# Fast Finality and Resilience to Long Range Attacks with Proof of Space-Time and Casper-like Finality Gadget

Alex Skidanov
Near Protocol
/AlexSkidanov
alex@nearprotocol.com

October 2019

**Abstract**

We present a design for a fork choice rule, Sybil resistance mechanism and finality gadget that collectively provide fast finality, resistance to long range attacks, and discourage pooling. The design is based on both proof of stake and proof of space time, and uses a finality gadget similar to Casper FFG. Once a block is finalized, it is irreversible in the short term unless more than $\lceil n/3 - 1 \rceil$ of total stake is slashed. Once the staked tokens are released, the block cannot be reverted unless malicious actors control close to 50% of the total space available in the system.

## 1 Introduction

In this paper we introduce a fork choice rule and a Sybil resistance mechanism that are currently proposed to be used in NEAR Protocol[1]. We compare it to three approaches to Sybil resistance and fork choice rules: Proof-of-Work with the heaviest chain, as initially introduced in the Bitcoin paper [1], (Delegated) Proof-of-Stake with the longest chain, for example see Ouroboros [2], and (Delegated) Proof-of-Stake with BFT consensus, for example see ByzCoin [3].

We begin by surveying the existing fork choice rules and Sybil resistance mechanisms on several dimensions.

[1]This is an ongoing research. NEAR will initially launch without Proof-of-Space-Time.

## 1.1 Desired Properties

### 1.1.1 Resilience to Long-Range Attacks

Proof-of-Work with the heaviest chain fork choice rule has a very desirable property that no matter what malicious actors do, unless they control more than 50% of the hash power, they cannot revert a block that was finalized sufficiently long ago.

Proof-of-Stake systems do not have the same property. In particular, after the block producers that were creating blocks at some point in the past get their staked tokens back, the keys that they used to create blocks no longer have value for them. An adversary can attempt to buy such keys for a price significantly lower than the amount of tokens that was staked when the key was used to produce blocks. Since, unlike in Proof-of-Work, Proof-of-Stake has no mechanism to force a delay between produced blocks, the adversary can then in minutes create a chain that is longer than the canonical chain, and have such chain chosen by the fork choice rule.

There are two primary ways to get around this problem:

**Weak subjectivity**. Require that all the nodes in the network periodically check what is the latest produced block, and disallow reorgs that go too far into the past. If the nodes check the chain more frequently than the time it takes to unstake tokens, they will never choose a longer chain produced by an adversary who acquired keys for which the tokens were unstaked.

The weak side of the weak subjectivity approach is that while the existing nodes won't be fooled by the attacker, all the new nodes that spin up for the first time will have no information to tell which chain was created first, and will choose the longer chain produced by an attacker. To avoid it, they need to somehow learn off-chain about the canonical chain, effectively forcing them to identify someone whom they trust in the network.

**Forward-secure keys**. Another approach is to make the block producers destroy the keys that they used to produce blocks immediately or shortly after the blocks were produced. This can be done by either creating a new key pair every time the participant creates blocks, or by using a construction called *Forward-secure keys*, which allows a secret key to change while the public key remains constant.

This approach relies on nodes being honest and following protocol strictly. There's no incentive for them to destroy their keys, since they know in the future someone might attempt to buy them, thus the key has some non-zero value. While it is unlikely that a large percentage of block producers at a certain moment all decide to alter their binaries and remove the logic that wipes the keys, a protocol that relies on the majority of participants being *honest* has different security guarantees than a protocol that relies on the majority of participants being *reasonable*. Proof-of-work works for as long as more than half of the participants are *reasonable* and do not cooperate, and it is desirable to have the fork choice rule and a Sybil resistance mechanism that have the same property.

We propose to use Proof-of-Space-Time (see e.g. [4]) that, like Proof-of-Work, makes use of some scarce resource (in this case space) and requires the adversary to control more than 50% of that resource in the network to carry out a reorg, which provides the same security guarantees against long range attacks as Proof-of-Work.

#### 1.1.2 Environment Friendliness

Bitcoin and Ethereum miners spend multiple terrawatt hours of electricity per year to carry out Proof-of-Work computation, severely damaging the environment.

While Proof-of-Stake improves significantly on it and has practically no imprint on the environment, it has other disadvantages described in the surrounding subsubsections, since participants do not use any scarce resource to secure the chain.

Proof-of-Space-Time is a significantly more environment-friendly version of the Sybil resistance mechanism than Proof-of-Work that does take advantage of a scarce resource, namely space, but doesn't consume much electricity, since hard drives do not need electricity to *store* information.

#### 1.1.3 Time to Finality

Proof-of-Work systems, such as Bitcoin or Ethereum, and non-BFT Proof-of-Stake systems require one to wait for a substantial time before they become sufficiently certain that a block that includes a transaction they care about is final. In Bitcoin it is common to wait for six blocks, which takes on the order of one hour.

Proof-of-Stake systems that use a BFT consensus to finalize each block, on the other hand, have very fast finality. Once a block is produced and is agreed upon with the BFT consensus, unless a large percentage of the stake is slashed, the block is irreversible.

