Acki Nacki: A Probabilistic Proof-of-Stake Consensus Protocol with Fast Finality and Parallelisation

Mitja Goroshevsky(✉), Nikita Sattarov, and Alina Trepacheva

GOSH, 919 North Market Street, Suite 950, Wilmington, Delaware 19801, USA
{mitja,nikita,alina.t}@gosh.sh

Abstract. We propose an asynchronous, highly effective proof-of-stake protocol optimized for fast finality, while allowing for high throughputs via execution parallelization. It is a probabilistic protocol that achieves higher Byzantine fault tolerance than Nakamoto, BFT (including Hotstuff and AptosBFT), Solana, and other modern consensus protocols. Our protocol reaches consensus in two communication steps and has a total number of messages that are subquadratic to the number of nodes, with probabilistic, dynamically adjusted safety guarantees. We trade off deterministic consensus with theoretical constraints on message complexity and the number of Byzantine agreements, with probabilistic algorithms overtaking these boundaries. We further claim that because of the use of randomness and socioeconomics in blockchain designs, no real trade-off is actually present. One of the key ingredients of our approach is separating the verification of execution by a consensus committee from the attestation of block propagation by network participants. Our consensus committee is randomly selected for each block and is not predetermined, while the Leader is deterministic.

Keywords: blockchain · consensus · BLS signature · DDoS attack

1 Introduction

Current public blockchains are almost exclusively used for financial applications, be it for the store and transfer of value or decentralized finance. Users are ready to pay gas and transaction fees when transacting in value. Blockchains do not achieve mass adoption because they cannot support the quality of user experience expected from modern computer software. For one, it is almost impossible to support free transactions to be able to offer a freemium business model for developers. Secondly, the blockchain user interfaces suffer from long delays for task completion related to block finalization times. The primary reason for this user experience inefficiency is the inherent lack of performance in both transaction execution throughput and time to finality, due to strict requirements on state validation. Private blockchains have also failed to achieve mass production in enterprise use cases due to their maintenance complexity and high computing costs.

In this paper, we present a highly efficient, scalable, and practical blockchain protocol optimized for heavy parallelization and extremely fast finality times. The goal of the protocol is to produce performance comparable to cluster cloud databases without compromising security.

Our paper has the following structure: in the next section, we give some background on assumptions and a survey of related works. The third section describes our protocol, the fourth and fifth sections analyze our protocol's security, the sixth section analyzes the performance of our protocol, and the last section concludes.

2 Background

Usually, computer science consensus protocols are classified into two groups: probabilistic and deterministic. The deterministic protocols, under different safety conditions, were developed from 1978 [11,19] to the present day for various applications [5], and with different safety properties, culminating in the development of pBFT [9]. However, they were not used for solving the double-spending problem of decentralized money use cases[1].

2.1 Bitcoin

The first protocol that addressed this use case was introduced by Nakamoto on Oct 31, 2008 [20]. Bitcoin uses a probabilistic consensus protocol based on *Proof-of-Work*, where miners compete to win a slot to propose the new block and be rewarded by expending computing resources to solve cryptographic puzzles.

In Bitcoin, economic incentives play a vital role in network safety and are embedded into the matrix of the protocol's safety guarantees.

The subsequent formula for the probability of a successful Double-Spend attack in the Bitcoin network is based on the article by A. Pinar Ozisik and Brian Neil Levine [21].

$$p_{bitcoin}\left(z,\delta\right) = 1 - \sum_{k=0}^{z+1} \left(\frac{(z\cdot\delta)^k \cdot e^{-z\cdot\delta}}{k!} \cdot \left(1-\delta^{z+1-k}\right) \right),$$

where z - number of blocks till probabilistic "finality", $0 < \mu < 1$ – fraction of malicious miners, $\delta = \frac{\mu}{1-\mu}$.

The downside of the Bitcoin protocol is its performance limitations. Bitcoin is known to produce just 7 transactions per second, and its transaction finalization time can exceed an hour. As we will see below, its security assumptions are

[1] A double-spend attack, in the context of blockchain and digital currencies, refers to a situation where a single set of digital tokens or currency is spent more than once. This type of attack exploits the digital nature of the currency, as digital information can be replicated.

quite weak as well. All of this did not prevent Bitcoin from being the largest cryptocurrency by value to date. However, it did prevent Bitcoin from being used for much more than a store and transfer of value.

Ethereum [31], introduced in 2014 as a smart contract platform [8], initially also used PoW consensus. Its TPS was about 17 per second, but even this was far from sufficient to meet the growing demand for Decentralized Applications (dApps), primarily in the Decentralized Finance (DeFi) sector.

This unsatisfactory performance of PoW forced researchers to look for alternatives.

2.2 BFT

Notably, even before Ethereum, back in 2012, S. King and S. Nadal proposed a protocol they called Proof-of-Stake [15] (PoS), where instead of committing computing resources and electricity, network participants would commit a valuable stake, which they could subsequently lose if proven to act maliciously. This opened a way to use deterministic consensus protocols such as pBFT-based and others [10,14] in cryptocurrency settings, in combination with PoW and later PoS protocols. Many protocols have been proposed since then, and some have been implemented in working systems, improving on the original pBFT messaging requirements and such [2,3,24].

All BFT-based protocols [30,34] generally have two states (faulty or not, 0 or 1) under the protocol assumption. However, in the PoS environment, the decision to act maliciously or not depends not on the properties of the protocol but on the economic realities of the PoS system. In addition, most blockchains rely on probabilistic encryption [13] for their cryptography. Therefore, the BFT consensus algorithms used in the settings of PoS consensus protocols somewhat lose their deterministic properties, as we can no longer prove that non-malicious participants will not turn Byzantine based on the content of the message they are registering, and their determinism will always be bound by cryptographic probability. Thus, if we have a probabilistic consensus protocol with safety guarantees comparable to modern cryptography and/or game theory, it will have practically the same safety as BFT. Yet the penalty we pay in performance for having a presumably deterministic protocol is limiting.

An upper bound on the number of malicious nodes for breaking BFT consensus protocols is $\frac{2}{3} \cdot N + 1$, where N is the total number of nodes in the network. From this, a formula for the successful probability of an attack for BFT consensus protocols is easily derived:

$$p_{BFT}(M, N) = \mathbb{I}_{[\lceil \frac{2}{3} \cdot N \rceil + 1,\ N]}(M),$$

where N is the number of network participants, M is the number of malicious network participants, and $\mathbb{I}_F(x)$ is an indicator function that takes the value 1 if $x \in F$, and 0 otherwise.

