Casper the Friendly Finality Gadget

Vitalik Buterin and **Virgil Griffith**
Ethereum Foundation

Abstract

We introduce Casper, a proof of stake-based finality system which overlays an existing proof of work blockchain. Casper is a partial consensus mechanism combining proof of stake algorithm research and Byzantine fault tolerant consensus theory. We introduce our system, prove some desirable features, and show defenses against long range revisions and catastrophic crashes. The Casper overlay provides almost any proof of work chain with additional protections against block reversions.

1. Introduction

Over the past few years there has been considerable research into "proof of stake" (PoS) based blockchain consensus algorithms. In a PoS system, a blockchain appends and agrees on new blocks through a process where anyone who holds coins inside of the system can participate, and the influence an agent has is proportional to the number of coins (or "stake") it holds. This is a vastly more efficient alternative to proof of work (PoW) "mining" and enables blockchains to operate without mining's high hardware and electricity costs.

There are two major schools of thought in PoS design. The first, *chain-based proof of stake*[1, 2], mimics proof of work mechanics and features a chain of blocks and and simulates mining by pseudorandomly assigning the right to create new blocks to stakeholders. This includes Peercoin[3], Blackcoin[4], and Iddo Bentov's work[5].

The other school, *Byzantine fault tolerant* (BFT) based proof of stake, is based on a thirty-year-old body of research into BFT consensus algorithms such as PBFT[6]. BFT algorithms typically have proven mathematical properties; for example, one can usually mathematically prove that as long as $> \frac{2}{3}$ of protocol participants are following the protocol honestly, then, regardless of network latency, the algorithm cannot finalize conflicting blocks. Repurposing BFT algorithms for proof of stake was first introduced by Tendermint[7], and has modern inspirations such as [8]. Casper follows this BFT tradition, though with some modifications.

1.1. Our Work

Casper the Friendly Finality Gadget is an overlay atop a *proposal mechanism*—a mechanism which proposes blocks[1]. Casper is responsible for finalizing these blocks, essentially selecting a unique chain which represents the canonical transactions of the ledger. Casper provides safety, but liveness depends on the chosen proposal mechanism. That is, if attackers wholly control the proposal mechanism, Casper protects against finalizing two conflicting checkpoints, but the attackers could prevent Casper from finalizing any future checkpoints.

Casper introduces several new features that BFT algorithms do not necessarily support:

- **Accountability**. If a validator violates a rule, we can detect the violation and know which validator violated the rule. Accountability allows us to penalize malfeasant validators, solving the "nothing at stake" problem that plagues chain-based PoS. The penalty for violating a rule is a validator's entire deposit. This maximal penalty is the defense against violating the protocol. Because proof of stake security is based on the size of the penalty, which can be set to greatly exceed the gains from the mining reward, proof of stake provides strictly stronger security incentives than proof of work.

[1]This functionality serves a similar role to the common abstraction of "leader election" used in traditional BFT algorithms, but is adapted to accomodate Casper's construction of being a finality overlay atop an existing blockchain.

- **Dynamic validators**. We introduce a safe way for the validator set to change over time (Section 3).
- **Defenses**. We introduce defenses against long range revision attacks as well as attacks where more than $\frac{1}{3}$ of validators drop offline, at the cost of a very weak tradeoff synchronicity assumption (Section 4).
- **Modular overlay**. Casper's design as an overlay makes it easier to implement as an upgrade to an existing proof of work chain.

We describe Casper in stages, starting with a simple version (Section 2) and then progressively adding validator set changes (Section 3) and finally defenses against attacks (Section 4).

2. The Casper Protocol

Within Ethereum, the proposal mechanism will initially be the existing proof of work chain, making the first version of Casper a hybrid PoW/PoS system. In future versions the PoW proposal mechanism will be replaced with something more efficient. For example, we can imagine converting the block proposal into a some kind of PoS round-robin block signing scheme.

