Improving Finality in Waves: Protocol Enhancements and Guarantees

Alexey Kiselev
Waves Foundation
akiselev@uniqsoft.ae

Sergey Nazarov
Waves Foundation
snazarov@units.network

Sasha Ivanov
Waves Foundation
sasha@waves.tech

***Abstract*—Fast finality is a key advantage of modern blockchains, as the user experience is directly affected by how quickly blocks are finalized. In this paper, we present enhancements to the Waves Proof-of-Stake protocol that improve finality by leveraging the voting power of balances actively validating blocks but not currently generating them. We introduce the design of a new finality protocol, demonstrate its desirable properties, and analyze its resilience against common attacks and prolonged network forks.**

***Index Terms*—Waves Blockchain, Proof-Of-Stake, PoS, Finality**

Contents

I. Introduction

Finality is a fundamental concept in blockchain systems, offering assurance that a transaction will not be reversed once confirmed. While Waves achieves high throughput and short block times, it currently lacks a formal mechanism to determine when blocks become final. This leads to uncertainty for users, developers, and systems integrating with Waves, particularly in contexts that demand strong safety guarantees.

This paper addresses this gap by introducing an enhancement to the Waves Proof-of-Stake protocol. Our approach leverages validator commitments and endorsements to track block support over time, enabling deterministic finality based on measurable consensus thresholds.

We formally specify the protocol, analyze its desirable properties, and evaluate its behavior under adversarial conditions. Our findings show that the enhanced finality model improves robustness against forks and enables practical guarantees for irreversible state transitions.

A. Motivation

The emergence of BFT-style consensus protocols has demonstrated that fast finality is achievable, but often at the cost of liveness. In contrast, classical Proof-of-Stake blockchains like Waves prioritize continuous block production and network responsiveness, avoiding the trade-off of halting in uncertain conditions. Nevertheless, achieving fast and reliable finality remains a highly desirable goal.

In this paper, we demonstrate that it is possible to attain both: strong finality guarantees without compromising the liveness properties of the Waves blockchain.

B. Overview of Finality in Blockchains

Finality refers to the point in time at which a block becomes irrevocable, it can no longer be removed or reorganized from the blockchain. Finality can be classified as *probabilistic* or *deterministic*, and as *immediate* or *eventual* [1].

Immediate finality guarantees that once a block is appended to a local copy of the blockchain, it is finalized instantly and will never be reverted.

Deterministic immediate finality is characteristic of permissioned blockchains such as Hyperledger Fabric [2], or BFT/ PBFT-based systems like Tendermint [3]. These systems ensure strong consistency by design and do not allow forks or chain reorganizations.

Probabilistic immediate finality can be found in some permissionless proof-of-stake (PoS) blockchains. A prominent example is Algorand [4], where a two-phase Byzantine Agreement protocol ensures that a block is not produced until quorum consensus is reached. This eliminates the possibility of forks or reorgs while maintaining decentralization.

Blockchains with immediate finality tend to prioritize *consistency over availability* during network partitions or faults, in line with the CAP theorem.

In contrast, *eventual finality* ensures that all honest nodes eventually agree on a common prefix of the blockchain. The confidence in the finality of a block increases as more blocks are added on top of it.

Most early blockchains, such as Bitcoin [5] and Ethereum 1.0 (PoW, pre-Merge) [6], offer *probabilistic eventual finality*. In these systems, blocks can theoretically be reversed, but the likelihood diminishes exponentially the deeper they are in the chain.

A more robust variant is *eventual deterministic finality*, where blocks are not finalized immediately but are guaranteed to become final under certain assumptions (e.g., honest majority, network liveness). Examples include Ethereum 2.0 (PoS, post-Merge) [7], [8], Polkadot [9], [10], Celo [11], and NEM [12].

These protocols strike a balance between deterministic safety and practical scalability. They represent a natural evolution from blockchains with purely probabilistic finality toward models with stronger and more predictable guarantees.

C. Finality Challenges in Waves

The Waves blockchain currently operates as a classic proof-of-stake (PoS) system with *eventual probabilistic finality*. Finality depends on the total generating balance of the active validators (generators) participating in block production. However, changes to this balance are only loosely constrained.

Specifically, there are two main limitations: a minimum balance threshold required to qualify as a generator, and a 1000-block delay before an increased generating balance becomes effective.

As a consequence, the sudden emergence of a generator with a large balance can significantly accelerate chain finalization, while the exit of a major generator can substantially delay block finalization.