2.3 Fast Byzantine Paxos

In [16,17], fast asynchronous Byzantine consensus was proposed. The authors state that this protocol can reach consensus in two communication steps in the common case. However, the cost of such fast finality is that the total number of nodes must be $\geq 5 \cdot f + 1$, where f is the number of Byzantine nodes [18]. Consequently, it can handle a much smaller number of malicious nodes than pBFT. Moreover, it's proven that this bound is tight for deterministic protocols, i.e., for the total number of nodes equal to $5f$, it's impossible to construct a Byzantine consensus that works in two steps.

2.4 Modern Blockchains

Recognizing the performance problems of Nakamoto and BFT consensus protocols, recently a few other approaches have surfaced. We will compare with the three most performant among them: Solana, Avalanche, and Aptos.

Solana. Solana is a blockchain platform engineered for hosting decentralized applications, emphasizing scalability and efficiency. It exhibits a higher transaction processing capacity, with an ability to handle a greater number of transactions per second, coupled with reduced transaction fees. Distinctively, Solana operates on a Proof-of-Stake (PoS) blockchain architecture, but it augments this with an additional mechanism called Proof-of-History (PoH). Yakovenko published a white paper [32] describing the Proof-of-History (PoH) concept. PoH allows the blockchain to reach consensus by verifying the passage of time between events, and it is used to encode the passage of time into a ledger. Instead of individual validator nodes, Solana uses validator clusters, where groups of validators work together to process transactions. Although the PoH-based network has shown some improvements in blockchain throughput, it has been criticized for lacking a sound scientific foundation for its claims [23].

Avalanche. In the Avalanche consensus mechanism [1], nodes decide on transaction acceptance by conducting repeated voting among a small, randomly selected group of validator nodes. When a node needs to determine the status of a transaction, it inquires of a subset of validators for their opinion. These chosen validators respond with their preferred transaction. If a significant majority of the sampled validators agree on a specific transaction, that transaction becomes the choice of the inquiring node. Over time, this node will also favor the transaction that most validators support. This process of sampling and gathering responses continues until there is consistent agreement among the validators over several consecutive rounds.

The threshold for what constitutes a significant majority, and the 'Confidence Threshold,' which is the required number of consecutive rounds for achieving consensus, are both adjustable parameters.

In Avalanche, subsampling has low message overhead. It doesn't matter whether there are twenty validators or two thousand; the number of consensus messages a node sends during a query remains constant. Transitive voting, where a vote for a block is a vote for all its ancestors, helps with transaction throughput. Each vote is effectively many votes in one.

A notable issue arises when multiple blocks are proposed at the same height. In such scenarios, the Avalanche protocol may face delays in determining the correct block to accept, even though all proposed blocks could potentially be valid. This delay is primarily due to the requirement that each block must be executed and assessed by the subset of validators.

As described in the Avalanche white paper [26], the attack probability dynamically changes based on the algorithm's input parameters, such as the number of nodes in the network, the number of malicious nodes, the size of the query sample sent to another node for knowledge about a transaction, and the number of rounds of these queries. Reducing the attack probability directly leads to an increase in finalization time and message complexity. The asymptotic message complexity is $O(k \cdot n \cdot \log n)$ [25], where n is the number of nodes in the network, and k is the size of the sample in a single query, with the constraint $1 \leq k \leq n - 1$.

AptosBFT. Aptos [27] improves on advanced variants of pBFT, namely Hotstuff [33]. In this respect, a comparison with Aptos in general is already described in the BFT section above.

Like many other protocols, Aptos places a lot of emphasis on randomly choosing the Leader and rotating it with every block. The main performance weakness of such an approach is that often, leader rotation necessitates replicating external messages, which users send to the blockchain, to all nodes in the network. This represents an additional quadratic complexity growth overhead, usually excluded when calculating the protocol's messaging complexity.

Sharding. In search of further performance improvements, researchers came up with the concept of sharding, which was first introduced in the Zilliqa blockchain [29] and later developed in Ethereum for state sharding [6,7].

Additionally, several sharded protocols were proposed. These protocols attempted to overcome the performance problem by sharding data and/or execution, introducing parallel leader selection, and state synchronization mechanisms, notably in TON [12], Near [22], Elrond [28], and others. We do not compare these protocols in our analysis because most of them use BFT as their basic consensus algorithm and, therefore, may be considered as belonging to the previously discussed groups.

Although the concepts of parallel execution of contracts and sharded states are important advances in consensus algorithms and have improved network scaling, these concepts alone have not overcome a certain barrier, approximately 100K TPS, even in laboratory environments.

3 Construction of Acki Nacki

Now we present the Acki Nacki probabilistic consensus protocol, with the goal of pushing the performance of fault-tolerant consensus protocols as far as possible.

In Acki Nacki, participants can perform three roles: Block Producer, Block Keeper, and Verifier (which we call an Acki-Nacki entity). All these roles could be performed by any network participant in parallel. Thus, many Acki Nacki chains (called Threads) can exist simultaneously, but since their security and functionality do not depend on each other, we will proceed below with a description of an isolated chain[2].

3.1 Definitions

Definition 1. ***Account*** *(contract) is a record in a distributed database.*

Definition 2. ***Thread*** *is a subset of nodes serving a particular subset of Accounts.*

Definition 3. ***Block Producer (BP)*** *is a leader of a particular Thread, responsible for block production.*

Definition 4. ***Block Keeper (BK)*** *is an entity with two functions:*

– *Receives blocks from BP and sends an Attestation with the block hash and other metadata back to BP. BK does not check the validity of block transactions, nor does it attempt to execute the block, only applies it to its local state with a mark 'Not Final'.*
– *Performs a self-check to determine if it needs to become a Verifier for this block as described below. If so, BK will verify the Block and broadcast the result: Ack if the Block is okay, and Nack if the block is invalid.*

Definition 5. ***Verifier (Acki-Nacki)*** *is a BK responsible for block validation and notifying all network participants about their verdict: whether the block is valid or not.*

Definition 6. ***Attestation*** *is a message sent to BP by any BK after receiving the block. Attestation is a BLS signature performed on BK's private key. The BP of the next block must aggregate all received Attestations for the previous block into one BLS signature and include it in the Common section of the new block.*

Definition 7. ***Ack*** *is a message broadcasted to all network participants by Acki-Nacki if the block is verified and valid.*

Definition 8. ***Nack*** *is a message broadcasted to all network participants by Acki-Nacki if the block is verified and not valid.*

Attestations and Verifier's messages must contain the block hash, its BLS signature [4] on BK's private key, and some extra data. For example, Nack contains the reason for block rejection.