In this simple version of Casper, we assume there is a fixed set of validators and a proposal mechanism (e.g., the familiar proof of work proposal mechanism) which produces child blocks of existing blocks, forming an ever-growing *block tree*. From [9] the root of the tree is typically called the "genesis block".

Under normal circumstances, we expect that the proposal mechanism will typically propose blocks one after the other in a linked list (i.e., each "parent" block having exactly one "child" block). But in the case of network latency or deliberate attacks, the proposal mechanism will inevitably occassionally produce multiple children of the same parent. Caspers job is to choose a single child from each parent, thus choosing one canonical chain from the block tree.

Rather than deal with the full block tree, for efficiency purposes[2] Casper only considers the subtree of *checkpoints* forming the *checkpoint tree* (Figure 1a). The genesis block is a checkpoint, and every block whose height in the block tree (or block number) is an exact multiple of 100 is also a checkpoint. The "checkpoint height" of a block with block height $100 * k$ is simply k; equivalently, the height $\mathrm{h}(c)$ of a checkpoint c is the number of elements in the checkpoint chain stretching from c all the way back to root along the parent links (Figure 1b).[3]

Each validator has a *deposit*; when a validator joins, its deposit is the number of deposited coins. After joining, each validator's deposit rises and falls with rewards and penalties. Proof of stake's security derives from the size of the deposits, not the number of validators, so for the rest of this paper, when we say "$\frac{2}{3}$ of validators", we are referring to the *deposit-weighted* fraction; that is, a set of validators whose sum deposit size equals to $\frac{2}{3}$ of the total deposit size of the entire set of validators.

Validators can broadcast a *vote* message containing four pieces of information (Table 1): two checkpoints s and t together with their heights $\mathrm{h}(s)$ and $\mathrm{h}(t)$. We require that s be an ancestor of t in the checkpoint tree, otherwise the vote is considered invalid. If the public key of the validator ν is not in the validator set, the vote is considered invalid. Together with the signature of the validator, we will write these votes in the form $\langle \nu, s, t, \mathrm{h}(s), \mathrm{h}(t) \rangle$.

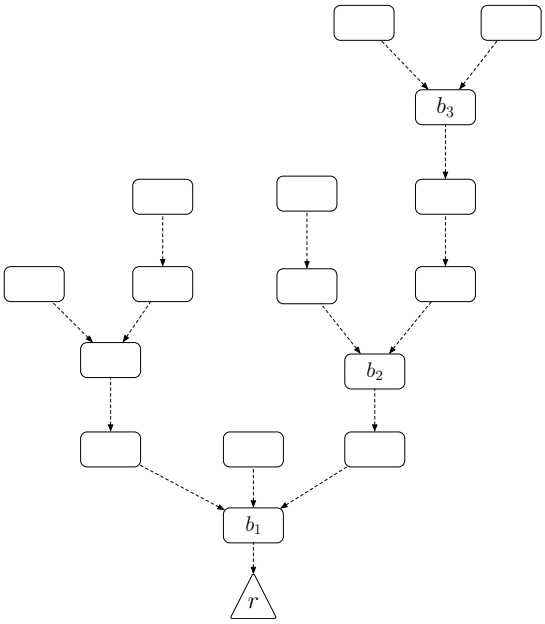
Notation	Description
s	the hash of any justified checkpoint (the "source")
t	any checkpoint hash that is a descendent of s (the "target")
$\mathrm{h}(s)$	the height of checkpoint s in the checkpoint tree
$\mathrm{h}(t)$	the height of checkpoint t in the checkpoint tree
$\mathcal{S}$	signature of $\langle s, t, \mathrm{h}(s), \mathrm{h}(t) \rangle$ from the validator ν's private key

Table 1: The schematic of a single **VOTE** message denoted $\langle \nu, s, t, \mathrm{h}(s), \mathrm{h}(t) \rangle$.