In theory, the finality of each block in the Waves blockchain could be determined by summing the generating balance of the block's generator together with the balances of all generators of subsequent blocks. If this cumulative balance exceeds 50% of the total generating stake, the block could be considered finalized.

In practice, however, it is not feasible to accurately compute the total generating balance at any given time, nor is it reliable to track the entry or exit of generators. These limitations make such a finality assessment impractical under the current protocol.

II. Background

Waves operates as a classic Proof-of-Stake (PoS) blockchain, where blocks are considered final once the cumulative balance of generators confirming them exceeds 50% of the total balance of all generators. At this point, no set of generators can create an alternative chain of blocks without surpassing this majority threshold.

Block confirmation occurs when other generators append their blocks on top of a given block, effectively endorsing it.

A. Overview of Waves Proof-of-Stake Consensus

Waves employs a modified Proof-of-Stake (PoS) consensus algorithm known as *Fair Proof-of-Stake* [13], designed to ensure more equitable participation among validators regardless of stake concentration.

In this system, block generation rights are assigned probabilistically based on the effective balance of each account. However, unlike traditional PoS systems that favor large stakeholders disproportionately, Waves introduces a mechanism that *smooths out variance* and improves fairness over time.

The selection process involves generating a random "hit" value derived from the generator's VRF and comparing it against a dynamically calculated target. This target is inversely proportional to the validator's effective balance and the time since their last block. As a result, nodes with smaller balances still retain meaningful chances of producing blocks if they remain online and consistently participate.

Waves also incorporates a *delayed activation* policy for stake increases (1000 blocks), preventing sudden balance shifts from disrupting the fairness of the system.

This PoS model supports the Waves-NG protocol by enabling frequent micro-block production while preserving the security and liveness guarantees of the main chain.

B. Current Finality Mechanism

In the current Waves consensus model, only the total supply of Waves is publicly known. While it is possible to estimate the cumulative balance of accounts that meet the minimum threshold to qualify as generators, the exact set and combined balance of generators actively competing to produce a block at any given time remain unpredictable.

As a result, Waves provides only *probabilistic finality*. There is always a possibility that a hidden group of generators —holding a greater cumulative balance than the currently active set—could construct an alternative chain. If such a chain is eventually revealed, existing generators would be required to accept it, resulting in a chain reorganization.

To mitigate this risk, several technical constraints have been implemented in the reference Waves node software.

The most significant of these is a hard limit on automatic rollbacks: nodes will not reorganize the blockchain beyond 100 blocks. While manual rollback is technically possible, it is further restricted by the design of the node's state storage, which limits rollback depth to a maximum of 2000 blocks.

III. Design Goals

A key challenge in computing finality in the Waves blockchain is the inability to reliably determine the balance of active generators at any given time.

To overcome this limitation, we propose the introduction of an *Explicit Generator Set*, a mechanism that makes the set of active generators and their corresponding balances explicitly defined and easily trackable.

In the following sections, we examine how the Explicit Generator Set contributes to enabling deterministic finality in the Waves protocol.

A. Fast and Predictable Finality

With knowledge of the active generators' balances, it becomes possible to compute, for each block, the ratio between the cumulative balance of generators that have voted for the block, i.e., those who have produced subsequent blocks on top of it, and the total active generating balance. Once this ratio exceeds the threshold of 2/3, the block is considered finalized.

A finalized block is immutable and cannot be removed from the blockchain. Consequently, chain reorganization is only possible up to the depth of the most recently finalized block.

This approach, enabled by the Explicit Generator Set, provides *predictable* finality, but not necessarily fast finality. To address this, we introduce a mechanism for *additional block endorsements* by generators.

This mechanism allows any generator that did not produce the current block but agrees with the choice of its parent block to send an endorsement to the current block's generator. This effectively contributes the endorser's balance to the block's cumulative support.

These additional endorsements allow finality to be reached more quickly: if the combined balance of the block's generator and its endorsers exceeds 2/3 of the total active generating balance, the parent block is considered finalized immediately. In this way, finality can closely track the blockchain tip, lagging behind by at most one block.

B. Security and Fork Resistance

Fast finality must not compromise the network's ability to continue producing new blocks. For this reason, the proposed mechanism is intentionally designed to be *optional and supplementary*. Failures in this mechanism may delay block finalization, but they do not interfere with the ability of generators to produce new blocks.