[2] Because of this multithreaded property, Acki Nacki uses an Asynchronous Virtual Machine to execute transactions. This is beyond the scope of this paper, so we mention it here for future references.

3.2 Security Assumptions

We follow standard assumptions of Safety and Liveness [23] properties for Acki Nacki protocol. These properties ensure that the network operation resembles that of a monolithic, valid server, i.e., a linearizably consistent block ledger.

- **Safety**: No two honest BKs accept different blocks of the same height, and no block with an incorrect transaction is finalized.
- **Liveness**: If an honest BP receives a transaction, it will eventually be included in every honest node's ledger.

In accordance with these properties, we classify attacks that violate them:

Safety Attacks. Such attacks include dissemblance and private chain attacks. Dissemblance means that the adversary maintains Byzantine nodes to send different messages to different nodes, potentially leading to nodes' disagreement. Private chain attacks occur when the adversary controls Byzantine nodes to work on a separate blockchain privately while ostensibly following the protocol.

Liveness Attacks. These types of attacks include the aforementioned dissemblance and withholding attacks. Apart from affecting safety, dissemblance may prevent honest nodes from making decisions indefinitely, thereby breaking liveness. Withholding means that the adversary controlling Byzantine nodes refrains from sending messages to particular nodes, potentially causing them to be unable to make decisions indefinitely.

3.3 Block Producer Selection Algorithm

BP selection in Acki Nacki is not random, as the security assumptions of the protocol allow for BPs to be potentially malicious. The following deterministic algorithm is used: the hash of the block with a shard split (or any other Thread rotation demand) message is taken as a seed, and random sampling of one key from the sorted list of BKs' public keys is performed. The current list of BPs is always presented in the Common Section of any Block.

Note: In Acki Nacki, a Block, besides containing TRXs, has a Common Section for collecting block-related data like Attestations, Verifier's messages, BPs list, slashing/reward conditions, etc.

3.4 Acki-Nacki Selection Algorithm

After receiving a block, BK checks whether they are Acki-Nacki for this block. To do this, they calculate $a = \mathsf{sign}(\mathsf{hash}(B), sk)$, where sk is the secret BLS key of BK, B is the current block. They then calculate the remainder of $r = a\%b$, where $b = N/v$, N—the total number of network participants, v—the desired average number of Acki-Nacki, with v such that N is divisible by v without a remainder. Both a and b are integers. If the remainder r equals 0, then it is Acki-Nacki; otherwise, it's not.

Thus, any BK can randomly become Acki-Nacki with a probability of v/N. The selection of each Acki-Nacki is an independent event.

This allows for controlling the average number of Acki-Nacki per block. More details are described in the section "Expected Number of Acki-Nacki per Block" 4.5.

3.5 Block Production and Broadcast

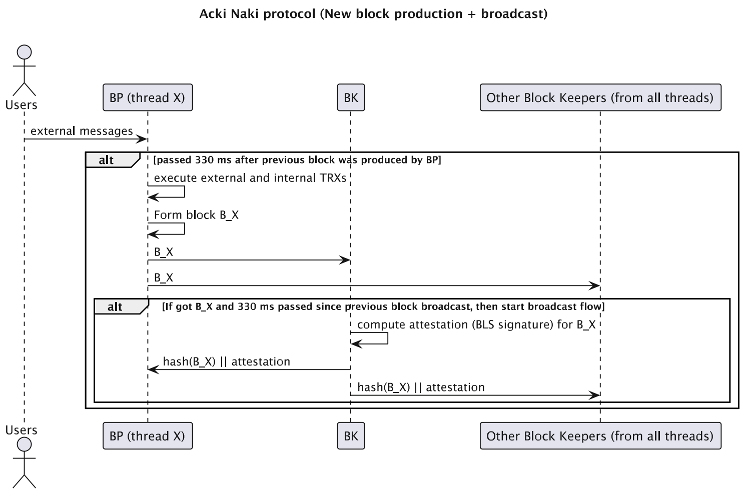
Acki Naki protocol (New block production + broadcast)
Users
BP (thread X)
BK
Other Block Keepers (from all threads)
external messages
alt
[passed 330 ms after previous block was produced by BP]
execute external and internal TRXs
Form block B_X
B_X
B_X
alt
[If got B_X and 330 ms passed since previous block broadcast, then start broadcast flow]
compute attestation (BLS signature) for B_X
hash(B_X) || attestation
hash(B_X) || attestation
Users
BP (thread X)
BK
Other Block Keepers (from all threads)

Fig. 1. The Acki Nacki protocol (Block production and broadcast).

BP releases a new block every 330 ms. As soon as the time arrives, it collects the unprocessed messages, executes transactions, and creates a block (the block is limited by the maximum computed operations and time). Once the block is created, BP signs it with its BLS private key and broadcasts it to all BKs (Fig. 1).

Upon receiving the block from BP, BK checks if the $\mathsf{min.\,block\,timeout}$ since the previous Attestation (at least 330 ms) is satisfied, computes an Attestation for the block (BLS signature), and sends it back to the BP.

The $\mathsf{min.\,block\,timeout}$ from the previous Attestation is necessary to prevent a 'too many blocks' safety attack, where a malicious BP generates so many blocks that verifying them and producing Ack/Nacks in time becomes impossible. The attacking BP can spam the network with valid blocks until it produces a malicious one, which may lead to the acceptance of a block with an incorrect transaction.

