Acki Nacki

***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 which achieves higher Byzantine fault tolerance than Bitcoin, BFT or other modern consensus protocols. Our protocol reaches consensus in two communication steps with probabilistic dynamically adjusted safety guarantees. We trade off deterministic consensus with theoretical constraints on message complexity and number of byzantine agreements with probabilistic algorithms overtaking these boundaries. We further claim that because of the use of randomness and social-economics 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 the block propagation by network participants. Our consensus committee is randomly selected for each block and is not predetermined while the Leader is deterministic.

***Index Terms*—blockchain, probabilistic consensus, BFT, BLS signature, DDOS attack**

I. Introduction

Current public blockchains are almost exclusively used for financial applications, be it 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 can not support the quality of user experience expected from modern computer software. For once 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 prime reason for this user experience inefficiency is inherited lack of performance in both transaction execution throughput and time to finality because of strict requirements on state validation. Private blockchains have also failed to achieve mass production in enterprise use cases for its maintenance complexity and high computing cost.

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 comparable performance to cluster cloud databases without compromising security.

II. Background

Usually the computer science consensus protocols are classified into two groups: probabilistic and deterministic. The deterministic protocols under different safety conditions were developed from 1978 [1], [2] to date for different applications [3] and with different safety properties culminating in development of pBFT [4], but were not used for solving the double spending problem of decentralized money use cases. Thus the first protocol that addressed that use case was introduced by Nakamoto on Oct 31 2008 [5]. Bitcoin uses a probabilistic consensus protocol based on Proof-of-work where miners are trying to win a slot to propose the new block and be rewarded by spending computing resources to solve cryptographic puzzles.

A. Bitcoin

In Bitcoin the economic incentives plays vital role in network safety and are embedded into the matrix of the protocol safety guarantees:

Subsequent formula about successful Double-Spend attack probability in the Bitcoin network is based on the article by A. Pinar Ozisik and Brian Neil Levine [6].

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

where z – number of blocks till probabilistic “finality”, $0\leq\mu\leq1$ – fraction of malicious miners, $\delta=\frac{\mu}{1-\mu}$.

The downside of Bitcoin protocol is its performance limitations. Bitcoin is known to produce just 7 transactions per second and its transaction finalization time can reach more than an hour. As we will see below its security assumptions are quite weak as well. All of which did not prevent Bitcoin from being the largest cryptocurrency by value up to date. But it did prevent Bitcoin from being used as something more than value store and transfer.

Ethereum [7] introduced in 2014 as a smart contract platform [8] initially also used PoW consensus. Its TPS was about 17 per second but even that was far from enough to meet the growing demand for Decentralized Applications (dApps), primarily in the Decentralized Finance (DeFi) sector.

This unsatisfactory performance of the PoW forced researchers to look for an alternative.

B. BFT

Notably even before Ethereum, back in 2012, S. King and S. Nadal proposed a protocol they called Proof-of-Stake [9] (POS) where instead of committing computing resources and electricity network participants would commit a valuable stake, which they can subsequently lose if proven to act maliciously. This opened a way to use deterministic consensus protocols such as pBFT based and others [10], [11] in cryptocurrency settings in combination with POW and later POS protocols. There are many protocols that were proposed since then and some that were implemented in working systems improving on original pBFT messaging requirements and such [12]–[14].

Notably all BFT [15], [16] based protocols have generally two states (faulty or not, 0 or 1) under the protocol assumption,

but 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 of the blockchains rely on probabilistic encryption [17] for its cryptography. Therefore the BFT consensus algorithms used in the settings of POS consensus protocol are somewhat losing their deterministic properties as we no longer can prove that non malicious participants will not turn Byzantine based on the content of the message they are registering and as 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 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, $\mathbb{I}_F(x)$ is an indicator function that takes the value 1 if $x \in F$, and 0 otherwise.

Recognizing performance problems of Nakamoto and BFT consensus protocols lately few other approaches surfaced. We will compare with two most performant of them: Solana and Avalanche.