The only condition that could halt the network is the failure to assemble a valid Generator Set. Even in such cases, a fallback mode, triggered when the Generator Set is empty, can be implemented to allow the network to continue functioning with the additional functionality fully disabled.

Furthermore, the mechanism must preserve the network's ability to *split* in the event of severe network disruptions, and more importantly, to *merge* back once those issues are resolved. To ensure this, we introduce a well-defined procedure for excluding generators from the Generator Set and conduct simulation-based analysis to evaluate its behavior under partition and recovery scenarios.

C. Incentive Compatibility

The introduction of the new mechanism should not interfere with the ability of generators to earn rewards for producing new blocks. However, generators that refuse to participate in the block endorsement process or intentionally disrupt its proper functioning should be subject to penalties. Such penalties may include exclusion from the Generator Set or financial slashing designed to disincentivize malicious or non-cooperative behavior.

In the following section, we define the rules that active generators are required to follow, along with the penalties for non-compliance.

D. Backward Compatibility

The supplementary nature of the proposed Waves finality mechanism allows the network to continue operating exactly as it does today. In such fallback mode, block production remains unaffected, but no explicit finality guarantees will be provided.

This ensures that the mechanism can be adopted incrementally and deactivated safely in environments where full finality support is not yet available or desired.

IV. Protocol Enhancements

The following sections provide a detailed description of the enhancements introduced to the Waves protocol to support *eventual deterministic* block finality.

A. Explicit Generator Set

To ensure transparency and predictability in block generation, each generator wishing to participate must declare their intent in advance.

This is accomplished through a new transaction type called the *Generation Commitment Transaction*. By submitting this transaction, the operator of a generating node requests inclusion of their account in the Generator Set for an upcoming generation period.

The Generator Set is established for a fixed-length period. Once the period ends, the commitment expires, and the operator must submit a new Generation Commitment Transaction to continue participating.

For convenience, each account may have up to two active commitments at any given time: one for the current period and one for the next. It is not possible to register for the current period retroactively, participation must be declared in advance for the next period.

B. Generation Commitment Transactions

The *Generation Commitment Transaction* has no recipient and serves exclusively to register the sender's intent to participate in block generation for a future period.

Each generator must submit this transaction from their own account to signal their commitment to the upcoming generation period.

The transaction includes the following fields:

- **Version** — the transaction version (e.g., 1).
- **Sender Public Key** — the public key of the sender.

- **Generation Period Start** — the block height at which the generation period begins.
- **Timestamp** — the time the transaction was created.
- **Fee** — the transaction fee, paid exclusively in WAVES.

To be accepted, the transaction must satisfy the following conditions:

- The sender's generating balance at the time of validation must exceed 1000 WAVES.
- The sender's account must have sufficient funds to cover both the transaction fee and the required deposit.
- The specified generation period start must correspond exactly to the start of the next period.
- The sender must not have already submitted a commitment transaction for the same period start.

C. Active Generator Set Tracking

Each node must maintain a view of the Generator Set. To do this, it processes Generation Commitment Transactions and includes eligible generators in the list for the current period.

For each active generator in this set, the node stores the public key and the current generating balance. The generating balance is recalculated at the beginning of every new block for all generators in the list. At the same time, the total generating balance of the set is updated.

If a generator's balance falls below the threshold of 1000 WAVES, it loses the ability to produce and endorse blocks until the balance is restored. The total generating balance is adjusted accordingly.

If a generator violates any of the block endorsement rules, it is removed from the Generator Set for the remainder of the current period and its deposit is fully slashed.

If a generator fails to participate in block endorsements, it forfeits a portion of its deposit proportional to its expected level of participation. The penalty amount is calculated, and the corresponding portion of the deposit is burned at the end of the current period.

D. Endorsement Mechanism

Every active generator, upon receiving and validating a key-block, must send back a signature of the parent block to the generator of the key-block.

The key-block generator should validate the received signatures against its copy of the parent block. If the signatures are valid, the generator must include the collected signatures in a designated field of the next micro-block.

Upon receiving such a micro-block, other generators must validate the collected signatures, excluding their own. The current generator may stop accepting endorsement signatures once the cumulative balance of endorsing generators reaches 2/3 of the total generator balance or upon fully filling the signatures block. To achieve this efficiently, the generator can sort supporters in descending order based on their generation balance and prioritize including those with the highest balances first.

If there are no transactions available in the transaction pool, the generator should produce an empty micro-block containing only endorsement signatures.