3.6 Block Verification

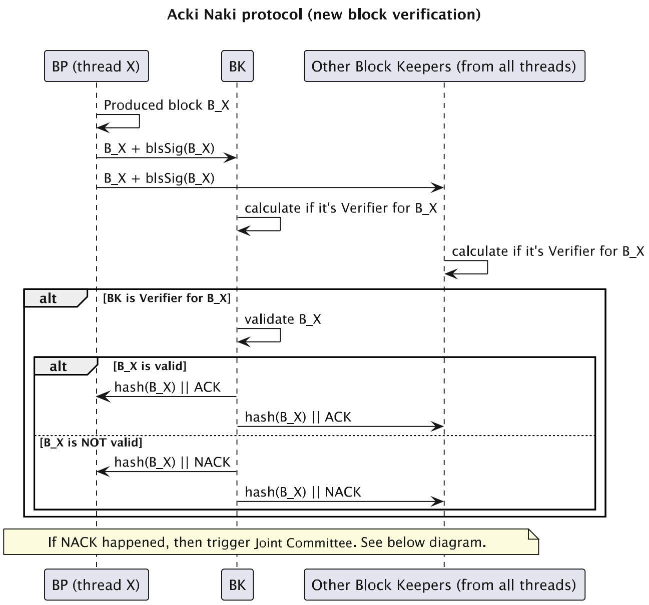
Acki Naki protocol (new block verification)
BP (thread X)
BK
Other Block Keepers (from all threads)
Produced block B_X
B_X + blsSig(B_X)
B_X + blsSig(B_X)
calculate if it's Verifier for B_X
calculate if it's Verifier for B_X
alt [BK is Verifier for B_X]
validate B_X
alt [B_X is valid]
hash(B_X) || ACK
hash(B_X) || ACK
[B_X is NOT valid]
hash(B_X) || NACK
hash(B_X) || NACK
If NACK happened, then trigger Joint Committee. See below diagram.

Fig. 2. The Acki Nacki protocol (Block verification).

Block verification is carried out solely by Acki-Nacki entities that are chosen based on the Acki-Nacki Selection Algorithm, as described above. Acki-Nacki must validate a block and send an Ack/Nack message to the network (Fig. 2). Otherwise, they will be subject to slashing (see 'Slashing' 3.11). While any third party may also validate the block and send acknowledgments, provided they put up a bond, they are not obliged to do so and thus are not part of the security assumptions of the protocol.

If a BP creates a block with overly complex execution, it may attempt to cause Acki-Nacki to delay block verification for more than the $\mathsf{max.}\,,\mathsf{verification}\,,\mathsf{time}$, preventing Ack/Nack transmission within the required timeframe. This could result in a block with an incorrect transaction being accepted.

To mitigate the 'Block with too complex execution' safety attack, an Acki-Nacki that executes such a block will stop after 330 ms and send a special Nack with the message 'too complex'. The committee will then check the block and penalize the BP if necessary.

3.7 Acki-Nacki Selection Proof

Periodically, BK generates a long list of BLS key pairs, sorted by sequential number (SeqNo). Each key pair is intended to be used only once for each block. BK then inserts the hash (BLS private key || SeqNo) into the leaves of a Merkle Tree, computes the Merkle root hash, and commits this hash to the network. After each block, if BK was an Acki-Nacki, it reveals its private key and its SeqNo, along with the Merkle Proof corresponding to this key and the block hash, within the Verification message (Ack/Nack).

It is important to note that the remaining BKs (those not serving as Acki-Nacki for this block) must also reveal their private keys for each block. This can be done at a later stage, for example, in Attestations for the subsequent block. The crucial point is that the reveal phase must be exhaustive in the end. Both Acki-Nacki sending incorrect Ack/Nack messages, or being negligent in not sending Verifications or revealing the keys, will be subject to slashing, as described in the following sections.

3.8 Proof-of-Stake and Fork Choice Rule

Acki Nacki is a PoS protocol that requires all network participants to commit a certain amount of Network Tokens as a Bond. While we do not discuss the economic motivation for becoming a network participant in this paper, we assume that the Tokens have a finite supply, as this plays a role in probability calculations, as will be shown below. In connection with the Fork Choice Rule, we provide an algorithm based on the weight of stakes (Fig. 3).

Sometimes, a situation may arise where the network has two valid blocks at the same height, without any malicious intent. To address this, we have developed the Fork Choice Rule algorithm, which deterministically selects one of the valid blocks for finalization by all BKs.

Key definitions and notations used throughout this section are as follows:

1. N—number of BKs;
2. A— number of Attestations till probabilistic "finality";
3. b_j—block with index j;
4. $\mathcal{K} = \{k_1, k_2, \dots, k_N\}$—BK set;
5. $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$—BK's stake set where s_i is stake of Block Keeper k_i;
6. $\mathcal{A}_j = \{k_i \mid k_i \text{ has attested } b_j\}$—set of BKs that have attested block b_j
7. $\mathsf{hash}\,(b_j)$—hash of the block b_j header;
8. $\mathsf{height}\,(b_j)$—height of the block b_j;
9. $\mathsf{KeySignBP}\,(b_j)$ — key of the BP proposed block b_j;
10. $\mathcal{F}_h = \{b_j \mid \mathsf{height}(b_j) = h\}$—conflicting blocks set at the height h;
11. $\mathcal{A} = \{\mathcal{A}_j \mid b_j \in \mathcal{F}_h\}$—set of $\mathcal{A}_j$ containing Block Keepers $k_i \in \mathcal{K}$ that have attested block $b_j \in \mathcal{F}_h$.

Each BK can attest to only one block at a certain height. In other words, if a BK attests to two blocks at the same height, they will be subject to slashing.