#### 1.1.4 Cost to Revert Finalized Block

Proof-of-Stake systems that use BFT consensus have significantly higher cost of reverting a recently finalized block than Proof-of-Work or non-BFT Proof-of-Stake systems. Specifically, within a particular epoch (where epoch is loosely defined as a time span during which the validator set was constant) for a block to be reverted one third of the stake needs to be slashed, which often means that something on the order of 10%-20% of all the circulating tokens need to get burnt. In the Proof-of-Work system just the recent rewards need to be paid for, which is generally significantly smaller amount of value.

However, as discussed above, reverting a block that was finalized long ago could cost very little in a Proof-of-Stake system, while in Proof-of-Work system the cost of the reorg increases over time.

In the construction we propose, creating a short reorg once a block is finalized would cost at least one third of the total stake slashed, similar to Proof-of-Stake

with a BFT consensus. On the other hand, creating a long reorg will cost the accumulated rewards for all the space-time proofs. The cost of the latter, while it is less than the cost of making a short range reorg, increases over time after the drop that occurs when finalized block tokens are unstaked.

### 1.1.5 Availability

Proof-of-Work and Proof-of-Stake systems that use some sort of heaviest chain fork choice rule remain available even if a large percentage of block producers simultaneously disappear. Achieving the same property in constructions that use BFT consensus is significantly harder.

BFT consensus protocols and BFT finality gadgets in the context of blockchains are primarily designed in the context of a single permissioned set of $n$ validators that doesn't change over time and has at least $\lfloor 2n/3 + 1 \rfloor$ validators online at any point. We discuss in detail the implications of validator set changes and possibility of more than $\lceil n/3 - 1 \rceil$ validators simultaneously going offline for safety and liveness of blockchain protocols in section 1.2.1.

### 1.1.6 Incentives for Pooling

Existing Proof-of-Work approaches provide high incentive for miners to pool. Since each block produced is a lottery, and the number of blocks produced per unit of time is relatively low, for most miners the expected time before they win the lottery at least once is huge. The miners would gladly trade a small percentage of the expected value for reduced variance, and thus they choose to join a pool instead of mining themselves.

Miners that mine via pools often get the hash to mine on top from the pool instead of watching the network themselves and choosing the heaviest chain known to them. Thus, a set of pools that collectively control more than half of the hash power can perform short time reorgs by sending a hash that doesn't correspond to the heaviest chain to their workers.

Currently for both Ethereum and Bitcoin the number of pools that need to collude to perform such an attack doesn't exceed five, thus posing major centralization concerns.

Proof-of-Space-Time differs from Proof-of-Work in that designing it as a lottery is very hard. The particular construction that we present here doesn't have a lottery, and makes pooling of Proof-of-Space-Time practically impossible.

### 1.1.7 Permissionlessness

While a contentious topic, it can be argued that Proof-of-Stake systems are not permissionless in the same sense as Proof-of-Work systems, since for someone to become a block producer, they generally need to submit a staking transaction, and the transaction needs to be settled, thus effectively requiring the block producer to get permission from the existing blockchain users to start producing blocks.

In the context of NEAR Protocol the transactions are not included in the blocks, and are instead grouped into so-called chunks that blocks consist of, see the details in [5]. Producing invalid chunks must be a slashable behavior, and thus we depend on Proof-of-Stake for the chunk production, and cannot have permissionlessness in the Proof-of-Work sense. As such, having permissionlessness was not a design goal for the construction presented here.

The construction can be changed in such a way that the block production is done by the participants identified by Proof-of-Space-Time instead of the participants identified by Proof-of-Stake, thus at least partially introducing permissionlessness in the same sense as in Proof-of-Work, but this is outside of the scope of this document.

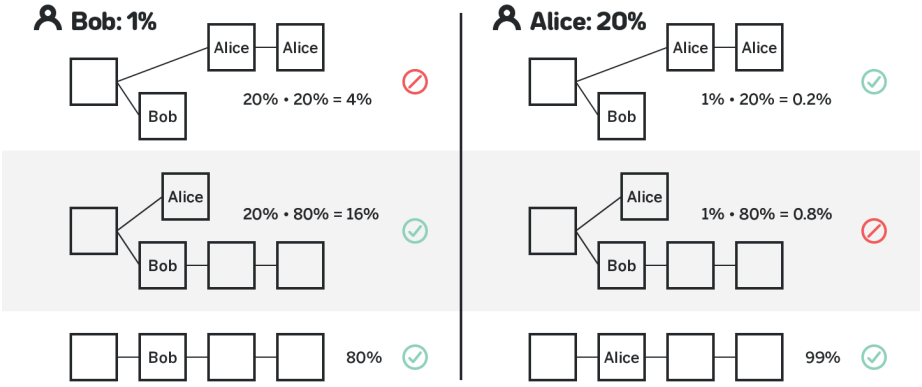


Figure 1: Large Proof-of-Work miner censoring a smaller miner

We note here that the permissionlessness of Proof-of-Work is also somewhat exaggerated. Consider figure 1. A malicious miner or a pool that controls a large percentage of the hash power, say 20%, can choose to censor small block producers that have, say, 1% of the hash power. By censoring their blocks the malicious miner loses on the order of 1% reward, while the target miner loses on the order of at least 4% of the reward (and likely more). Assuming that the margin of mining on top of the cost of electricity is less than 4% the target miner will be losing money, and will eventually be forced to stop mining, effectively increasing the reward for all the remaining miners, including the attacker, by 1% in the future. Thus, big miners can and have an incentive to censor small players, and thus arguably Proof-of-Work systems are not truly permissionless either.