C. Fast Byzantine Paxos

In [18], [19] fast asynchronous Byzantine consensus was proposed. The authors state this protocol can reach consensus in two communication steps in the common case. But the cost of such fast finality is that the total number of nodes must be $>= 5 \cdot f + 1$, where f is the number of Byzantine nodes. So 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 equals to $5 \cdot f$, it's impossible to construct Byzantine consensus working in two steps.

D. Comparison

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

1) Solana: Solana is a blockchain platform designed to host decentralized, scalable applications. Solana can process many more transactions per second and charges lower transaction fees than rival blockchains like Ethereum. Solana is a PoS blockchain but improves on it with a mechanism called proof-of-history (PoH), which uses hashed timestamps to verify when transactions occur.

Yakovenko published a white paper in November 2017 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 validator nodes, Solana uses validator clusters, where groups of validators work together to process transactions.

Today, Solana (SOL) is considered the fastest and most scalable network. According to the project whitepaper [20], the Solana network is capable of processing up to 710,000 transactions per second. However, this is only a theoretical number. During tests, processing speeds only reached 65,000 TPS. Meanwhile, the project provides a block finalization time of 21-46 seconds.

2) Avalanche: Avalanche uses a novel consensus mechanism that builds on the POS foundation. When a transaction is initiated by a user, it's received by a validator node that samples a small, random set of other validators, checking for agreement. The validators perform this sampling procedure repeatedly, "gossiping" with each other to ultimately reach consensus.

In this way, one validator's message is sent to other validators, which sample more validators, which sample even more validators – again and again, until the whole system reaches agreement on an outcome. Just as a single snowflake can become a snowball, a single transaction can eventually turn into an avalanche.

Validator rewards scale according to the amount of time a node has staked its tokens, called Proof of Uptime, and if the node has historically acted according to the software's rules, called Proof of Correctness.

Avalanche users can launch specialized chains that can operate using their own sets of rules. This system is comparable to other blockchain scaling solutions, like Polkadot's parachains and Ethereum 2.0's shards. Consensus on these chains is reached by subnetworks (or subnets), which are groups of nodes that participate in validating a designated set of blockchains. All subnet validators must also validate Avalanche's Primary Network.

3) Aptos: Aptos [21] improves on pBFT advanced variants, namely Hotstuff [22] In this respect comparison with Aptos in general already described in the BFT section above.

Like many other protocols Aptos puts a lot of attention in randomly choosing the Leader and rotating it every block. Main performance weakness of such an approach is that often leader rotation presents a necessity to replicate external messages which users send to a blockchain to all nodes in the network. This represents an additional quadratic complexity growth overhead usually excluded when calculating protocol messaging complexity.

E. Sharding

In search of further performance improvements, researchers came up with the concept of sharding which was first introduced in the Ziliqa [23] blockchain and then developed in Ethereum for state sharding [24], [25].

Additionally several sharded protocols were proposed. These tried to overcome the performance problem by sharding

data and/or execution by introducing parallel leader selection and state synchronization mechanisms, notably TON [26], Near [27], Elrond [28] and others. We do not compare with these because most of them use BFT as a basic consensus algorithm and therefore may be considered belonging to the previous groups.

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

III. The Construction of Acki Nacki

Now we present an Acki Nacki probabilistic consensus protocol with a goal to take the performance of fault tolerant consensus protocols as far as we can.

In Acki Nacki there are three roles participants can perform: Block Producer, Block Keeper and a Verifier (which we call Acki-Nacki entity). All of these roles could be performed by any network participant in parallel. So many Acki Nacki chains (called threads) can exist simultaneously, but since their security and functionality does not depend on each other we will proceed below with a description of an isolated chain[1].

A. Definitions

Account (contract) is a record in a distributed database.

Thread is a subset of nodes that serves a particular subset of Accounts.

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

Block Keeper (BK) is an entity having two functions:

- Receives blocks from BP and sends out an Attestation with block hash and other metadata back to BP. BK does not check block transactions validity, it does not try to execute the block, only apply it to its local state with a mark 'Not Final'
- Performs a self check if it needs to become a Verifier for this block as described below. If it does, BK will verify the Block and broadcast the result: Ack, if the Block is ok and Nack, if the block is invalid.

