, we could use "term number" and "index" to locate a node.

Leader election We use VRF for leader election. For a committee member, let Seed be the seed when it participates in committee election. $\mathsf{VRF.Compute}(\mathsf{VRF.sk}, \mathsf{VDF}(\mathsf{Seed}).\mathsf{val} \oplus v)$ is the VRF result for leader election, where $\circ$ represents the concatenation of bit-string and v is a big-endian encoded string. Each leader will serve for two continuous views. It means that we only elect a leader for odd view numbers. In each view, the committee members should wait for a sufficiently long time to receive the VRF result for leader election. Once the committee member has chosen the leader, it cannot change its mind in the same view number.

Additional constraints for validity See section 4.4.

4.4 Validity

4.4.1 PoS block

A *QC* of any type is valid if it satisfies the following properties:

- All fields in PoS block header are filled correctly;
- All the transactions in a block are valid (specified later);
- It has at most one *Decision* transaction. Furthermore, the *Decision* transaction must be the first transaction in the transaction list and have a consistent term number with the block packing it.
- It receives signatures from proper nodes, according to its PoS state. We require $> 5/6$ of committee members' sign.

4.4.2 PoW block

A PoW block becomes invalid if it includes an invalid *commitQC* hash. Especially, if there are conflict *commitQC*s or zero *commitQC* hash among its parent block and reference blocks, it should fill the *commitQC* hash with zero.

If a PoW block includes a valid *commitQC*, but it does not follow its pivot decision, it is partially invalid. We do not care if a PoW block has the same opinion as the PoS block in blaming process.

An era genesis block will become stable only if it satisfies existing constraints and is acknowledged by pivot decision in *commitQC*.

4.5 Execute PoS block

The PoS state will be updated in the preparing process of a block or executing a transaction.

4.5.1 Preprocessing

For the PoS block with view number n and term number m, it should updates the state as follows:

- If its parent is not in the same term, it should pop the Term with number $m - 6$ in TermList and push a new Term with number $m + 2$. The StartView of Term i is $60i + 1$, the Seed of new term is the hash value of its parent, the NodeIDs is an empty list.
- The NodeID becoming retire status for 20160 views should be unlocked. Update the Nodes if there are any new nodes be unlocked.

4.5.2 Executing Transaction

Decision The *Decision* transaction is invalid if

- The aggregated BLS signature is inconsistent with the target term number. (Only the committee members in the given term can vote.)
- The block header hash is inconsistent with its epoch height.
- The state root is inconsistent with its epoch height.
- The block header hash does not extent the pivot block in current Decision.
- The state root is not the latest correct state root with respect to the block header hash.

If the *Decision* transaction is valid, the Decision component of PoS should be updated accordingly. If any new nodes have been deposited in the PoW ledger state according to the state root, the Nodes will update corresponding Node to accepted status. If a node is disputed in the PoW chain, it will be dropped from Nodes and TermList at once.

Election The *Election* transaction is invalid if

- The VRF.val is not computed correctly.
- The term is not open for election.
- The sender does not becomes accepted for no more than 240 views at the beginning of this term.
- The sender is in service or participating election of another term, which will not end at the beginning of this term. For example, if a sender is in service of term i, it can not participate in the election from term $i+1$ to term $i+5$. But in term $i+4$, it can participate election for term $i+6$.

If the *Election* transaction is valid, the TermList should be updated accordingly. If the number of NodeIDs in given Term exceeds 100, the NodeID with the largest VRF.val should be popped.

Retire The *Retire* transaction is invalid if

- The sender is in service or participating election of a term.

If the *Election* transaction is valid, its status in Nodes will be updated to retire.

4.6 New Internal Contract

The PoW chain should support a new contract with the following interface:

- deposit(BLSpk, VDFpk). This function allows a user to deposit a given amount of CFX token and becomes a PoS node.
- dispute(BLSpk, VDFpk, sig1, sig2). This function allows a user to submit two different signatures that the PoS node signs for the same view number and the same QC type. It could be aggregated signatures.
- withdraw(BLSpk, VDFpk, path). This function allows a user to withdraw its staked CFX tokens if it is unlocked.