## 1.2 BFT Finality, Validator Rotation and Availability

When desigining Proof-of-Stake blockchains that rely on a BFT consensus it is generally desirable that if a block is finalized by such a consensus, it cannot get

reverted unless a very large percentage of the stake is slashed.

Many BFT protocols provide this guarantee for as long as the validator set is constant and at least $\lfloor 2n/3+1 \rfloor$ of validators are online at any given moment.

### 1.2.1 BFT Finality and Validator Rotation

The problem with validator set rotation in general is that, if two consecutive validator sets do not have a large overlap, the latter validator set can pretend, without any risk of being slashed, that they didn't see a block finalized by the previous validator set. This problem has at least two different solutions. One solution is to prevent large validator set changes by only allowing up to some percent of validators to change between two consecutive validator sets. Another approach is to force both validator sets to finalize a block during a transition between the two sets.

### 1.2.2 BFT Finality and Availability

A bigger and more fundamental problem is how to handle situations when for some reason more than $\lceil n/3-1 \rceil$ validators simultaneously went offline. For example, in a blockchain in which there are multiple clients this can occur if one of the major clients hits a bug and a large percentage of validators simultaneously disconnect.

We survey here three approaches used by Cosmos, Polkadot and Ethereum Serenity. Cosmos uses BFT consensus as the primary fork choice rule: the canonical chain is the longest chain that only consists of finalized blocks. Polkadot and Ethereum Serenity use a non-BFT fork choice rule (respectively BABE and LMD GHOST), as well as a BFT finality gadget (respectively GRANDPA and Casper FFG).

- **Cosmos approach**. Cosmos heavily favors consistency, and thus if more than $\lceil n/3-1 \rceil$ validators simultaneously go offline, the chain stalls. If the validators have crashed unrecoverably, the only way to resume the chain is via a hard fork.

- **Polkadot approach**. In Polkadot the underlying non-BFT consensus BABE will continue operating even if a very large percentage of validators went offline, but the BFT finality gadget GRANDPA stalls until the validators of the last set that needed to finalize a block get back online. Similarly, if the validators have crashed unrecoverably, currently the finality gadget can only be resumed via a hard fork, but the chain will continue producing blocks via BABE. Since in Polkadot many components rely on blocks being finalized, this approach heavily favors consistency.

- **Ethereum Serenity approach**. Serenity heavily favors availability. In Serenity the underlying non-BFT consensus LMG GHOST will continue producing blocks even if a very large percentage of validators goes offline. Their BFT finality gadget Casper FFG will stall if more than $\lceil n/3-1 \rceil$

validators are offline, however the offline validators will slowly lose their stake, and at some point will drop out, bringing the system into a state in which at least $\lfloor 2n/3 + 1 \rfloor$ validators are online, at which point Casper FFG will resume finalizing blocks. Note that since the two validator sets can greatly differ in this case, a block that was previously finalized can now become orphaned with nobody being slashed, as shown on figure 2.

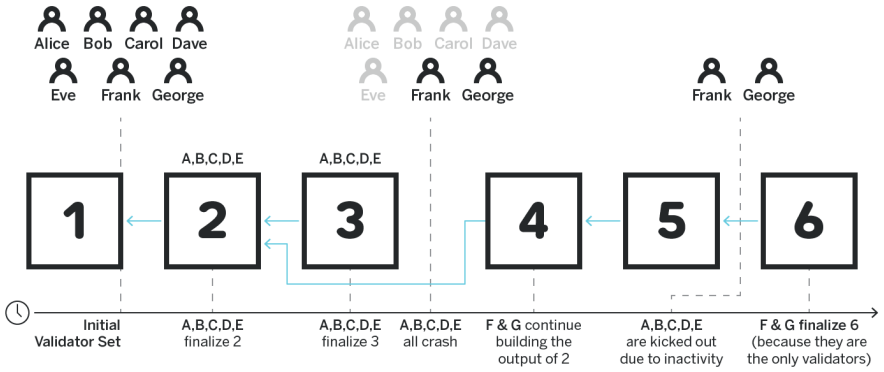


Figure 2: Availability-favoring finality gadget can have finalized blocks reverted without any entity getting slashed

Generally we argue that favoring availability over consistency is preferred, since clients that want to have consistency can always choose to locally treat an available chain as consistent. For example, in the case of Ethereum Serenity, a client can choose not to respect any blocks that are produced by a committee that differs greatly from the last committee which finalized a block (and thus locally stall the chain). In such an event if a sufficient number of clients are concerned with the committees being different, and have a reason to believe that a finalized block was reverted in the process, they can choose to reach a social consensus and still perform a hard fork.

In this document we present an approach that favors consistency, and leave exploring exact changes necessary to make it favor availability for future work, see section 4.1.

# 2 Construction

In this section we present our construction that relies on Proof-of-Space-Time for long term security and a Casper-like finality gadget for short term security.

## 2.1 Block Production

In the construction we propose, block production is split into epochs. Each epoch has a constant set of participant that are responsible for block production, who we refer to as *block producers*. Block producers have some amount of tokens staked, and in our construction the stake for all the block producers is equal. While in practice it is expected that the block producer sets will greatly overlap between epochs, we do not make such an assumption.