E. Endorsement Network Message

A new network message type, the *Endorsement Message*, is introduced to propagate block endorsements across the network. It consists of the following fields:

- *Endorser Public Key* — the public key of the generator issuing the endorsement.
- *Endorsed Block ID* — the hash of the block being endorsed.
- *Endorsed Block Height* — the height of the endorsed block.
- *Endorsement Signature* — the digital signature generated by signing the block ID and height, proving the authenticity of the endorsement.

F. Block and Micro-block Structure Updates

To support fast and deterministic finality, the structures of both blocks and micro-blocks must be extended to include endorsements.

- The *Micro-block* structure is updated to include all valid endorsements collected at the time of its creation.
- The *Block* structure is updated to include the full set of finalized endorsements relevant to the parent block.

These updates ensure that endorsements are reliably propagated and recorded, enabling accurate finality tracking and rule enforcement throughout the network.

G. Finality Calculation

Every validation node must calculate the Finality Stake for the most recent blocks. If the Finality Stake of a block reaches the Quorum Balance, defined as 2/3 of the cumulative generator balance, the block is considered final, meaning:

- Blockchain reorganization beyond this block is prohibited.
- The block cannot be excluded from the chain.
- Transactions within the block become irreversible.

The Finality Stake of a block is calculated as the sum of its generator's balance and all endorsement balances recorded in the block. If a single block does not accumulate enough Finality Stake to reach the Quorum Balance, its stake is carried forward and added to the Finality Stake of its parent block. This accumulation process continues up the chain until the required quorum is met, at which point the parent block achieves finality.

H. Detecting Alternative Block Endorsements and Punishing Malicious Behavior

A well-behaved supporting generator must not sign multiple conflicting blocks that reference the same parent. However, malicious actors may attempt to endorse several alternative blocks and propagate these endorsements to different generators.

Due to the decentralized, peer-to-peer nature of blockchain communication, such behavior may initially go undetected. To make it discoverable and enforceable, each block endorsement includes the following fields:

- Supporter's Public Key.
- Endorsed Block Height.
- Endorsed Block Hash.

- Signature (covering both height and hash).

By signing the combination of block height and block hash, any attempt to endorse multiple conflicting blocks can later be proven, even if the alternative blocks were not initially known. The inclusion of the height ensures that the signer cannot plausibly deny a conflicting endorsement once the alternative block becomes visible to the network.

To report such violations, a new transaction type is introduced: the *Violation Report Transaction*. This transaction includes the conflicting endorsement and may be submitted by any network participant.

Once a valid Violation Report Transaction is processed, the following penalties are enforced against the offending generator:

- Exclusion from the Generator Set: The generator's Generation Commitment Transaction is annulled and cannot be replaced. As a result, the generator is excluded from the Generator Set until the end of the period specified in the original commitment.
- Deposit Slashing: The generator's deposit for the current period is confiscated and awarded to the reporter of the violation.

This rule also has implications for node operators who run multiple backup instances using the same private key. If these instances become active on divergent forks, they may unintentionally endorse different blocks at the same height, triggering the same punishment for double endorsement.

V. Finality Threat Analysis

This section outlines key threats to the finality mechanism, prioritized by severity:

- Passive Commitment Attack.
- Commitment Flooding.
- Endorsement Censorship.
- Self-Endorsement Domination.
- Commitment Flip-Flopping.

For each threat, potential mitigation techniques are identified and evaluated.

A. Passive Commitment Attack

In this attack, a large balance holder submits a Generation Commitment Transaction to participate in block generation and validation but deliberately refrains from endorsing blocks. This behavior disrupts finality by inflating the quorum denominator without contributing to endorsement weight.

Possible Mitigations:

1) *Validator Liveness Tracking*: Track the number of validated (endorsed) blocks for each committed validator. Exclude consistently inactive validators from the active generator set. The exclusion threshold may depend on both balance and activity, for example, accounts with larger balances are allowed to skip fewer blocks before being removed.
2) *Commitment Deposit Slashing*: Penalize inactive validators by reducing their commitment deposit in proportion to their validation inactivity. This discourages passive participation and encourages active contribution to finality.

B. Commitment Flooding

An attacker with a large amount of Waves may attempt to overwhelm the network by splitting their balance across thousands of accounts and submitting Generation Commitment Transactions from each, without any intention to generate or validate blocks.

This behavior leads to:

- Increased load on tracking the generators set.
- Performance degradation during generators set verification for each block.
- Potential denial-of-service (DoS) on data structures related to the generators set.