These three functions should be implemented by a solidity contract, other than an internal contract. The internal contract should implement the following functionalities with the most simplified logic, including

- Register/Unregister (BLS.pk, VRF.pk). When registering, the applicant should also provides a zero-knowledge proof of knowing private key for BLS.pk. This can prevent rogue public-key attack on BLS signature scheme. Once a node unregistered, it can never register with the same NodeID. Only the management contract (specified in the PoS block header) can call this interface. In the future, we can introduce a mechanism that allows the PoS nodes to upgrade the management contract address.
- Returns the block header fields in the current PoS block.

5 Activation

5.1 Initialization steps

The activation of PoS finality should consist of the following three stages, each of which starts at a given epoch number.

- The solidity contract for PoS management should be deployed earlier than stage 1. Its address should be decided before the hard fork plan.
- In stage 1, the new internal contract for PoS is activated. However, when the caller tries to query the PoS block header, it will reverts until the PoW blocks include PoS block headers.
- In stage 2, the applicants who have staked successfully before stage 2 become the accepted PoS nodes in the beginning stage. (If the first block of stage 2 has an incorrect state root, we find the latest correct state root before it.) Once the beginning block of stage 2 has been confirmed for one hour, the initial PoS state will be specified later.
- Stage 3 starts at a given timestamp instead of an epoch number. The PoS nodes start to produce PoS blocks at this time.

5.2 Initial PoS state

In the PoS chain, the indexes for term and view are started by 1.

Initalization Seed We need an initialization random seed to compute the PoS genesis state. Let h be the block header hash value of the beginning pivot block of stage 2, t be the timestamp in it. Let b be the block hash of the first bitcoin block with timestamp $t + 3600$. Then the initialization seed s will be

$$s = \mathsf{KEC}(h, b).$$

Conflux team will declare this seed. We believe this setting will not damage the decentralization since the genesis blocks of most blockchain systems are specified by their developer.

Inital Decision The pivot decision is the beginning block of stage 2. The state decision is the last correct state root before the pivot decision, according to the ground truth.

Inital terms Conflux computes the random value $r = \mathsf{KEC}(\mathsf{NodeID}, s)$ for each node, orders these node in ascending of random value. If there are more than 600 nodes, only the first 600 nodes will be accepted in the initial committee. The ordered node list is divided into six balanced groups, and these groups decide the selected nodes from term 1 to term 6. Term 7 has the same nodes as term 1. Term 8 has the same nodes as term 1. Term 9 is the first term open for election. s becomes the Seed of the first eight terms, and the random value r replaces $\mathsf{VRF.val}$ for the first eight terms.

Inital nodes The initial Nodes are the nodes accepted in state decision.

6 Local decisions

There are several points to be designed for local decisions. For example, should a PoS node always participate in the committee election? What strategy should it use? When should a PoS node broadcast its *pivot candidate*?

7 Security discussions

51% Attack If the attacker controls >51% computing power and the PoS chain works well, the pivot blocks that have been decided by the PoS chain are still safe. However, the PoW chain may lose liveness because the PoS nodes cannot confirm any new blocks. If the attacker disappears, the consensus protocol can execute as usual.

17% Staking Attack If the attacker controls >17% committee members, the PoS chain may lose liveness. Then the Conflux chain runs as if there is no PoS finality. If the attacker disappears, the consensus protocol can execute as usual.

84% Staking Attack If the attacker controls >84% committee members, it can generate conflict PoS blocks. In this case, the PoW chain will diverge, even if the attacker disappears.

Long-range attack Different from the 84% staking attack, in a long-range attack, a PoS node becomes malicious not because it intended to break the consensus, but because it lost private keys. So we assume the attacker can only control the majority committee in a very early stage. If the attacker controls >51% computing power and can revert the PoW chain from the early stage, a double-spending attack happens. Otherwise, the consensus protocol can execute as usual.

References

[1] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.