The block producers within a particular epoch propose blocks on some fixed schedule. One such schedule could be to split the epoch into 10 second intervals and assign one block producer to each interval.

When a block producer produces a block, they request that all other block producers provide an approval for the previous block upon which the newly produced block will be built. They then include these approvals into their own block. Nightshade [5], the sharding design behind NEAR Protocol, requires that at least half of the block producers in the current epoch provide an approval, but the construction presented here doesn't rely on such an assumption.

## 2.2 Proof-of-Space-Time

Besides the block producers (for whom the Sybil resistance mechanism used is Proof-of-Stake) there's another type of participant that we call *miners*. Miners do not need to have any stake in the system, and are tasked with carrying out Proof-of-Space-Time computation. Unlike Proof-of-Work that runs a lottery on top of each block, Proof-of-Space-Time in our construction is not a lottery, and assuming the system estimated the target complexity of Proof-of-Space-Time correctly, all the proofs generated will be included.

Specifically, the miners compute the Proof-of-Space-Time seeded by the first block of a particular epoch, as shown on figure 3, thus endorsing an epoch, not a block. Each block produced by a block producer has some number $s, s > 1$ of slots for space-time proofs that endorse the previous epoch. A block producer gets a reward for each included space-time proof in their block, thus they have an incentive to include as many of these proofs as possible (up to their allowed slot limit $s$).

At the end of the epoch if the average number of proofs per block exceeds one, the target complexity of Proof-of-Space-Time is slightly increased, and if it is lower than one, the target complexity is decreased. Thus, assuming the available space for Proof-of-Space-Time doesn't suddenly increase by a factor of $s$ or more, and the complexity settled at a reasonable value, all the proofs computed for a particular epoch will be included in the blocks in the next epoch. Note that participants that do not have sufficient space for the target Proof-of-Space-Time complexity cannot benefit in any way from pooling, and thus there's no incentive at all for miners to pool.

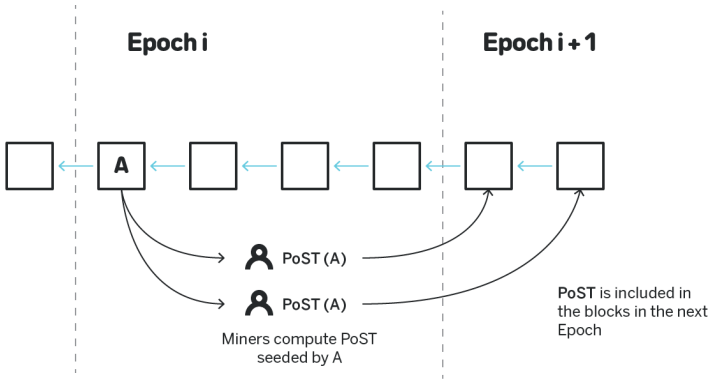


Figure 3: The timeline of Proofs-of-Space-Time computation and inclusion

## 2.3 Approvals

Approvals in our construction have the same purpose as votes in Casper FFG [6]. An approval by a block producer $v$ is an ordered tuple

$$\langle v, p, r \rangle$$

where $p$ is the block that is being approved, $r$ is a so-called *reference block*.

The reference block must be in the ancestry of $p$ (we say that a block $b$ is in the ancestry of another block $a$ if $b$ is equal to $a$, if it is the previous block of $a$, or if $b$ is in the ancestry of the previous block of $a$).

We define a score $s(b)$ and a weight $w(b)$ of a block in section 2.5. For any two approvals any particular block producer issues

$$\langle v, p_1, r_1 \rangle$$

and

$$\langle v, p_2, r_2 \rangle$$

it must be that

1. $r_1 \neq r_2 \wedge s(r_1) > s(p_2) \wedge w(r_1) > w(p_2)$, or
2. $r_1 \neq r_2 \wedge s(r_2) > s(p_1) \wedge w(r_2) > w(p_1)$, or
3. $r_1 = r_2 \wedge (p_1 \in A(p_2) \vee p_2 \in A(p_1))$

where $A(b)$ denotes all the blocks in the ancestry of $b$.

In other words, if two approvals have the same reference block, the blocks they approve must be on the same chain, and if they have different reference

blocks, the ranges of scores and weights of $(r, p)$ must not have any points in common, and moreover the approval that has higher scores must also have higher weights. Creating two approvals that violate these rules is a slashable behavior.

When a block producer $v$ creates a block $b$, they request approvals for the previous block $p$ that $b$ is built on top of. Another block producer $v'$ is expected to provide such an approval if $p$ is the tip of the canonical chain chosen by the fork choice rule (see section 2.5 below). If the last block the block producer approved is in the ancestry of $p$, the block producer should use the same reference block as in the last approval. Otherwise the reference block must be the lowest score block in the ancestry of $p$ that has higher score and weight than the last block the block producer approved. Such approvals by construction cannot trigger the two slashable conditions listed above. See figure 4.