Possible Mitigations:

1) *Raise the Minimum Generation Balance Threshold*

Currently, the minimum generation balance is 1,000 Waves. Increasing this threshold to 10,000 Waves would reduce the number of potentially abusive accounts to a manageable level.

> Note: Generation balance does not affect the ability to validate blocks. Every committed validator is expected to endorse every new block during their commitment period.

1) *Introduce a Non-Trivial Fee for Generation Commitment Transactions*

Introduce a balance-dependent fee model, where lower-balance commitments require proportionally higher fees. For example, a 1M Waves commitment might require a 1 Waves fee, whereas a 1K Waves commitment could incur a 1000 Waves fee. The fee may be refundable in cases of active and honest participation, but burned if the account fails to generate or validate blocks.

1) *Introduce a Cap on the Generator Set Size*

Impose a hard limit on the number of active generators per block — for example, allow only the top 100 generators (by balance) to participate in block validation. More sophisticated prioritization can be introduced, such as using the number of successful generation epochs as a secondary sorting criterion.

C. Endorsement Censorship

An attacker withholds endorsements from other validators when producing a block, preventing the block from collecting enough endorsements to finalize. However, this is not considered a serious attack, as the next block produced by another generator can effectively finalize the entire chain retroactively.

Possible Mitigations:

1) *Penalize Blocks Without Endorsements When a Generator Set Exists*

If the generator set is not empty and includes other active generators, but a block is produced without any endorsements, the generator of such a block should be penalized. Specifically, the generator could be removed from the active generator set or subjected to other slashing mechanisms.

D. Self-Endorsement Domination

A network of generators endorses only blocks produced by member nodes, while withholding endorsements from other validators. This attack can be viewed as an attack on the slashing mechanism of Waves Finality: if the network is large enough and censorship is significant, it could lead to the slashing of honest generators who are not part of the network.

Possible Mitigations:

1) *Reset Validator Rating Upon Inclusion*

The attacking network cannot produce blocks indefinitely without external endorsements. Therefore, when a block produced by a non-member includes endorsements from previously censored validators, their rating should be effectively reset. This prevents long-term penalization of validators who were victims of targeted censorship.

E. Commitment Flip-Flopping

An attacker attempts to repeatedly commit and revoke their Generation Commitment to destabilize the generator set.

However, this attack is not possible because Generation Commitment Transactions are irrevocable and can only expire after the completion of the commitment period.

VI. Simulation and Evaluation

To evaluate the proposed finality model for Waves, we conducted two simulation experiments. In the first, we evaluate how many blocks are required to achieve finality under conditions similar to the actual Waves MainNet balance distribution.

In the second experiment, we explore different approaches to mitigating the consequences of a long-term chain split. We demonstrate that the proposed model can survive such situations and enable chain reunion if the conditions are resolved within a matter of days.

A. Waves Finality Model

This model evaluates the finality process in the Waves blockchain. By simulating the real block generation mechanics, we calculate how many blocks are required to achieve finality under modeled network conditions.

a) *Blocks Data:*

The block data is generated using the utility `waves-delays-generator`. This utility takes as input a list of generator accounts with real balances captured from MainNet in March 2025.

For each generator, a random key pair is generated and associated with its balance. At each simulation step, a hit source (VRF) is generated for each generator, and the delay from the previous block is calculated and recorded in the output file.

The block with the smallest delay is selected and stored as the next blockchain block, while other blocks are considered as possible candidates. In subsequent steps, the selected block is used as the reference for generating the next blocks.

In the output file, block delays are recorded for each height (step) and for each generator.

b) *Finality Model:*

Before evaluating finality, we first determine how many alternative blocks were generated close enough to the block with the minimum delay at each height.

A threshold of 300 milliseconds is used to filter eligible alternative blocks. This threshold reflects the fact that blocks and their alternatives could be received by other generators in a different order, making them valid candidates for inclusion in the blockchain and for receiving votes.

c) *Finality Model Simulation:*

A block is considered final if it receives at least $\frac{2}{3}$ of the total generation balances as votes.

Generators are divided into three groups:

- *Block generator*: A generator that produces the block with the minimum delay.
- *Alternative generators*: Generators that produce alternative blocks within the delay threshold.
- *Non-participating generators*: Generators whose delays exceed the threshold and are considered non-participating in block generation at this height.

Each alternative generator receives a *delay weight* proportional to its closeness to the minimum delay (closer = higher weight).