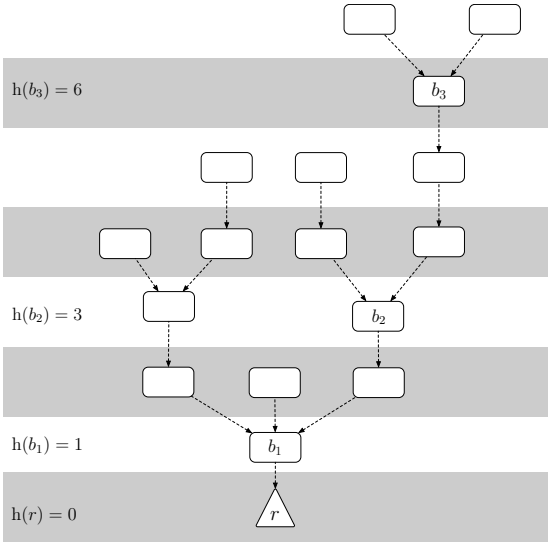
We define the following terms:

[2] A long distance between checkpoints reduces the overhead of the algorithm, but also increases the time it takes to come to consensus. We choose a spacing of 100 blocks between checkpoints as a middle ground.

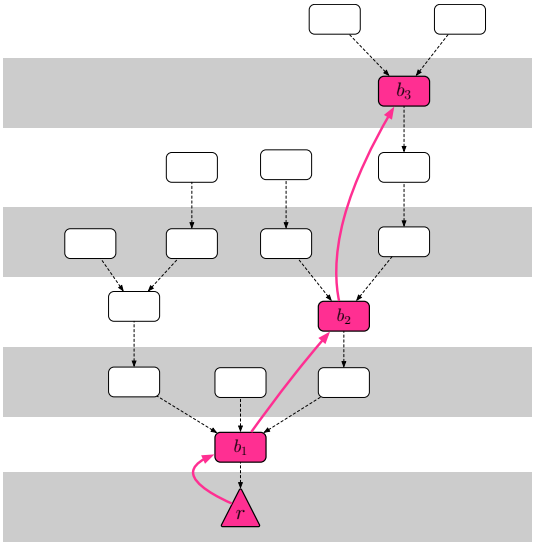
[3] Specifically, the height of a checkpoint is *not* the same as the number of ancestors in the checkpoint tree all the way back to root along the supermajority links (defined in the next section).



(a) The checkpoint tree. The dashed line represents 100 blocks between the checkpoints, which are represented by rounded rectangles. The root of the tree is denoted "r".



(b) The height function



(c) The justified chain $r \rightarrow b_1 \rightarrow b_2 \rightarrow b_3$

Figure 1: Illustrating a checkpoint tree, the height function, and a justified chain within the checkpoint tree.

- A *supermajority link* is an ordered pair of checkpoints (a, b), also written $a \rightarrow b$, such that at least $\frac{2}{3}$ of validators (by deposit) have published votes with source a and target b. Supermajority links *can skip checkpoints*, i.e., it's perfectly okay for $\mathrm{h}(b) > \mathrm{h}(a) + 1$. Figure 1c shows three distinct supermajority links in red: $r \rightarrow b_1$, $b_1 \rightarrow b_2$, and $b_2 \rightarrow b_3$.
- Two checkpoints a and b are called *conflicting* if and only if they are nodes in distinct branches, i.e., neither is an ancestor or descendant of the other.
- A checkpoint c is called *justified* if (1) it is the root, or (2) there exists a supermajority link $c' \rightarrow c$ where c' is justified. Figure 1c shows a chain of four justified blocks.
- A checkpoint c is called *finalized* if it is justified and there is a supermajority link $c \rightarrow c'$ where c' is a direct child of c.