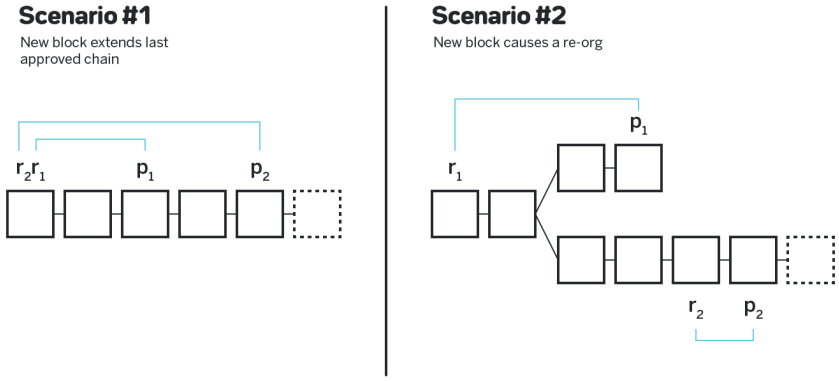


Figure 4: Approvals for a block that extends the last known canonical chain and a block that causes a reorg

## 2.4 Finality Gadget

In this section we describe a Casper-like finality gadget, that we call Casper NFG[2].

For an approval $a = \langle v, p, r \rangle$ and a block $b$ we call $a$ a pre-vote on $b$ iff

1. $b$ is in the ancestry of $p$;
2. $r$ is in the ancestry of $b$.

[2]Nightshade Finality Gadget. Also a play of words Naughty Finality Gadget to contrast with Friendly Finality Gadget. Coincidentally, NFG is also a reference to the biggest problem in the space today, namely the current adoption of blockchain protocols in the wider community, with N and G standing for "no" and "given" correspondingly.

We say a block $b_p$ has a *quorum pre-vote* for block $b$, if $b$ is in the ancestry of $b_p$ and there are $\lfloor 2n/3 + 1 \rfloor$ of block producers for whom there's an approval

$$a = \langle v, p, r \rangle$$

included in a block in the ancestry of $b_p$ such that $p$ is in the ancestry of $b_p$ and $a$ is a pre-vote on $b$.

For an approval $a = \langle v, p, r \rangle$ and a block $b$ we call $a$ a pre-commit on $b$ iff

1. $b$ is in the ancestry of $p$;
2. $r$ is in the ancestry of $b$;
3. There's a block $b_p$ in the ancestry of $p$ that has a quorum pre-vote on $b$.

We say a block $b_p$ has a *quorum pre-commit* for block $b$, if $b$ is in the ancestry of $b_p$ and there are $\lfloor 2n/3 + 1 \rfloor$ of block producers for whom there's an approval

$$a = \langle v, p, r \rangle$$

included in a block in the ancestry of $b_p$ such that $p$ is in the ancestry of $b_p$ and $a$ is a pre-commit on $b$.

As will be shown in section 3.1, no two blocks, assuming neither is in the ancestry of the other, can have a quorum pre-commit by the same set of block producers, unless at least $\lceil n/3 \rceil$ of the block producers are slashed. Thus, a block that has a quorum pre-commit in practice can be considered final. We analyze how this finality gadget works in the context of rotating block producer sets in the section 2.7.

## 2.5 Fork Choice Rule

We first define concepts of the *weight* and *score* of a block, and then define the fork choice rule.

We define **block weight** of a block $b$ (denoted as $w(b)$) as the sum of all the complexities of the space-time proofs included in all the blocks in the ancestry of $b$.

Let $HQV(b)$ be the heaviest (in terms of weight) block for which a quorum pre-vote exists in the ancestry of $b$.

The **score** of a block (denoted as $s(b)$) is then:

$$s(b) = w(HQV(b))$$

To choose a canonical chain among a set of chains, one first computes the *score* of the tip of each chain, and then chooses the chain with the higher score of the tip.

If the scores are equal for two chains, the exact fork choice rule (we refer to such a fork choice rule as the *low-level* fork choice rule) is not as important for the rest of the construction. We do a survey of possible low-level fork choice rules in the next section.

## 2.6 Survey of Low-Level Fork Choice Rules

### 2.6.1 LMD GHOST

LMD GHOST (Latest Message Driven Greedy Heaviest Observed Subtree) works on a tree of blocks that originates at the last block finalized by the finality gadget. LMD GHOST first for each block producer identifies the highest weight block approved by such block producer. It then starts from the last finalized block and builds the canonical chain one block at a time. At each step for each child of the last added block it computes the number of block producers, whose heaviest approved block is in the subtree of the child. It then chooses the child with the largest number of such block producers to the canonical chain, and continues until the last added block has no children.

LMD GHOST has a property that it's "sticky". In other words, if a particular block $b$ has a large percentage of the block producers having their heaviest approved block in the subtree of $b$, for as long as most block producers are honest and build on top of the chain chosen by the LMD GHOST, $b$ will remain on the canonical chain.

The disadvantage of using LMG GHOST in the context of the presented construction is that if the finality gadget stops finalizing blocks, the system becomes vulnerable to the long range attacks again. We note that the finality gadget is unlikely to stop finalizing blocks for more than several epochs, so in practice it is not a big concern. In particular, if a large percentage of block producers go offline, they will be removed from the block producers set in a few epochs, and the remaining block producers will be sufficient to start finalizing blocks again.

### 2.6.2 Highest Weight Chain

The alternative to LMD GHOST is to just use the weight of the tip of the chain as the tie breaker if the scores of two chains are equal.

The Proof-of-Space-Time model we presented has a major difference from more classic Proof-of-Work. Specifically, a particular proof is computed based on the hash as of two epochs ago, and as such can be included in all the conflicting chains that diverged during the previous or the current epoch. Thus, a malicious actor with a very small amount of space available can create a short fork, include all the proofs of space time computed by all the honest actors and a few proofs computed by themselves, and have a heavier chain.