Each BK executes the Acki-Nacki Selection Algorithm only for the block that has more than A Attestations; otherwise, it is executed only for the block that currently has the highest stake amount. For instance, if after verifying block B_i, another block B_j appears at the same height with a greater stake amount, the BK executes the Acki-Nacki Selection Algorithm for block B_j. After applying the Fork Choice Rule, the BK sends to other BKs either the block with Attestations, or the block with Attestations and Ack/Nack, depending on whether they became Acki-Nacki for that block.

```
Algorithm 1 Acki-Nacki Fork Choice Rule
procedure $FCR(\mathcal{K}, \mathcal{S}, \mathcal{F}_h, \mathcal{A}, A)$
    repeat
        update $(\mathcal{F}_h)$                                   ▷ Checking for the receipt of new blocks
        update $(\mathcal{A})$                                     ▷ Checking for the receipt of new attestations
        BPFailure $\leftarrow \exists\, b_i, b_j \in \mathcal{F}_h : \mathsf{KeySignBP}(b_i) = \mathsf{KeySignBP}(b_j)$   ▷ Checking for the several blocks by one BP
        BKFailure $\leftarrow \left| \bigcap_{\mathcal{A}_j \in \mathcal{A}} \mathcal{A}_j \right| \neq \varnothing$   ▷ Attestation of several blocks by one BK
        if BKFailure or BPFailure then
            JOINT COMMITTEE
        else
            if $\exists!\, b_j \in \mathcal{F}_h : |\mathcal{A}_j| \geq A$ then   ▷ Checking for the absence of forks
                return $b_j$
            else
                $\mathcal{U} \leftarrow \left\{ u_j \,\middle|\, b_j \in \mathcal{F}_h,\, \mathcal{A}_j \in \mathcal{A},\, u_j = \sum_{s_i \in \mathcal{S}:\, k_i \in \mathcal{A}_j} s_i \right\}$   ▷ Set of stake amounts having confirmed blocks
                $\mathcal{M} \leftarrow \{u_j \mid \forall u_k \in \mathcal{U} : u_j \geq u_k\}$   ▷ Set of maximum stake amounts
                $\mathcal{D} \leftarrow \{d_j \mid u_j \in \mathcal{U} \backslash \mathcal{M},\, u_m \in \mathcal{M},\, d_j = u_m - u_j\}$
                $d' \leftarrow \min \mathcal{D}$   ▷ Min add stake amount for condition change
                $\mathcal{C} \leftarrow \left\{ s_i \,\middle|\, s_i \in \mathcal{S},\, k_i \in \bigcup_{\mathcal{A}_j \in \mathcal{A}} \mathcal{A}_j \right\}$   ▷ Set of stakes having already confirmed blocks
                $r \leftarrow \sum_{s_i \in \mathcal{S} \backslash \mathcal{C}} s_i$   ▷ Stake amount having not yet confirmed blocks
    until $r \geq d'$ or $\sum_{\mathcal{A}_j \in \mathcal{A}} |\mathcal{A}_j| < A$
    if $|\mathcal{M}| = 1$ then   ▷ Checking for several maximum stakes
        $b' \leftarrow b_j \in \mathcal{F}_h : u_j \in \mathcal{M}$
        return $b'$
    else   ▷ If there are multiple maxima, check hashes
        $\mathcal{H} \leftarrow \{h_j \mid u_j \in \mathcal{M},\, h_j = \mathsf{hash}(b_j)\}$   ▷ Set of hashes of block headers having
        $b' \leftarrow b_j \in \mathcal{F}_h : h_j \in \mathcal{H},\, \forall h_k \in \mathcal{H} : h_j \leq h_k$   ▷ the maximum amount of stake
        return $b'$
```

Fig. 3. Pseudocode of the Fork Choice Rule algorithm.

3.9 Block Finalization

Each BK obtains a new block, mutates the state, and marks the mutations as not final. They then wait for Attestations for this block, sent by BP in the Common section of subsequent blocks, until the minimum Attestation Threshold is reached. The Minimum Attestation Threshold percentage is specified in the network configuration.

After receiving the block and while collecting the necessary amount of block Attestations, BK also waits for T ms as specified in the min. finality time for block B_X. If T ms pass and no Nacks have been received, then BK marks this block as final (Fig. 4).

If there are not enough block attestations, the block won't be finalized. In this case, network participants can decide on the course of action: whether to allow continuous, not finalized block production; to halt the network after a certain number of blocks; to slash or not to slash BKs for not providing attestations, etc.

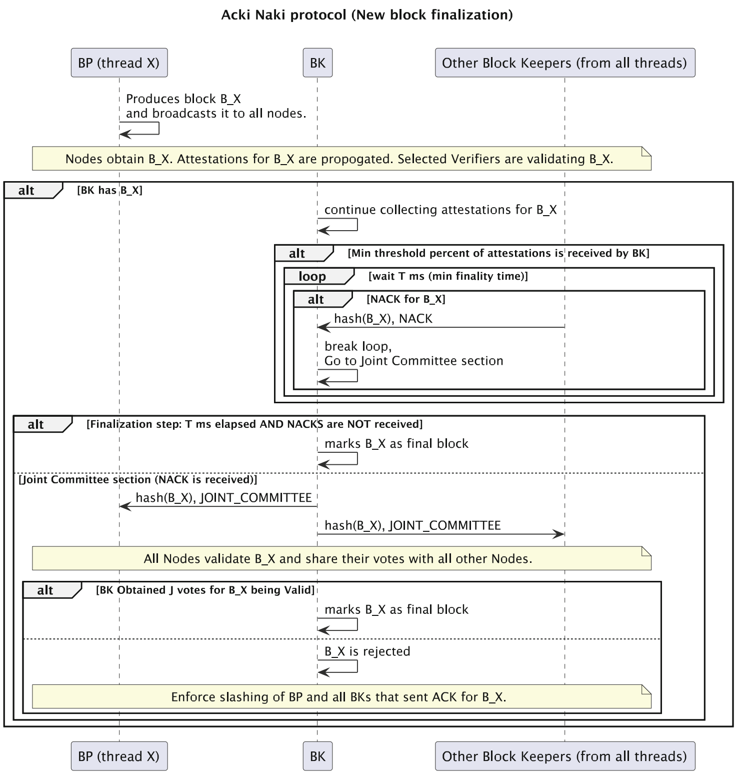
Acki Naki protocol (New block finalization)
BP (thread X)
BK
Other Block Keepers (from all threads)
Produces block B_X and broadcasts it to all nodes.
Nodes obtain B_X. Attestations for B_X are propogated. Selected Verifiers are validating B_X.
alt [BK has B_X]
continue collecting attestations for B_X
alt [Min threshold percent of attestations is received by BK]
loop [wait T ms (min finality time)]
alt [NACK for B_X]
hash(B_X), NACK
break loop, Go to Joint Committee section
alt [Finalization step: T ms elapsed AND NACKS are NOT received]
marks B_X as final block
[Joint Committee section (NACK is received)]
hash(B_X), JOINT_COMMITTEE
hash(B_X), JOINT_COMMITTEE
All Nodes validate B_X and share their votes with all other Nodes.
alt [BK Obtained J votes for B_X being Valid]
marks B_X as final block
B_X is rejected
Enforce slashing of BP and all BKs that sent ACK for B_X.

Fig. 4. The Acki Nacki protocol (Block finalization).