AN INDIVIDUAL VALIDATOR ν MUST NOT PUBLISH TWO DISTINCT VOTES,

$$\langle \nu, s_1, t_1, \mathrm{h}(s_1), \mathrm{h}(t_1) \rangle \qquad \text{AND} \qquad \langle \nu, s_2, t_2, \mathrm{h}(s_2), \mathrm{h}(t_2) \rangle \ ,$$

SUCH THAT EITHER:

I. $\mathrm{h}(t_1) = \mathrm{h}(t_2)$.
Equivalently, a validator must not publish two distinct votes for the same target height.

OR

II. $\mathrm{h}(s_1) < \mathrm{h}(s_2) < \mathrm{h}(t_2) < \mathrm{h}(t_1)$.
Equivalently, a validator must not vote within the span of its other votes.

Figure 2: The two Casper Commandments. Any validator who violates either of these commandments gets their deposit slashed.

The most notable property of Casper is that it is impossible for two conflicting checkpoints to be finalized without $\geq \frac{1}{3}$ of the validators violating one of the two[4] Casper Commandments/slashing conditions (Figure 2).

If a validator violates either slashing condition, the evidence of the violation can be included into the blockchain as a transaction, at which point the validator's entire deposit is taken away with a small "finder's fee" given to the submitter of the evidence transaction. In current Ethereum, stopping the enforcement of a slashing condition requires a successful 51% attack on Ethereum's proof-of-work block proposer.

2.1. Proving Safety and Plausible Liveness

We prove Casper's two fundamental properties: *accountable safety* and *plausible liveness*. Accountable safety means that two conflicting checkpoints cannot both be finalized unless $\geq \frac{1}{3}$ of validators violate a slashing condition (meaning at least one third of the total deposit is lost). Plausible liveness means that, regardless of any previous events (e.g., slashing events, delayed blocks, censorship attacks, etc.), if $\geq \frac{2}{3}$ of validators follow the protocol, then it's always possible to finalize a new checkpoint without any validator violating a slashing condition.

Under the assumption that $\frac{2}{3}$ of the validators by weight do not violate a slashing condition, we have the following properties:

(i) If $s_1 \to t_1$ and $s_2 \to t_2$ are distinct supermajority links, then $\mathrm{h}(t_1) \neq \mathrm{h}(t_2)$.

(ii) If $s_1 \to t_1$ and $s_2 \to t_2$ are distinct supermajority links, then the inequality $\mathrm{h}(s_1) < \mathrm{h}(s_2) < \mathrm{h}(t_2) < \mathrm{h}(t_1)$ cannot hold.

From these two properties, we can immediately see that, for any height n:

(iii) there exists at most one supermajority link $s \to t$ with $\mathrm{h}(t) = n$.

(iv) there exists at most one justified checkpoint with height n.

With these four properties in hand, we move to the main theorems.

Theorem 1 (Accountable Safety)**.** *Two conflicting checkpoints a_m and b_n cannot both be finalized.*

Proof. Let a_m (with justified direct child a_{m+1}) and b_n (with justified direct child b_{n+1}) be distinct finalized checkpoints as in Figure 3. Now suppose a_m and b_n conflict, and without loss of generality $\mathrm{h}(a_m) < \mathrm{h}(b_n)$ (if $\mathrm{h}(a_m) = \mathrm{h}(b_n)$, then it is clear that $\frac{1}{3}$ of validators violated condition **I**). Let $r \to b_1 \to b_2 \to \cdots \to b_n$ be a chain of checkpoints, such that there exists a supermajority link $r \to b_1, \ldots, b_i \to b_{i+1}, \ldots, b_n \to b_{n+1}$. We know that no $\mathrm{h}(b_i)$ equals either $\mathrm{h}(a_m)$ or $\mathrm{h}(a_{m+1})$, because that violates property (iv). Let j be the lowest integer such that $\mathrm{h}(b_j) > \mathrm{h}(a_{m+1})$; then $\mathrm{h}(b_{j-1}) < \mathrm{h}(a_m)$. However, this implies the existence of a supermajority link from a checkpoint with an epoch number less than $\mathrm{h}(a_m)$ to a checkpoint with an epoch number greater than $\mathrm{h}(a_{m+1})$, which is incompatible with the supermajority link from a_m to a_{m+1}. □

[4]Earlier versions of Casper had two types of messages and four slashing conditions[10], but we have reduced this to one message type and two slashing conditions. We removed the conditions: (i) Committed hashes must already be justified, and (ii) prepare messages must point to an already justified ancestor. This is a design choice. We made this choice so that slashing violations are independent of the state of the chain.