One way it can be addressed is if the miners, when submitting their space-time proofs, include not only the last epoch hash that was used as the input to the proof computed, but also the last block they know to be on the canonical chain at the time of the submission. Then the submitted proof can only be used towards the weight of the chain that extends such block. The reward for the proof is slightly delayed, and if the same proof is used with multiple previous blocks, a cryptographic proof of such existence can be used to slash the reward.

This construction is not vulnerable to long range attacks. We do not deeply analyze the security of such approach, but note that it is vulnerable to adaptive

corruption: an adversary can attempt to pay a small premium to the miners to buy their existent proofs and create a heavier chain than the canonical chain, causing a short reorg.

## 2.7 Block Producers Rotation

We split block production into epochs, where the length of an epoch is some constant number of blocks. In practice we tune the number of blocks in an epoch in such a way that the epochs last for approximately one day.

During a particular epoch $k$ participants can stake tokens to express their desire to become block producers, or issue a transaction that expresses their desire to stop producing blocks. At the end of epoch $k$, transactions from epoch $k-1$ are applied and any new participants that staked tokens in epoch $k-1$ are added to the pool of participants available for block production. At this time, participants that expressed their desire to stop producing blocks during epoch $k-1$ are removed from the pool. Then the block producers for epoch $k+1$ are chosen from the pool according to some algorithm, the exact details of which are not important for the construction. The algorithm that we use in NEAR, called *thresholded proof-of-stake*, selects the largest threshold $t$ such that

$$\sum_{v \in V} \lfloor s_v / t \rfloor \geq n$$

Where $n$ is the number of block producer seats, $V$ is the pool of participants available for block production, and $s_v$ is the stake of the participant $v$. It then assigns each participant with $\lfloor s_v/t \rfloor$ block producer seats.

Since the block producers for an epoch are known almost from the very beginning of the previous epoch, once the block producers for the next epoch are known, we make them also provide approvals for blocks. We then make clients consider a block final if and only if it is independently finalized by the block producers of the current epoch and of the next epoch.

If a block producer failed to provide approvals for some predefined percentage $pk$ of blocks during a particular epoch (say $pk = 80\%$), it is treated as if they submitted a transaction wishing to get excluded from the block producers set, and either a small percentage of their stake is slashed, or they are banned from rejoining the block producers set for some number of epochs.

Thus, in the case where more than $\lceil n/3 - 1 \rceil$ block producers go offline and the finality gadget is stalled, offline block producers will be removed from the block producers set and the finality gadget will start finalizing blocks again within two epochs. We note, however, that if a particular block producers set hasn't finalized a single block in a particular epoch, it is possible that in the future two or more conflicting blocks will be finalized without any participant being slashed.

A client can then choose to treat the chain as either available or consistent. An availability-favoring client will respect all blocks finalized by the finality gadget. A consistency-favoring client will only respect blocks, such that for each epoch up to and including the epoch in which the block is produced,

at least one block was finalized by both the current epoch and the next epoch block producer sets. We discuss the exact finality guarantees and what resources malicious actors need to possess to create short term or long term forks in section 3.3.

# 3 Analysis

## 3.1 Safety of the Finality Gadget

In this section we prove that two blocks such that neither is in the ancestry of the other cannot both be finalized (as in, have a quorum pre-commit) by the same block producers set, *unless a block that is not in the ancestry of at least one of the two given blocks was finalized by some other block producers set.* We discuss the implications of the possibility of a conflicting block being finalized by another block producers set in section 3.3.

For a chain with a tip $t$ and a block $b$ that is finalized in such a chain, we define as $QV(t, b)$ the lowest score block in the ancestry of $t$ that has a quorum pre-vote on $b$.

**Lemma 1.** *If a block $b_1$ is finalized in a chain with a tip $t_1$, no block $b_2$ such that $w(b_2) > w(b_1)$ can have a quorum pre-vote by the same block producers set in a chain with some tip $t_2$ if neither $b_1$ nor $b_2$ are in the ancestry of one another, unless more than $\lceil n/3 - 1 \rceil$ block producers committed a slashable act, or a block heavier than $w(b_1)$ that is not in the ancestry of $b_1$ was finalized by another block producers set.*

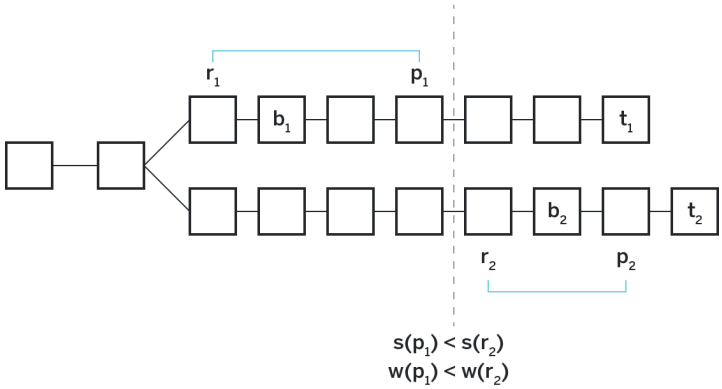


Figure 5: If a block $b_1$ is finalized on one chain, no heavier block can get a quorum pre-vote on any other chain

*Proof.* Say it is not the case, and there's such block $b_2$. Consider such block $b_2$ with the lowest height (where height is defined as the number of blocks in its ancestry).