3.10 Joint Committee

If at least one BK receives a Nack, or if they receive an Attestation for the same height on more than one block from a single BK, or detect any other malicious action as described in the ‘Slashing’ section, they invoke the Joint Committee function. This requires each BK to vote on whether to slash malicious BKs, malicious BPs, and/or to reject a malicious block. In response, each BK either verifies the block, checks Attestations, or performs any other necessary action to ascertain whether someone is acting maliciously or otherwise falls under one of the Slashing conditions. To confirm or reject that an action is a result of malicious activity, the network must gather J votes, where J is a parameter set by the network participants.

3.11 Slashing

The following are slashing conditions which can lead network participants to either lose their entire stake or a portion of it (bleeding). When we say 'lose,' we mean that the stake will be burned and not transferred to any other party. Burning plays a role in security assumptions, as discussed in a separate section below.

- An accepted Nack will slash the BP and every Acki-Nacki which sent an Ack for their entire stake.
- Attesting to more than one block at the same height will result in the slashing of the entire stake of the BK.
- Producing two blocks at the same height from the same BP will result in the slashing of the BP's entire stake.
- A non-performing Acki-Nacki will result in the bleeding of its stake.
- A non-performing BK will result in the bleeding of its stake.
- Non-randomized BK keys will result in the bleeding of the stake.
- Non-sequential Acki-Nacki Keys in the Acki-Nacki Merkle Tree will result in the bleeding of the BK's stake.
- Too complex execution of a block by BP will result in the bleeding of its stake.

3.12 Dynamically Adjustable Parameters

One of the main advantages of the Acki Nacki consensus protocol is the presence of several dynamically adjustable parameters. These include the number of Attestations needed for block finalization, the average expected number of Acki-Nacki per block, the number of votes required for the Joint Committee, and the probability of a successful attack given a certain percentage of malicious BKs. All these parameters can be changed by network participants through voting, according to their preferences. For instance, participants can input the number of BKs and the desired attack probability with a certain number of malicious BKs. Based on this, Acki Nacki will then automatically adjust the parameters for the number of Attestations and Acki-Nacki, ensuring that the network achieves the highest throughput with the shortest finality.

4 Attack Analysis

4.1 Input Parameters

Notations, types and domains of the terms used in this section:

Name	Notation	Type	Domain
Number of BKs	N	$\mathbb{Z}_+$	$[3; +\infty]$
Number of malicious BKs	M	$\mathbb{Z}$	$[0; N-1]$
Number of spammed BKs	d	$\mathbb{Z}$	$[0; N-M]$
Number of Attestations	A	$\mathbb{Z}_+$	$[1; N]$
Successful attack probability	p	$\mathbb{R}$	$(0; 1)$
Expected number of Acki-Nacki per block	v	$\mathbb{Z}$	$[0; N]$
Number of votes for Joint Committiee	J	$\mathbb{Z}_+$	$[1; N]$

4.2 Combined Double-Spend and DDoS Attack

There are N BKs, of which M are malicious. The malicious BKs, using a distributed denial-of-service (DDoS) attack[3], disconnect d honest BKs from the network and perform a Double-Spend attack (Fig. 5). Verification prevents attacks on consensus. If at least one of the honest BKs, which have survived the DDoS attack, becomes an Acki-Nacki, the attack is deemed unsuccessful.

4.3 Constraints on the Number of Malicious Block Keepers

A malicious block will always be finalized if none of the honest BKs survive after a DDoS attack. Additionally, malicious blocks will always be finalized if, in the Joint Committee, the number of malicious BKs exceeds the number of honest BKs. Thus, to determine the number of malicious BKs at which the attack will be successful, or, in other words, the number of malicious BKs at which the Safety property is violated:

$$M \geq min(A, J) \tag{1}$$

At the same time, if all malicious BKs disconnect and stop sending messages, the network will halt, as it will not be able to collect enough Attestations. A similar situation may occur if all malicious BKs disconnect or reject Nacks for a malicious block during the execution of the Joint Committee function. Thus, to determine the number of malicious BKs at which the network will stop, or, in other words, the number of malicious BKs at which the Liveness property is violated, we have:

$$M \geq \min(N - A + 1, N - J + 1) = N + 1 - \max(A, J) \tag{2}$$

As the constraints on the number of malicious BKs depend on the number of Attestations needed for block finalization and the number of votes for the Joint Committee, the Acki Nacki Security Assumptions are dynamically adjustable. This allows the network to maintain the probabilistic safety property even when the number of malicious BKs exceeds 50% of the network participants, and

[3] A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

furthermore, even when it exceeds 66% of the network participants. This holds true for any assumption regarding the number of malicious BKs agreed upon by network participants through voting.

4.4 Constraints on the Number of Spammed Block Keepers

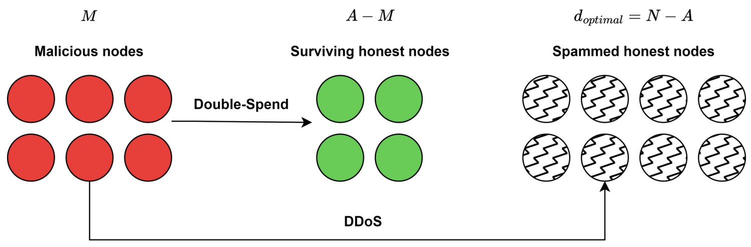

Fig. 5. Scheme of the optimal DDoS Attack

Since consensus requires A Attestations, among the $(N - d)$ BKs that continue to function after the DDoS attack, at least A BKs must be capable of gathering the required Attestations. Therefore $N - d \geq A$.

In order for malicious BKs to launch a successful attack, they aim to spam as many honest BKs as possible.

From this understanding, we conclude that $d_{optimal} = N - A$.

4.5 Expected Number of Acki-Nacki per Block

The probability of becoming an Acki-Nacki per block for each BK is expressed as v/N.

Let the random variable ξ denote the number of BKs that become an Acki-Nacki. Then, the probability that k BKs become Acki-Nacki is described by the following probability:

$$\mathbf{P}\left(\xi = k\right) = C_N^k \cdot \left(\frac{v}{N}\right)^k \cdot \left(1 - \frac{v}{N}\right)^{N-k} \tag{3}$$

We find that the random variable ξ follows a Binomial distribution, and its expected value is $\mathbb{E}(\xi) = v$. In other words, v is precisely the mathematical expectation of the number of BKs that become an Acki-Nacki per block.