Theorem 2 (Plausible Liveness)**.** *Supermajority links can always be added to produce new finalized checkpoints, provided there exist children extending the finalized chain.*

Proof. Let a be the justified checkpoint with the greatest height, and b be the target checkpoint with the greatest height for which any validator has made a vote. Any checkpoint a' which is a descendant of a with height $\mathrm{h}(a') = \mathrm{h}(b) + 1$ can be justified without violating either commandments **I** or **II**, and then a' can be finalized by adding a supermajority link from a' to a direct child of a', also without violating either **I** or **II**. □

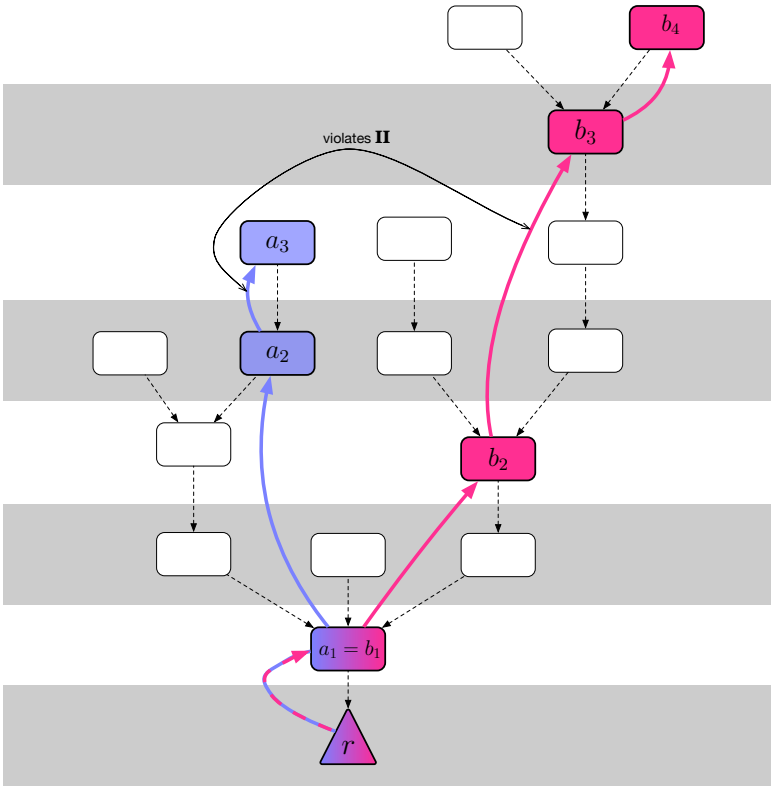


Figure 3: Figure for Theorem 1 (Accountable Safety).

2.2. Casper's Fork Choice Rule

Casper is more complicated than standard PoW designs. As such, the fork-choice must be adjusted. Our modified fork-choice rule should be followed by all users, validators, and even the underlying block proposal mechanism. If the users, validators, or block-proposers instead follow the standard PoW fork-choice rule of "always build atop the longest chain", there are pathological scenarios where Casper gets "stuck" and any blocks built atop the longest chain cannot be finalized (or even justified) without some validators altruistically sacrificing their deposit. To avoid this, we introduce a novel, correct by construction, fork choice rule: FOLLOW THE CHAIN CONTAINING THE JUSTIFIED CHECKPOINT OF THE GREATEST HEIGHT. This fork choice rule is correct by construction because it follows from the plausible liveness proof (Theorem 2), which precisely states that it's always possible to finalize a new checkpoint on top of the justified checkpoint with the greatest height. This fork choice rule will be tweaked in Sections 3 and 4.