By definition of block finality, there's $\lfloor 2n/3 + 1 \rfloor$ block producers that have an approval of a form $\langle v, p_1, r_1 \rangle$ such that $QV(t_1, b_1)$ is in the ancestry of $p_1$. Note that since $QV(t_1, b_1)$ has a quorum pre-vote on $b_1$, the score of $p_1$ is at least $w(b_1)$.

Since $b_2$ has a quorum pre-vote in the chain originating at $t_2$, there are $\lfloor 2n/3 + 1 \rfloor$ block producers that have an approval of a form $\langle v, p_2, r_2 \rangle$ such that $b_2$ is in the ancestry of $p_2$ and $r_2$ is in the ancestry of $b_2$. At least $\lceil n/3 \rceil$ block producers have approvals of both forms.

Consider such block producer. Recall that $w(b_1) \leq w(b_2)$. Since $b_2$ is in the ancestry of $p_2$, and $r_1$ is in the ancestry of $b_1$, it follows that $w(r_1) \leq w(b_1) \leq w(b_2) \leq w(p_2)$. Since $p_2$ and $p_1$ are on two different chains, unless the block producer committed a slashable act, the ranges $(w(r_1), w(p_1))$ and $(w(r_2), w(p_2))$ do not intersect, and since $w(r_1) \leq w(p_2)$, it must be that $w(p_1) < w(r_2)$. Recall that for two approvals such that $w(p_1) < w(r_2)$ it is also required that $s(p_1) < s(r_2)$, as shown on figure 5.

Since the score of $p_1$ is at least $w(b_1)$, the score of $r_2$ is strictly greater than $w(b_1)$. But that implies that there is a block $b_2'$ with weight higher than $w(b_1)$ that has a quorum pre-vote in the chain originating at $b_2$.

Thus, for a block $b_2$ with a weight higher than $w(b_1)$ that is not on the same chain as $b_1$ to exist, another such block $b_2'$ must exist in its ancestry, which contradicts the assumption that $b_2$ is the earliest such block. □

**Theorem 1** (Safety Guarantee)**.** *If a block $b_1$ is finalized in a chain with a tip $t_1$, any participant that has seen $t_1$ will have $b_1$ in their canonical chain, unless more than $\lceil n/3 - 1 \rceil$ block producers committed a slashable act, or a block heavier than $w(b_1)$ that is not in the ancestry of $b_1$ was finalized by another block producers set.*

*Proof.* Say it is not the case, and another chain is chosen as canonical. Since the chain originating at $t_1$ has a quorum pre-vote for $b_1$, its score is at least $w(b_1)$. Thus, the canonical chain that does not include $b_1$ must have a score of at least $w(b_1)$, otherwise it will not be chosen by the fork choice rule. By definition of score, that implies that there is a block other than $b_1$ with weight equal or greater than $w(b_1)$ that has a quorum pre-vote. This contradicts lemma 1. □

## 3.2 Plausible Liveness of the Finality Gadget

We say a protocol has *plausible* liveness if for any state there is a sequence of actions by the participants of the protocol that results in finalizing a block that previously was not finalized. It is different from liveness in general which usually refers to a property that, if honest participants follow the protocol, they are guaranteed to make progress.

Casper NFG has plausible liveness for as long as the blocks continue getting produced. Consider all the blocks that exist in a given moment in time, and a

canonical chain chosen among such blocks by the fork choice rule. The block producers collectively can always finalize a block on top of such chain. It is sufficient to keep producing blocks on top of the chain until a block with a weight heavier than the heaviest block ever approved by any block producer is created, at which point all the block producers will be able to produce an approval for such a block, and thus within the next two blocks finalize it.

Note that while Casper NFG has plausible liveness, naively it does not have liveness in general. Specifically, it is easy to construct a scenario in which a block on one chain gets a quorum pre-vote but then, before it gets a quorum pre-commit, a heavier block on another chain gets a quorum pre-vote and then, before that block gets a quorum pre-commit, a yet heavier block on the first chain gets a quorum pre-vote and the process repeats.

In practice we assume a synchronous network, meaning that the blocks are generally distributed across the network faster than the time between two blocks. With such synchronous network assumption we hypothesize that it is likely that if a block $b_h$ at height $h$ has a quorum pre-vote for some other block $b_0$, $b_h$ will be seen by the block producer of height $h+1$ before the block $b_{h+1}$ for height $h+1$ is produced, and it is likely that the block producer for $h+1$ will have time to accumulate approvals for $b_h$, in which case $b_0$ will become finalized.

### 3.3 Finality Guarantees and Long-Range Attacks

In this section we analyze what an adversary needs to control or commit to reverse a finalized block with an assumption that at least $\lfloor 2n/3+1 \rfloor$ of the block producers at any point are online, are producing and approving blocks they observe, and all the clients are consistency-favoring and do not consider a block finalized unless each epoch in the past had at least one block finalized. Ensuring finality without such an assumption is left for future work, see section 4.1.