Non-participating generators are sorted by balance. Their balances are split as follows:

- The *top 50%* (by balance) are assigned directly to the min delay block.
- The *bottom 50%* are redistributed among alternative blocks proportionally to their *delay weight*.

For each alternative block, the total vote is calculated as the sum of the generator's own balance and the share of redistributed non-participating balances.

Finally, for each block, we calculate how many additional blocks are needed until finality is achieved. A block may only finalize later if subsequent blocks finalize and "confirm" it.

Fig. 1 shows the number of blocks required to achieve finalization.

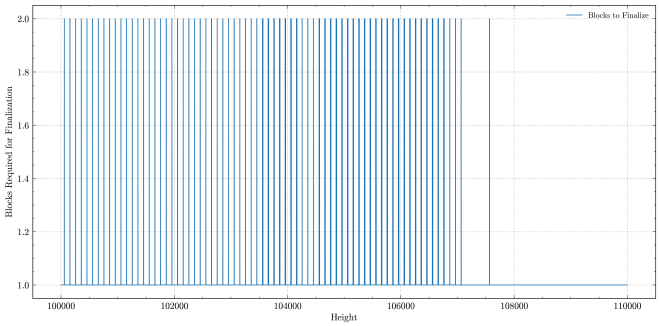

Fig. 1: Finalization: Number of Blocks Required to Achieve Finality

As you can see, the majority of blocks are finalized in 1 block. However, in cases where the network endorses multiple competing blocks, finality is achieved within 2 blocks.

VII. Graceful Chain Splitting

In this analysis, we evaluate the scenario of an *irreparable network split*. In this scenario, the network is partitioned into two parts for a very long time. During the split, finality is temporarily unreachable in both parts due to an insufficient number of generators committed to generate blocks. However, finality is expected to be restored independently in each part after the expiration of old generator commitments.

A. How Generator Commitment Works

To become a generator on the Waves blockchain, an account must hold a sufficient balance and issue a *Generation Commitment Transaction*. This transaction specifies two key parameters: the start height and the duration of the commitment.

- The *start height* allows generators to submit their commitment transaction in advance.
- The *duration* enforces mandatory renewal of commitments, ensuring that inactive or unreachable generators are eventually excluded from the generation process.

The *duration* parameter can be defined either explicitly or implicitly. In the former case, the duration is specified within bounds defined by *lower* and *upper* limits. In the latter case, the duration is fixed and constant for all commitments.

B. Models

We evaluate three models for the expiration and renewal of generator commitments:

1) Synchronized Commitments Model
   This model naturally emerges at the moment the commitment mechanism is introduced. Since the new functionality is activated at a specific block height, all active generators issue their first commitments to start at the same height, but with varying durations.
2) Randomized Commitment Start Model
   Over time, the initial synchronization degrades as:
   - New generators join and issue commitments at arbitrary times.
   - Existing generators may miss renewal deadlines and re-commit asynchronously.

   As a result, commitment start times become naturally randomized across the generator set.
3) Synchronized, Equal-Duration Commitments Model
   In this model, all generators commit at the same start height and use the same fixed commitment duration. However, some may renew in advance while others may not, leading to divergence in expiration after a chain split.

C. Model Explanation

After the network split, the generator set is divided into two partitions: $P_1(t)$ and $P_2(t)$. The number of generators active on both forks at time t is given by:

$$I(t) = |P_1(t) \cap P_2(t)| \tag{1}$$

Finality becomes possible if:

$$I(t) < (1 - f) \cdot N \tag{2}$$

where:

- f is the required fraction for finality (e.g., $f = \frac{2}{3}$ in Waves).
- N is the total number of generators.

Once the intersection falls below $\frac{1}{3}$ of the generators, the forks become independent in terms of finality, meaning that each can safely finalize its own chain. From this point on, rejoining the forks becomes impossible under the finality rules.

All models are evaluated using a dataset of real MainNet generator balances as of March 2025. This dataset is provided in the `mainnet.generators.parquet` file.

D. Synchronized Commitments Model

All generators commit at block height 0, but the commitment duration is randomly selected between 10,000 and 20,000 blocks.