3. Enabling Dynamic Validator Sets

The set of validators needs to be able to change. New validators must be able to join, and existing validators must be able to leave. To accomplish this, we define the *dynasty* of a block. The *dynasty of block* b is the number of finalized checkpoints in the chain from root to the parent of block b. When a would-be validator's *deposit message*

is included in a block with dynasty d, then the validator ν will join the validator set at first block with dynasty $d+2$. We call $d+2$ this validator's *start dynasty*, $\mathrm{DS}(\nu)$.

To leave the validator set, a validator must send a "withdraw" message. If validator ν's withdraw message is included in a block with dynasty d, it similarly leaves the validator set at the first block with dynasty $d+2$; we call $d+2$ the validator's *end dynasty*, $\mathrm{DE}(\nu)$. If a withdraw message has not yet been included, then $\mathrm{DE}(\nu) = \infty$. Once validator ν leaves the validator set, the validator's public key is forever forbidden from rejoining the validator set. This removes the need to handle multiple start/end dynasties for a single identifier.

At the start of the end dynasty, the validator's deposit is locked for a long period of time, called the *withdrawal delay* (think "four months' worth of blocks"), before the deposit is withdrawn. If, during the withdrawal delay, the validator violates any commandment, the deposit is slashed.

We define two functions that generate two subsets of validators for any given dynasty d, the *forward validator set* and the *rear validator set*. They are defined as,

$$\begin{aligned}\mathcal{V}_{\mathrm{f}}(d) &\equiv \{\nu : \mathrm{DS}(\nu) \leq d < \mathrm{DE}(\nu)\} \\ \mathcal{V}_{\mathrm{r}}(d) &\equiv \{\nu : \mathrm{DS}(\nu) < d \leq \mathrm{DE}(\nu)\}\ .\end{aligned}$$

Note this means that the forward validator set of dynasty d is the rear validator set of dynasty $d+1$.

Note that in order for the chain to be able to "know" its own current dynasty, we need to restrict our definition of "finalization" slightly: Before, a checkpoint c is called *finalized* if it is justified and there is a supermajority link from c to any of its direct children in the checkpoint tree. Now, finalization has one additional condition—c is finalized only if the votes for the supermajority link $c \to c'$, as well as all of the supermajority links recursively justifying c, are included in the block chain before the child of c', i.e., before block number $\mathrm{h}(c') * 100$. To support dynamic validator sets, we redefine a supermajority link and finalization as follows:

- An ordered pair of checkpoints (s, t), where t is in dynasty d, has a *supermajority link* if both at least $\frac{2}{3}$ of the forward validator set of dynasty d have published votes $s \to t$ and at least $\frac{2}{3}$ of the rear validator set of dynasty d have published votes $s \to t$.
- A checkpoint c is called currently *finalized* if c is justified and there is a supermajority link from $c \to c'$ where c' is a child of c. We add the condition that c is finalized only if the votes for the supermajority link $c \to c'$, as well as the supermajority link justifying c, are included in c''s block chain and before the child of c'—i.e., before block number $\mathrm{h}(c') * 100$.

The forward and rear validator sets will usually greatly overlap; but if the two validator sets substantially differ, this "stitching" mechanism prevents safety failure in the case when two grandchildren of a finalized checkpoint have different dynasties because the evidence was included in one chain but not the other. For an example of this, see Figure 4.

4. Stopping Attacks

There are two well-known attacks against proof-of-stake systems: *long range revisions* and *catastrophic crashes*. We discuss each in turn.