Verifier (Acki-Nacki) – is a BK being responsible for block validation and notifying all network participants about his verdict: is block valid or not.

Attestations – message that is sent to BP by any BK after receiving the block. Attestation is BLS signature done on BK's private key. BP of the next block must aggregate all received attestations for the previous block into one BLS signature and include it into the Common section of the new block.

Ack – Verifier's message that is broadcasted to all network participants by Acki-Nacki if the block is verified and it's valid.

[1]Because of this multithreaded property Acki Nacki is using an Asynchronous Virtual Machine to execute transactions but this is beyond the scope of that paper, so we are just mentioning it here for future references

Nack – Verifier's message that is broadcasted to all network participants by Acki-Nacki if the block is verified and it's NOT valid.

Attestations and Verifier's messages must contain block hash, its BLS signature [29] on BK's private key. Some extra data may be added. For example Nack contains the reason for block rejection.

B. Block Producer selection

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

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

C. Acki-Nacki Selection Algorithm

After receiving a block BK checks whether he is Acki-Nacki for this block or not. To do this, he calculates $a = hash(B||sk)$ (or $r = sign(hash(B), sk)$), where sk is secret BLS key of BP and then calculates the remainder of $r = a \bmod b$, where $b = N/v$, N – the total number of network participants, v – the desired average number of Acki-Nacki. If remainder r equals to 0, then it is Acki-Nacki, otherwise it's not.

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

D. Acki-Nacki Selection Proof

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

Note that remaining BKs (that are not Acki-Nacki for this block) also must reveal their private keys for each block. It can be done later: for example in Attestations for the next block. The point is that the reveal phase must be exhaustive in the end. Both Acki-Nacki sending incorrect Ack/Nack messages or being lazy and not sending Verifications, nor showing the keys will be slashed as described below.

E. Dynamically adjustable parameters

One of the main advantages of Acki Nacki consensus protocol is presence of several dynamically adjustable parameters such as the number of Attestations needed for block

finalization, the average expected number of Acki-Nacki per block, and the probability of a successful attack with a certain percentage of malicious BKs. All these parameters can be changed by network participants through voting according to their preferences. For example, one can input the number of BKs, the desired attack probability with a certain number of malicious BKs. Acki Nacki will then automatically adjust the parameters of the number of Attestations and the number of Acki-Nacki so that the network achieves the highest throughput with the shortest finality.

F. The new block production and broadcast

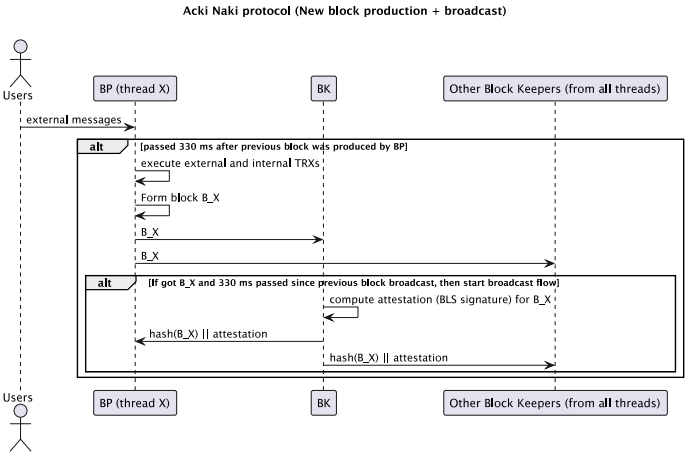

Fig. 1: The Acki Nacki protocol (New block production and broadcast).

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

BK gets the block from BP, check if min. timeout from previous attestation is satisfied (see "Too many Blocks" below) and computes Attestation for the block (BLS signature) and sends it to the BP.

G. The block verification

Block verification is carried out obligatory only by Acki-Nacki entities that are chosen based on the above formula. They must validate the block and send the Verifier's Ack/Nack message to the network. Otherwise they will be punished. Any third party may also validate block and send acknowledgments, as long as they put a bond, but this is their free will, they are not obliged.