Consider figure 6. Say a client observes two blocks $A$ and $B$ such that both $A$ and $B$ are the latest finalized blocks in their corresponding chains, neither $A$ nor $B$ are in the ancestries of each other, and each epoch preceding both $A$ and $B$ had at least one block finalized. Say that (unknown to the client) $A$ is on the canonical chain built by the honest block producers, and $B$ is created by an adversary. Consider the common ancestor $C$ of $A$ and $B$ in epoch $i$. Since the block producers set in epoch $i+1$ is known as of the last block in epoch $i-1$, the block producers sets in epoch $i+1$ in the ancestry of $A$ and $B$ is the same. Since by assumption each epoch has at least one block finalized, there is a block $A'$ in epoch $i+1$ in the ancestry of $A$ that is finalized. Similarly there is a block $B'$ in the epoch $i+1$ in the ancestry of $B$ that is finalized. Naturally, those two blocks are not in the ancestries of each other. Thus $A'$ and $B'$ were finalized by the same block producers set.

Say the stake is returned to the user $z$ epochs after the last epoch in which they signed at least one block on the chain.

Say $A$ is in the epoch $i+k+1$. Consider the value of $k$. If $k \leq z$, then the stake of the block producers that produced blocks in epoch $i+1$ in the ancestry

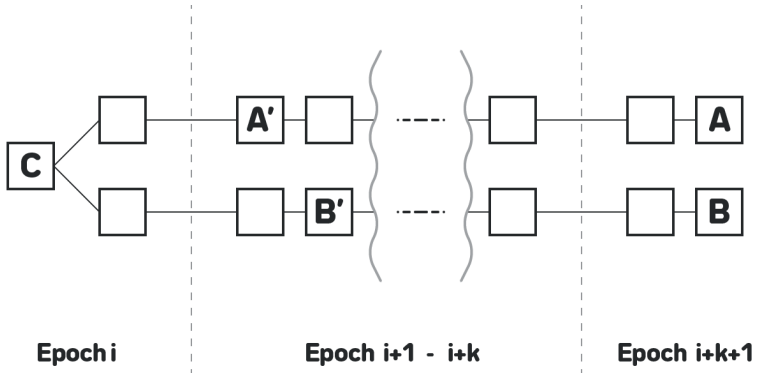


Figure 6: Fork in a blockchain with each epoch having at least one block finalized

of $A$ is still staked, and will be slashed.

Say $k > z$. Then the stake is already unstaked, and the block producers of epoch $i + 1$ cannot be penalized for double signing $A'$ and $B'$. Since $A$ is finalized, the score of the chain with $A$ is at least $w(A)$. The weight of $A$ includes all the space-time proofs computed by the honest miners in the last $k$ epochs, excluding the proofs that were already computed, but are yet to be included during the current epoch. Thus, the weight of $A$ is at least $\frac{k-1}{k}$ of all the space-time proofs computed by the honest actors since the fork occurred. For the chain containing $B$ to have higher score than $A$, at least one block in that chain needs to have weight higher than the weight of $A$, and thus the adversary had to compute $\frac{k-1}{k}$ of the space-time proofs computed by the honest actors in the same time frame.

From the above it follows, that assuming the adversary cannot compute proofs of elapsed time considerably faster than the honest actors, to make a client choose a chain that contains $B$ the adversary needs to either have at least $1/3$ of the total stake slashed, or control more than $\frac{k-1}{2k-1}$ of the total space used for space-time proofs in the system.

# 4 Future Work

## 4.1 Favoring Availability

Consider figure 6. Say we want to allow finalizing blocks for which there is at least one epoch in their ancestry that does not have any finalized blocks. Then for the adversary to have $B$ finalized they do not need to have a block finalized in epoch $i+1$. Instead they can have a subset of their block producers that is smaller than $\lceil n/3 \rceil$ only produce and approve blocks on the chain that

will contain $B$. No block on such chain will be finalized until all the honest participants drop out due to inactivity on the chain, but also no block producer controlled by the adversary will be slashed. If the adversary creates the fork sufficiently early in the epoch $i$, from perspective of the chain that contains $B$, the honest block producers will not have produced and approved sufficient number of blocks in epoch $i$, and thus will be kicked out by the beginning of epoch $i + 2$ at which point the adversary will be able to finalize a block.

Our early analysis suggests that it is possible with minor modifications to the score function to ensure that the adversary cannot revert a block unless they control a large percentage of space used for mining, even if the epochs that have no blocks finalized in them are allowed in the ancestry of a finalized block. Formalizing the exact changes to the score function and the corresponding proofs are left for future revisions of this paper.

### 4.2 Guaranteed Liveness

Our early analysis suggests that with some modifications to the finality gadget (specifically, carefully placed timeouts that increase with each block height for which finality was not achieved) it is likely to have provable liveness under partially synchronous network assumption. The exact modifications and the corresponding proofs are left for future revisions of this paper.

## 5 Conclusion

The construction presented in this document provides the fast finality of Proof-of-Stake blockchains based on BFT consensus protocols and resilience to long range attacks of Proof-of-Work-based protocols. As an added benefit, the construction discourages pooling.

Using Proof-of-Space-Time instead of Proof-of-Work significantly reduces the negative effects on the environment.

## References


[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`.

[2] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388. Springer, 2017.

[3] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, August 2016. USENIX Association.

[4] Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 381–409. Springer, 2019.

[5] Alex Skidanov and Illia Polosukhin. Nightshade: Near protocol sharding design. `http://near.ai/nightshade`.

[6] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.