4.6 Successful Attack Probability in the Acki Nacki Consensus

Since honest BKs only collect A number of Attestations to finalize the block and check for the absence of Nacks, the successful attack probability on the block will be equal to the probability that no honest surviving BK has become Acki-Nacki:

$$p\,(N, M, d, v) = \left(1 - \frac{v}{N}\right)^{N-M-d}. \tag{4}$$

Since malicious BKs DDoS the maximum possible number of honest BKs, then the resulting successful attack probability on the block is expressed as:

$$p\,(N, M, A, v) = \left(1 - \frac{v}{N}\right)^{A-M}. \tag{5}$$

5 Safety Analysis

We assume that if malicious network participants successfully attack the network at least once, the entire network breaks. Let's find the probability of at least one successful attack on the network in R attempts:

$$\mathbf{P}\,(\text{at least one successful attack}) = 1 - (1-p)^{R}. \tag{6}$$

Since, in the Acki Nacki consensus protocol, the probability of breaking the network at least once increases more rapidly when colluding with more malicious BKs than when attempting more times, it is more advantageous to attempt once with the maximum possible number of malicious BKs. The number of malicious BKs may be at a maximum of $A-1$. If $A > N/2$, then it is almost impossible to place stakes for so many malicious BKs, so let's assume that attempts to break the network will be made about once every year. We say 'a year' rather than 'a day' because if the attacker had easy enough access to that much money, why wouldn't they buy the whole network at once?

Even if the attacker attempts to attack our network once a year, they have a limited number of attempts since all stakes of the network participants that were slashed are burned.

Figure 6 and Fig. 7 are illustrating the successful attack probability from a number of malicious network participants for Bitcoin, pBFT, and Acki Nacki protocols with a total of 1000 network participants. To calculate the successful attack probability in Bitcoin, we use the commonly accepted number of blocks for probabilistic 'finality', which is 6. For calculating the successful attack probability in Acki Nacki, we use the number of Acki-Nacki set to 40 and the number of Attestations set to 800.

As observed, Acki Nacki provides significantly higher security guarantees compared to Bitcoin. Furthermore, this holds true when compared to pBFT, especially in scenarios where the number of malicious network participants exceeds 2/3 of the total network participants. To further illustrate this point, we compared the probability of at least one successful attack on our network in the coming years with the probability of a comet hitting the planet Earth, leading to a global catastrophe. Assume that such a comet falls once every 10^6 years.

As we can see in Fig. 8, it is more likely that a comet will fall in the coming years and destroy life as we know it than for malicious BKs to successfully attack the network.

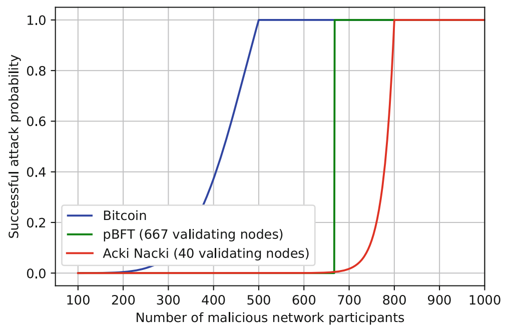

Fig. 6. Comparison of successful attack probabilities in Bitcoin, pBFT and Acki Nacki

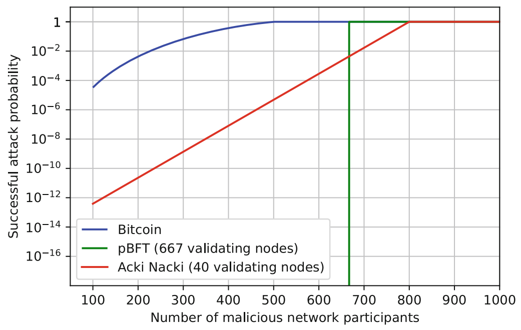

Fig. 7. Fig. 6 with log-scaled y-axis

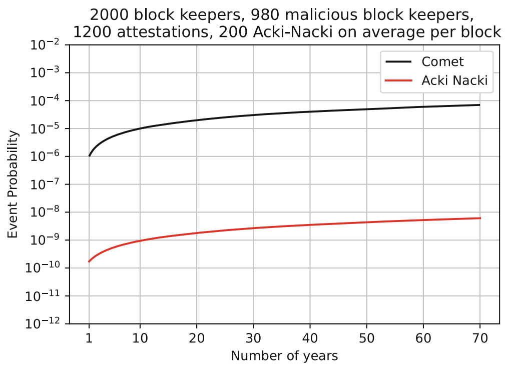

Fig. 8. Comparison plot of the successful attack probability in the Acki Nacki consensus protocol with the global catastrophe probability

6 Performance Analysis

Without taking state sharding into account, the limitation to performance in the Acki Nacki network boils down to two factors: the number of blocks a BK can receive over the network and apply, and the number of blocks all network Acki-Nacki can process at any given moment. This performance is entirely dependent on the computer and network resources committed by participants, the number of BKs, and the expected number of Acki-Nacki per block.

We do not discuss state sharding solutions in this paper, but it is quite easy to envision an Acki Nacki sharded design: some BKs can choose not to store a state that belongs to a certain address space. The only remaining practical limitation in an asynchronous system would be how messages passed from one Account to another, residing in two different shards, would be executed by the BP and verified if the BP and Acki-Nacki do not possess the state of one of the participating Accounts.

With a sharded design, there is no theoretical limit to the throughput of the Acki Nacki network. Without sharding, and considering modern computer hardware and datacenter internet connections, we calculate a practical limit of $250,000$ transactions per second for minimal 500-byte messages, achieving less than 1-second finality. With sharding enabled, the protocol can scale to millions of transactions of any complexity, merely by adding computing resources, making it comparable to centralized cloud services.

Acki Nacki achieves this performance as a result of significantly reduced message complexity during most of its operation time.

The Acki Nacki algorithm achieves consensus in two communication steps. The first step involves sending the block from BP to BKs. The second step involves sending Ack/Nacks from Acki-Nacki to all BKs, in parallel with the sending of Attestations from BKs to BP.