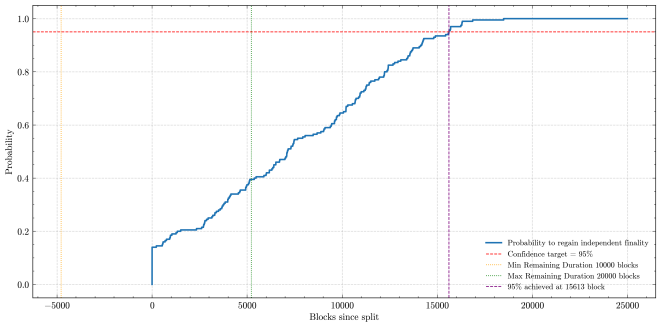

Fig. 2: Probability to Regain Independent Finality (Synchronized Commitments, Random Split)

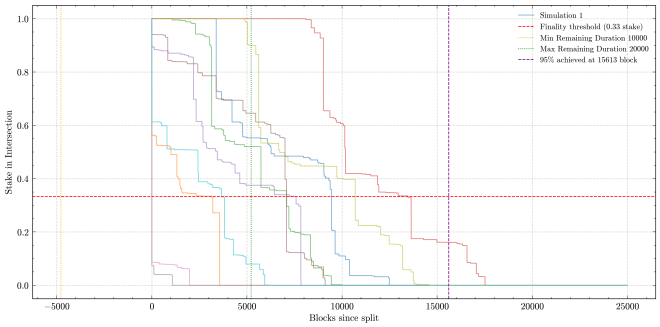

Fig. 3: Decay of Shared Stake Over Time (Synchronized Commitments, Random Split)

The model shows that independent finality is achieved with 95% confidence after approximately 15,000 blocks, or about 11 days from the start of the split.

E. Randomized Commitment Start Model

Generators commit at random block heights, with commitment durations randomly selected between 10,000 and 20,000 blocks.

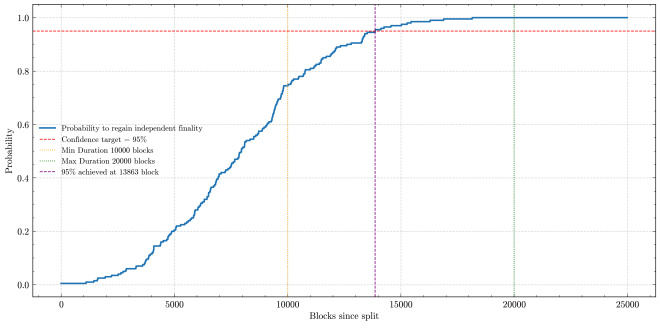

Fig. 4: Probability to Regain Independent Finality (Randomized Starts)

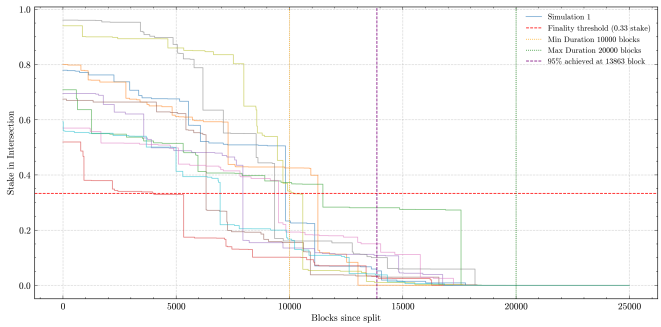

Fig. 5: Decay of Shared Stake Over Time (Randomized Starts)

In this setup, independent finality is achieved after approximately 13,000 blocks, or about 9 days, with 95% confidence.

F. Synchronized, Equal-Duration Commitments Model

In this model, all generators begin their generation periods at the same time, and each commitment has the same fixed duration ($D = 10000$ blocks). The variation arises from whether or not a generator has renewed its commitment before the network split:

- If the generator renewed for the next period, it's generation commitment expires in $2D$
- If not renewed it expires in D

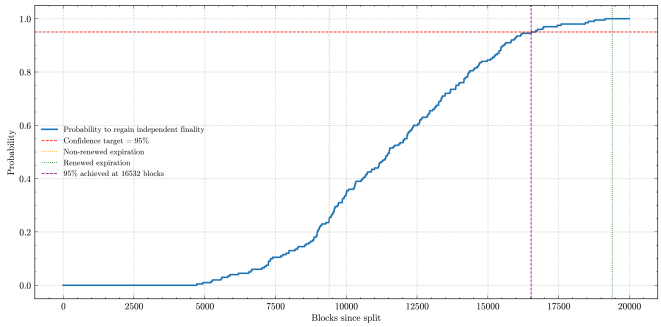

Fig. 6: Probability to Regain Finality (Equal Commitments with Probabilistic Renewal)