4.1. Long Range Revisions

The withdrawal delay after a validator's end dynasty introduces a synchronicity assumption between validators and clients. Once a coalition of validators has withdrawn their deposits, if that coalition had more than $\frac{2}{3}$ of deposits *long ago in the past*, they can use their historical supermajority to finalize conflicting checkpoints without fear of getting slashed (because they have already withdrawn their money). This is called the *long-range revision attack*, see in Figure 5.

In simple terms, long-range attacks are prevented by a fork choice rule to never revert a finalized block, as well as an expectation that each client will "log on" and gain a complete up-to-date view of the chain at some regular frequency (e.g., once per 1–2 months). A "long range revision" fork that finalizes blocks older than that will simply be ignored, because all clients will have already seen a finalized block at that height and will refuse to revert it.

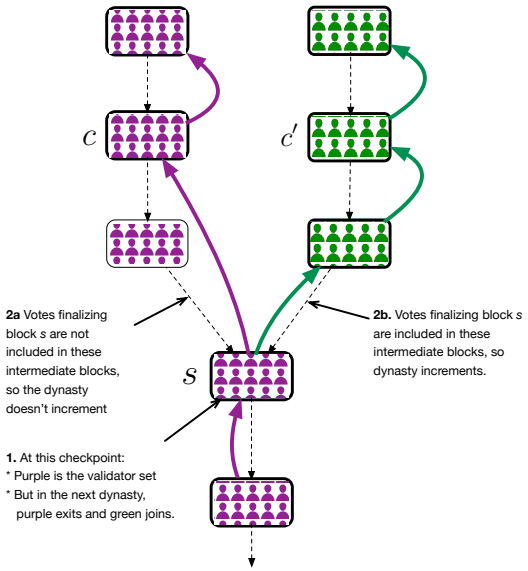


Figure 4: **Attack from dynamic validator sets**. Without the validator set stitching mechanism, it's possible for two conflicting checkpoints c and c' to both be finalized without any validator getting slashed. In this case c and c' are the same height thus violating commandment **I**, but because the validator sets finalizing c and c' are disjoint, no one gets slashed.

We make an informal proof of the mechanism as follows. Suppose that:

- There is a maximum communication delay δ between two clients, so if one client hears some message at time t, all other clients are guaranteed to have heard it by time $t + \delta$. This means that we can talk about the "time window" $[t_{\min}, t_{\max}]$ during which a block was received by the network, with width $t_{\max} - t_{\min}$ at most δ.
- We assume all clients have local clocks are perfectly synchronized (any discrepancy can be treated as being part of the communication delay δ).
- Blocks are required to have timestamps. If a client has local time T_L, then it will reject blocks whose timestamp T_B satisfies $T_B > T_L$ (i.e., in the future), and they will refuse to accept as finalized (but may still accept as part of the chain) blocks where $T_B < T_L - \delta$ (i.e., too far in the past)
- If a validator sees a slashing violation at time t (that's the time they hear the *later* of the two votes), then they reject blocks with timestamps $> t + 2\delta$ that are part of chains that have not yet included this slashing evidence.

Suppose that a large set of slashing violations results in two conflicting finalized checkpoints, c_1 and c_2. If the two time windows do not intersect, then all validators agree which checkpoint came first and everyone follows the rule to not revert finalized checkpoints then there is no issue.

If the two time windows *do* intersect, then we can handle the case as follows. Let c_1's time window be $[0, \delta]$ and c_2's time window be $[\delta - \epsilon, 2\delta - \epsilon]$. Then the timestamps of both are at least 0. By time 2δ it is guaranteed that all clients have seen the slashing violation, so they reject blocks with timestamp $> 4\delta$ whose chains that have not yet included the evidence transaction. Hence, as long as $\omega > 4\delta$, it's guaranteed that malfeasant validators will lose their deposits in all chains that any client accepts, where ω is the "withdraw delay" (Figure 5), the delay between the end-epoch and when validators actually receive their deposits back.