H. Proof-of-Stake and Fork Choice rule

Acki Nacki is a POS protocol which requires all network participants to commit a certain amount of Network Tokens as a Bond. We do not discuss economic motivation for becoming a network participant in this paper, but we assume that Tokens

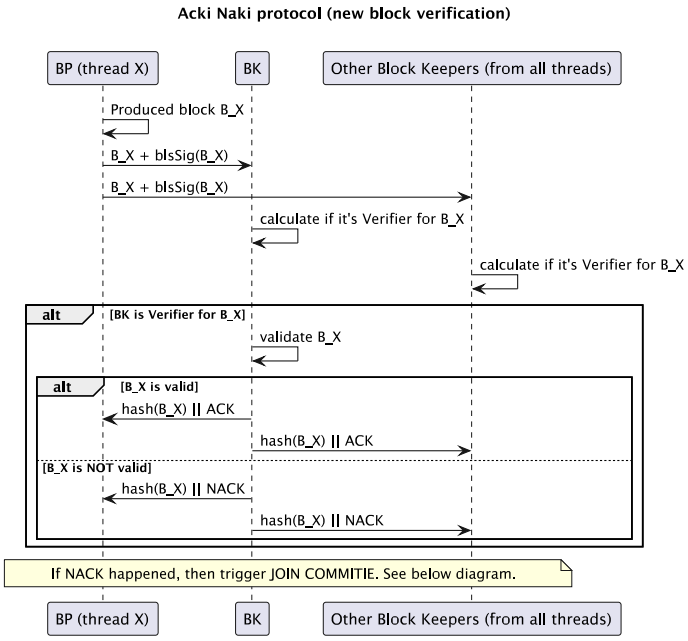

Fig. 2: The Acki Nacki protocol (Block verification).

have finite supply as it plays a role in probability calculations as will be shown below. In connection with the Fork Choice Rule we provide an algorithm based on stakes weight.

Sometimes a situation can arise when the network has two valid blocks at the same height, without a malicious intent. We developed the Fork Choice Rule algorithm to deterministically select one of the valid blocks for finalization by all BKs.

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

Key definitions and notations used throughout this section:

1) N — number of BKs;
2) A — number of Attestations till probabilistic "finality";
3) b_j — block with index j;
4) $\mathcal{K} = \{k_1, k_2, \ldots, k_N\}$ — BK set;
5) $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ — 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) $\mathsf{SignBP}\,(b_j)$ — signature of BP that 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 executes the Acki-Nacki Selection Algorithm only for the block that has more than A Attestations; otherwise, only for the block that currently has the highest stake amount. For example, 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 $\leftarrow \exists b_i, b_j \in \mathcal{F}_h : \mathsf{SignBP}(b_i) = \mathsf{SignBP}(b_j)$ ▷ Several blocks by one Block Producer
    BKFailure $\leftarrow \left| \bigcap_{\mathcal{A}_j \in \mathcal{A}} \mathcal{A}_j \right| \neq \varnothing$ ▷ Attestation of several blocks by one Block Keeper
    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.

I. The block finalization

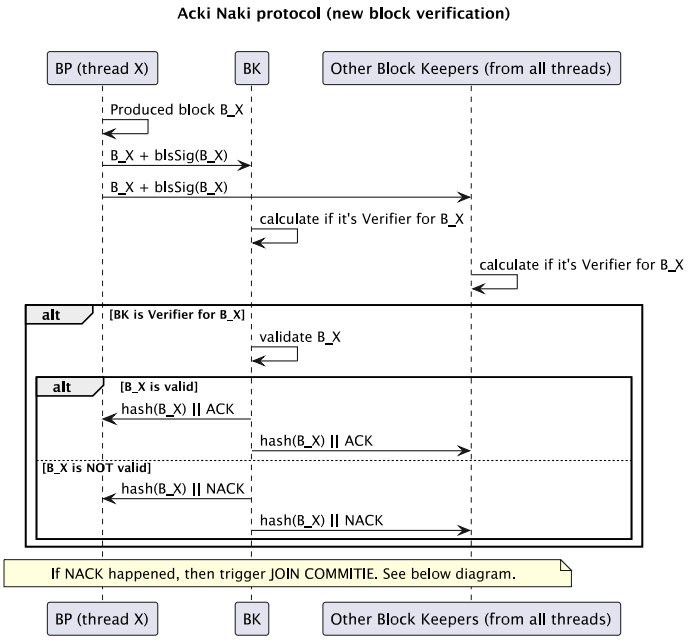

Fig. 4: The Acki Nacki protocol (Block verification).