In total, the following messages are sent: The block from BP to BKs, the Attestations from BKs to BP, and the Ack/Nack messages from several chosen Acki-Nacki to BKs. Here, the optimistic scenario ends. The Nack message and accidental Forks will trigger more messages, but as we have shown, Nack messages are highly improbable, and Forks are rare events. Most of the time, the network will operate by sending just 3 types of messages, where the total number of all messages sent equals $(N-1) \cdot v + 2 \cdot N$. The message complexity of Acki Nacki depends on the desired security parameters, taking into account that, in practice, $v \ll N$.

7 Conclusion

We have demonstrated an efficient probabilistic consensus protocol with reduced message complexity and high parallelism in transaction execution, leading to fast finality times and scalability improvements. Our security assumptions are dynamic and can change during network operations, demonstrating the protocol's flexibility. Our safety analysis shows high adaptability to network parameters while maintaining desired safety guarantees.

References

1. Ava Labs, Inc.: The avalanche documentation. avalanche consensus (2024). https://docs.avax.network/learn/avalanche/avalanche-consensus
2. Bach, L.M., Mihaljevic, B., Zagar, M.: Comparative analysis of blockchain consensus algorithms. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1545–1550. IEEE (2018). https://doi.org/10.23919/MIPRO.2018.8400278
3. Berrang, P., von Styp-Rekowsky, P., Wissfeld, M., França, B., Trinkler, R.: Albatross - an optimistic consensus algorithm. In: 2019 Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 39–42. IEEE (2019). https://doi.org/10.1109/CVCBT.2019.000-1
4. Boneh, D., Drijvers, M., Neven, G.: Bls multi-signatures with public-key aggregation. In: ASIACRYPT (2018). https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html
5. Boneh, D., Shoup, V.: A graduate course in applied cryptography (2020). https://dlib.hust.edu.vn/bitstream/HUST/18098/3/OER000000253.pdf. draft 0.5
6. Buterin, V., et al.: Ethereum roadmap, what about sharding? (2022). https://ethereum.org/en/roadmap/#what-about-sharding
7. Buterin, V., et al.: Combining ghost and casper (2020). https://doi.org/10.48550/arXiv.2003.03052
8. Buterin, V., Wood, G.: A next generation smart contract and decentralized application platform. White Paper (2014). https://static.peng37.com/ethereum_whitepaper_laptop_3.pdf
9. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI 1999), New Orleans, Louisiana, pp. 173–186. (1999). https://pmg.csail.mit.edu/papers/osdi99.pdf
10. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems, pp. 34–50 (2022). https://doi.org/10.5281/zenodo.6353717
11. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
12. Durov, N.: Telegram open network blockchain (2020). https://ton.org/tblkch.pdf. white Paper
13. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984). https://mit6875.github.io/PAPERS/probabilistic_encryption.pdf
14. Grigg, I.: Eos-an introduction (2017). https://iang.org/papers/EOS_An_Introduction.pdf. white paper
15. King, S., Nadal, S.: Ppcoin: peer-to-peer crypto-currency with proof-of-stake (2012). https://decred.org/research/king2012.pdf
16. Ku, T.W., Chen, K.: No need for recovery: a simple two-step byzantine consensus (2019). https://doi.org/10.48550/arXiv.1911.10361
17. Martin, J.P., Alvisi, L.: Fast byzantine consensus. In: 2005 International Conference on Dependable Systems and Networks, DSN 2005, pp. 402–411 (2005). https://doi.org/10.1109/DSN.2005.48
18. Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Trans. Depend. Secure Comput. **3**(3), 202–215 (2006). https://doi.org/10.1109/TDSC.2006.35

19. Merkle, R.C.: Secure communications over insecure channels. Commun. ACM **21**(4), 294–299 (1978). https://doi.org/10.1145/359460.359473
20. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
21. Ozisik, A.P., Levine, B.N.: An explanation of nakamoto's analysis of double-spend attacks. CoRR abs/1701.03977 (2017). https://doi.org/10.48550/arXiv.1701.03977
22. The NEAR White Paper (2021). https://near.org/papers/the-official-near-white-paper
23. Shoup, V.: Proof of history: what is it good for? (2022). https://www.shoup.net/papers/poh.pdf
24. Sun, Z., Chang, J., Zhu, N., et al.: Rangers protocol 2.0 (2022). https://rangersprotocol.obs.ap-southeast-1.myhuaweicloud.com/Navigation/RangersProtocolWhitepaper.pdf
25. Team Rocket: Snowflake to avalanche: a novel metastable consensus protocol family for cryptocurrencies (2018). https://knowen-production.s3.amazonaws.com/uploads/attachment/file/1922/Snowflake%2Bto%2BAvalanche%2B-%2BA%2BNovel%2BMetastable%2BConsensus%2BProtocol%2BFamily.pdf
26. Team Rocket, Yin, M., Sekniqi, K., van Renesse, R., Sirer, E.: Scalable and probabilistic leaderless bft consensus through metastability (2020). https://doi.org/10.48550/arXiv.1906.08936. Cornell University
27. The Diem Team: Diembft v4: State machine replication in the diem blockchain (2021). https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf
28. The MultiversX Team: Multiversx, a highly scalable public blockchain via adaptive state sharding and secure proof of stake (2019). https://files.multiversx.com/multiversx-whitepaper.pdf. Technical whitepaper - release 2 - revision 2
29. The Zilliqa Team: The zilliqa technical whitepaper (2017). https://docs.zilliqa.com/whitepaper.pdf
30. Tse, S., Liu, M., et al.: Harmony technical whitepaper-version 2.0 (2023). https://harmony.one/whitepaper.pdf
31. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2014), p. 32. https://ethereum.github.io/yellowpaper/paper.pdf. Ethereum project yellow paper 151.2014
32. Yakovenko, A.: Solana: a new architecture for a high performance blockchain v0. 8.13 (2018). https://solana.com/solana-whitepaper.pdf
33. Yin, M., Malkhi, D., Reiter, M., Gueta, G., Ittai, A.: Hotstuff: bft consensus with linearity and responsiveness. In: 38th ACM Symposium on Principles of Distributed Computing (PODC 2019), Toronto, ON, Canada, 29 July–2 August 2019 (2019). https://doi.org/10.1145/3293611.3331591
34. Zhong, W., et al.: Byzantine fault-tolerant consensus algorithms: a survey. Electronics **12**(18), 3801 (2023). https://doi.org/10.3390/electronics12183801