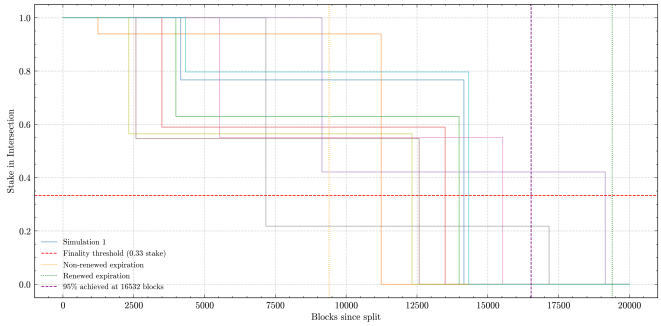

Fig. 7: Decay of Shared Stake Over Time (Equal Commitments with Probabilistic Renewal)

The simulation shows that independent finality is achieved after approximately 16,000 blocks, or about 11 days.

VIII. Conclusion

In this paper, we proposed a protocol extension to the Waves blockchain that introduces *Deterministic Finality* while preserving the system's core properties of liveness and decentralization.

By incorporating an explicit Generator Set and a lightweight endorsement mechanism, the protocol allows nodes to reliably track validator participation and determine finality based on observed consensus rather than heuristics or probabilistic estimates. This design enables block finalization within one or two blocks in typical scenarios, significantly improving user confidence and integration safety for applications built on Waves.

To ensure robustness in adverse conditions such as long-term network splits, we modeled the behavior of generator commitments and demonstrated through simulation that finality can be independently re-established in each fork once outdated commitments expire. We evaluated three commitment renewal models—synchronized, randomized, and synchronized with equal durations—and showed that the system recovers finality predictably within days in all cases.

The protocol is designed to be backward compatible and failsafe: when the Generator Set is absent the blockchain reverts to its original probabilistic behavior without risk of consensus failure.

Overall, the proposed enhancements significantly improve the finality guarantees of the Waves blockchain while maintaining its performance, openness, and adaptability to real-world network conditions.

References

[1] E. Anceaume, R. Ludinard, B. Sericola, and L. Simon, "On Finality in Blockchains," *arXiv preprint arXiv:2012.10172*, 2020, [Online]. Available: https://arxiv.org/abs/2012.10172

[2] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*, ACM, 2018. doi: 10.1145/3190508.3190538.

[3] E. Buchman, J. Kwon, and Z. Milosevic, "The Latest Gossip on BFT Consensus," *arXiv preprint arXiv:1807.04938*, 2018, [Online]. Available: https://arxiv.org/abs/1807.04938

[4] J. Chen and S. Micali, "Algorand: A Secure and Efficient Distributed Ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019, doi: 10.1016/j.tcs.2019.02.001.

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Bitcoin.org*, 2008, [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] V. Buterin, "A Next Generation Smart Contract and Decentralized Application Platform," 2013. [Online]. Available: https://ethereum.org/en/whitepaper/

[7] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," *arXiv preprint arXiv:1710.09437*, 2017, [Online]. Available: https://arxiv.org/abs/1710.09437

[8] V. Buterin, V. Griffith, and a. et\, "Gasper: Combining GHOST and Casper for Proof-of-Stake Consensus in Ethereum 2.0," *arXiv preprint arXiv:2003.03052*, 2020, [Online]. Available: https://arxiv.org/abs/2003.03052

[9] J. Burdges *et al.*, "Overview of Polkadot and Its Design Considerations," *arXiv preprint arXiv:2005.13456*, 2020, [Online]. Available: https://arxiv.org/abs/2005.13456

[10] C. Grunspan and R. Pérez-Marco, "GRANDPA: a Byzantine Finality Gadget," *arXiv preprint arXiv:2007.01560*, 2020, [Online]. Available: https://arxiv.org/abs/2007.01560

[11] T. Moreton, “Consensus and Proof of Stake in the Celo Protocol,” 2019. [Online]. Available: https://celo.org/papers/Celo__A_Multi_Asset_Cryptographic_Protocol_for_Decentralized_Social_Payments.pdf
[12] NEM Foundation, “NEM Technical Reference (Catapult Platform),” 2015. [Online]. Available: https://www.allcryptowhitepapers.com/nem-whitepaper/
[13] A. Begicheva and A. Kofman, “Fair Proof of Stake,” *arXiv preprint arXiv:1908.XXXXX*, 2020, [Online]. Available: https://www.researchgate.net/publication/335147249_Fair_Proof_of_Stake