Each BK obtains a new block, mutates the state and marks mutations as not final and waits for Attestations for this block sent by BP in the Common section of the next blocks to reach min Attestation Threshold. . Minimum Attestation Threshold percent is specified in the network config.

After the block is received while collecting the necessary amount of block Attestations, BK also waits T ms as specified in min. finality time for block B_X. If T ms passed and no Nacks were received, then Node marks this block as final.

J. Joint Committee

If at least one BK receives a Nack, receives an Attestation of the same height on more than one block by one BK, or detects any other malicious action as described in Section "Slashing", they invoke the Joint Committee function and requests each BK to vote for slashing malicious BKs, malicious BPs, and/or rejecting a malicious block. In turn each BK either verifies the block, checks Attestations, or performs any other action to ensure that someone is acting maliciously or otherwise falls under one of the Slashing conditions. To accept/reject that an action is malicious activity, the network must gather J votes, where J is a parameter set by the network participants.

K. Slashing

The following are slashing conditions which will lead network participants to either lose their stake entirely, or by some factor (bleeding). When we say "lose" we mean that the stake will be burned and not transferred to some other honest party. Burning plays a role in security assumptions as discussed in a separate section below.

- Accepted Nack will slash BP and every Verifier which sent an Ack for the whole stake
- Attestation of the same height of more than one block will slash BK for the whole stake
- Two blocks of the same height from the same BP will slash it for the whole stake
- Non performing Verifier will bleed its stake
- Non performing BK will bleed its stake
- Non randomized BK keys will bleed stake
- Non sequential Verifier Keys in Verifier Merkle Tree detection will bleed BK stake
- Too complex Execution of a block by BP will bleed its stake

IV. Attacks

A. Combined DDoS and Double Spent attack

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 voices for Joint Committiee	J	$\mathbb{Z}_+$	$[0; N)$

1) Input Parameters: There are N BKs, of which M are malicious. The malicious BKs, using DDoS, disconnect d honest BKs from the network and perform a Double-Spend attack. Verification prevents attacks on consensus. If at least one of

the honest BKs, which have survived after the DDoS attack, becomes an Acki-Nacki the attack is deemed unsuccessful. To finalize the block, the BK must collect Attestations from A BKs and check for the absence of Nacks.

2) Security Assumptions: 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, get the number of malicious nodes at which the attack will be successful:

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

At the same time, if all malicious BKs disconnect and stop sending messages, the network will stop, since it will not be able to collect enough Attestations. The same situation may occur if all malicious BKs disconnect or reject Nack's for a malicious block during the execution of the Joint Committee function. It happens when:

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

Rewriting this inequality:

$$M \geq N + 1 - \max(A, J) \tag{3}$$

B. Constraints on the Number of Spammed Block Keepers

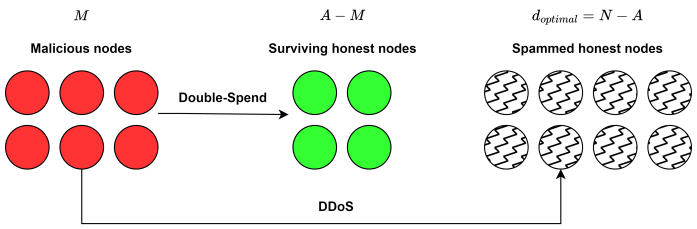

Fig. 5: Scheme of the optimal DDoS Attack.

Since consensus requires A Attestations, among the $(N - d)$ BKs that continue to function after the DDoS attack, A BKs must be able to gather the required Attestations.

Therefore,

$$N - d \geq A \tag{4}$$

For malicious BKs to launch a successful attack, it is desirable to spam as many honest BKs as possible.

From this, it follows that

$$d_{optimal} = N - A \tag{5}$$

C. 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:

$$\mathbf{P}(\xi = k) = C_N^k \cdot \left(\frac{v}{N}\right)^k \cdot \left(1 - \frac{v}{N}\right)^{N-k} \tag{6}$$

We find that the random variable ξ follows a Binomial distribution, and its expected value is:

$$\mathbb{E}(\xi) = v \tag{7}$$

In other words, v is precisely the mathematical expectation of the number of BKs that become an Acki-Nacki per block.

D. 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:

$$p(N, M, d, v) = \left(1 - \frac{v}{N}\right)^{N-M-d}. \tag{8}$$

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

$$p(N, M, A, v) = \left(1 - \frac{v}{N}\right)^{A-M}. \tag{9}$$

E. Too many blocks attack

The "Too many block attack" is the case when a malicious BP generates so many blocks that it is becoming impossible to verify them and produce Ack/Nacks in time. The attacking BP can spam the network with valid blocks until producing a malicious one. The attack is mitigated with BKs' requirement to wait at least 330 ms. before releasing Attestation between any two blocks regardless of the time it received the block.

F. Block with too complex execution

If a BP makes a block with too complex execution it can try to make Verifiers delay verification for so long, the min. Time will pass and no Ack/Nack will be transmitted.

To mitigate such an attack the Verifier which executes such a block will stop after 330 ms and send a special Nack with message "too complex". The committee will check the block and penalize BP if needed.

G. Safety Analysis

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

$$\mathbf{P}(\text{at least one successful attack}) = 1 - (1 - p)^R. \tag{10}$$

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 malicious BKs. The number

of malicious BKs may be at a maximum of $A-1$. If $A > 1/2$, 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.

Below are plots illustrating 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.

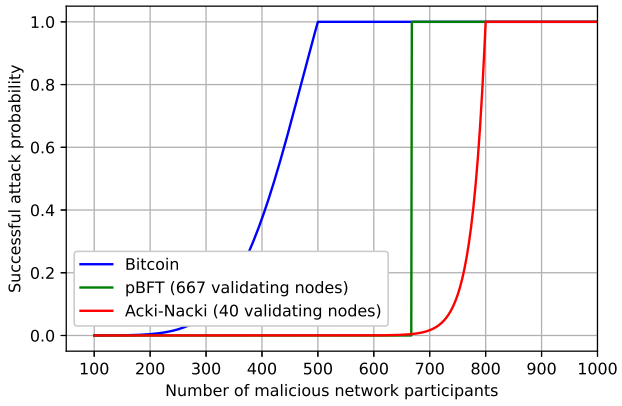

Fig. 6: Comparison of successful attack probabilities in Bitcoin, pBFT and Acki Nacki.

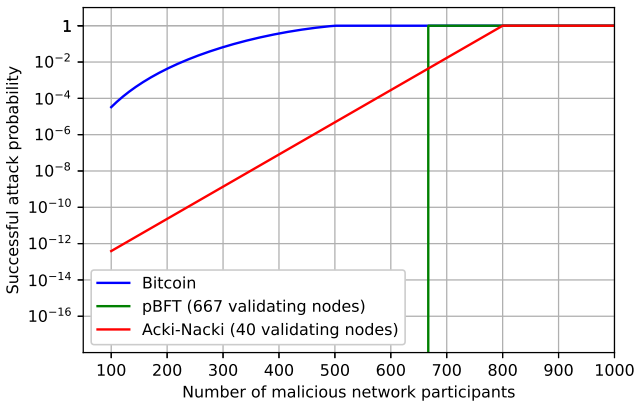

Fig. 7: Fig. 6 with log-scaled y-axis.

As observed, Acki Nacki provides significantly higher security guarantees compared to Bitcoin. Furthermore, this holds true when compared to pBFT in scenarios where the number of malicious network participants exceeds $2/3$ of the total network participants. Since we went further and 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.

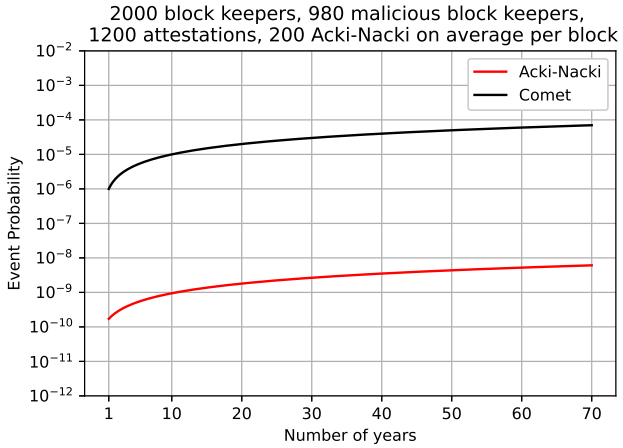

Fig. 8: Comparison plot of the successful attack probability in the Acki Nacki consensus protocol with the global catastrophe probability.

As we can see: it is more likely that a comet will fall in the coming years and destroy life as we know it than a malicious BK attacking the network successfully.

H. Performance Analysis

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

We do not discuss in this paper the state sharding solutions but it is quite easy to imagine an Acki Nacki sharded design: some BKs can decide not to store a state which belongs to certain address space and the only remaining question in asynchronous system would be — how the messages that are passed from one Account to another residinging in two shards would execute by the BP and Verified if the BP and Verifier are not in possession of one of the participating Accounts state.

With sharded design there is no theoretical limit to the Acki Nacki network throughput. Without sharded design, taking into account modern computer hardware and datacenter Internet connection we calculate a practical limit of 250,000 transactions per second of a minimal 500 byte messages with less than a 1 sec finality. With sharding enabled the protocol

can scale to millions of transactions of any complexity just by adding computing resources which makes it comparable with centralized cloud services.

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

In total the following messages are sent: The block sent from BP to BKs, the Attestations sent from BKs to BP and the Ack/Nack messages sent from several chosen Acki-Nacki to BKs. Here the positivistic scenario ends. The Nack message and accidental Forks will trigger more messages, but as we showed above the Nack message is highly improbable and Forks are rare events. Most of the time the network will operate by sending just 3 types of Messages where total of all messages sent equals $2 \cdot N + v \cdot (N - 1)$.

V. Conclusion

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

References

[1] M. Diffie, M. Hellman, "New Directions in Cryptography," November 1976. https://www-ee.stanford.edu/~hellman/publications/24.pdf

[2] R. Merkle, "Secure Communications Over Insecure Channels," (April 1978) https://dl.acm.org/doi/10.1145/359460.359473

[3] D. Boneh, V. Shoup, A graduate course in applied cryptography. Draft 0.5 (2020). https://dlib.hust.edu.vn/bitstream/HUST/18098/3/OER000000253.pdf

[4] M. Castro, B. Liskov, "Practical Byzantine Fault Tolerance," in Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99), New Orleans, Louisiana, (February 1999). https://pmg.csail.mit.edu/papers/osdi99.pdf

[5] N. Satoshi, "Bitcoin: A peer-to-peer electronic cash system," 2008. https://bitcoin.org/bitcoin.pdf

[6] A. Ozisik, L. Pinar, B. Neil, "An Explanation of Nakamoto's Analysis of Double-spend Attacks," in CoRR. — Vol. abs/1701.03977. — arXiv : 1701.03977. (2017) https://arxiv.org/pdf/1701.03977.pdf

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper 151.2014, p. 32, 2014. https://ethereum.github.io/yellowpaper/paper.pdf

[8] V. Buterin, "A Next Generation Smart Contract y Decentralized Application Platform, " January, 2015. https://ethereum.org/en/whitepaper

[9] S. King, S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," (August 19, 2012) https://decred.org/research/king2012.pdf

[10] G. Danezis, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, "Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus," in Proceedings of the Seventeenth European Conference on Computer Systems, pp. 34-50. (March, 2022) https://arxiv.org/pdf/2105.11827.pdf

[11] I. Grigg, "Eos-an introduction," White paper, 2017. https://whitepaperdatabase.com/eos-whitepaper

[12] L.M. Bach, B.Mihaljevic, M. Zagar, "Comparative analysis of blockchain consensus algorithms," in 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, (2018). https://ieeexplore.ieee.org/document/8400278

[13] Z.K. Sun, J. Chang, N. Zhu et all. "Rangers protocol 2.0.," p. 59, 2022. https://rangersprotocol.obs.ap-southeast-1.myhuaweicloud.com/Navigation/RangersProtocolWhitepaper.pdf

[14] P. Berrang, P. von Styp-Rekowsky, M. Wissfeld, B. França, R. Trinkler, "Albatross–an optimistic consensus algorithm," in 2019 Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 39-42, IEEE. (June, 2019). https://ieeexplore.ieee.org/document/8787561

[15] W. Zhong, C. Yang, W. Liang, J. Cai, L. Chen, J. Liao, N. Xiong, "Byzantine Fault-Tolerant Consensus Algorithms: A Survey," in Electronics 12, no. 18 (2023): 3801. https://www.mdpi.com/2079-9292/12/18/3801

[16] S. Tse, M. Liu et al. "Harmony Technical Whitepaper-version 2.0.," 2023. https://harmony.one/whitepaper.pdf

[17] S. Goldwasser and S. Micali, "Probabilistic Encryption," Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984 https://mit6875.github.io/PAPERS/probabilistic_encryption.pdf

[18] J.-P. Martin, L. Alvisi, "Fast Byzantine Consensus," in 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005 https://ieeexplore.ieee.org/document/1467815

[19] Tung-Wei Ku, and Kung Chen, "No Need for Recovery: A Simple Two-Step Byzantine Consensus," November 2019 https://arxiv.org/pdf/1911.10361.pdf

[20] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13." Whitepaper, 2018. https://solana.com/solana-whitepaper.pdf

[21] 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

[22] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in 38th ACM symposium on Principles of Distributed Computing (PODC '19, July 29–August 2, 2019, Toronto, ON, Canada) https://dl.acm.org/doi/pdf/10.1145/3293611.3331591

[23] The Zilliqa Team, "The Zilliqa technical whitepaper," August 2017 https://docs.zilliqa.com/whitepaper.pdf

[24] V. Buterin et all, "Ethereum roadmap, What about sharding?" 2022 https://ethereum.org/en/roadmap/#what-about-sharding

[25] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, Y. X. Zhang, "Combining GHOST and Casper," 2020 https://arxiv.org/pdf/2003.03052.pdf

[26] N. Durov, "Telegram Open Network","Telegram Open Network Virtual Machine." Open Netw., White Paper, Mar 2020. https://ton.org/tblkch.pdf

[27] I. Polosukhin, V. Frolov, et all. "The NEAR White Paper," 2023 https://near.org/papers/the-official-near-white-paper

[28] The MultiversX Team, "MultiversX, A Highly Scalable Public Blockchain via Adaptive State Sharding and Secure Proof of Stake [Technical whitepaper — release 2 — revision 2]," June 19, 2019 https://files.multiversx.com/multiversx-whitepaper.pdf

[29] D. Boneh, M. Drijvers, G. Neven, "BLS multi-signatures with public-key aggregation," in ASIACRYPT, 2018 (2018). https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html

[30] R. Yang, "Analysing the efficiency and robustness of gossip in different propagation processes with simulations," in 2019 4th International Seminar on Computer Technology, Mechanical and Electrical Engineering (ISCME), pp. 13-15, December 2019. https://iopscience.iop.org/article/10.1088/1742-6596/1486/3/032001/pdf

[31] L.M. Bach, M. Branko, Z. Mario. "Comparative analysis of blockchain consensus algorithms," in 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2018.

[32] D. Ongaro, J. Ousterhout, "In search of an understandable consensus algorithm (extended version)," Proceeding of USENIX annual technical conference, USENIX ATC, pp. 19-20, 2014.

[33] B. Chase, E. MacBrough, "Analysis of the XRP Ledger Consensus Protocol," https://arxiv.org/pdf/1802.07242.pdf, 2018