Due to network delays, it's possible that clients will disagree whether a given piece of slashing evidence was submitted into a given chain "on time" or as having accepted it too late. However, this is only a liveness failure, not a safety failure, and this possibility does not weaken our security claims because it is already known that a corrupted proposal mechanism (which would be required to prevent evidence inclusion) can prevent finality.

We can also sidestep the issue of evidence inclusion timeouts by informally arguing that attacks will be short-lived, because the validators will perceive a long-running chain without including slashing evidence as an attack and

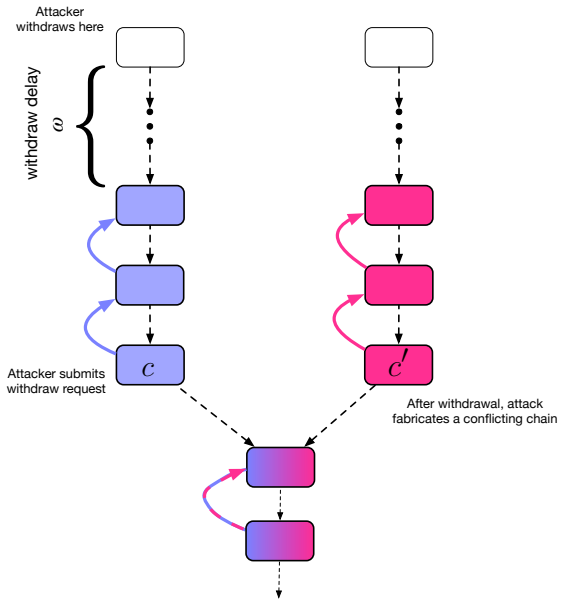


Figure 5: **The long range attack**. As long as a client gains complete knowledge of the justified chain at a regular interval, it will not be susceptible to a long range attack.

switch to another branch supported by an honest minority of validators that are not part of the attack (see Section 4.2) thus stopping the attack and slashing the attacker.

4.2. Castastrophic Crashes

Suppose that $> \frac{1}{3}$ of validators crash-fail at the same time—i.e., they are no longer connected to the network due to a network partition, computer failure, or the validators themselves are malicious. Intuitively, from this point on, no supermajority links can be created, and thus no future checkpoints can be finalized.

We can recover from this by instituting an "inactivity leak" which slowly drains the deposit of any validator that does not vote for checkpoints, until eventually its deposit sizes decrease low enough that the validators who *are* voting are a supermajority. The simplest formula is something like "in every epoch a validator with deposit size D fails to vote, it loses $D * p$ (for $0 < p < 1$)", though to resolve catastrophic crashes more quickly a formula which increases the leak rate in the event of a long streak of non-finalized blocks may be optimal.

This drained ether can be burned or returned to the validator after ω days. Whether leaked assets should be burned or returned as well as the exact formula for the inactivity leak is outside the scope of this paper as these are questions of economic incentives, not Byzantine-fault-tolerance.

The inactivity leak introduces the possibility of two conflicting checkpoints being finalized without any validator getting slashed (as in Figure 6), with validators only losing money on only one of the two checkpoints. If the validators are split into two subsets, with subset $\mathbf{V}_A$ voting on chain A and subset $\mathbf{V}_B$ voting on chain B. On chain A, $\mathbf{V}_B$'s deposits will leak, and vice versa, leading to each subset having a supermajority on its respective chain, allowing two conflicting checkpoints to be finalized without any validators being explicitly slashed (but each subset will lose a large portion of their deposit on one of the two chains due to leaks). If this situation happens, then each validator should simply favor whatever finalized checkpoint it saw first.

The exact algorithm for recovering from these various attacks remains an open problem. For now, we assume validators can detect obviously malfeasant behavior (e.g., not including evidence) and manually create a "minority soft fork". This minority fork can be viewed as a blockchain in its own right that competes with the majority chain in the market, and if the majority chain truly is operated by colluding malicious attackers then we can assume that the market will favor the minority fork.

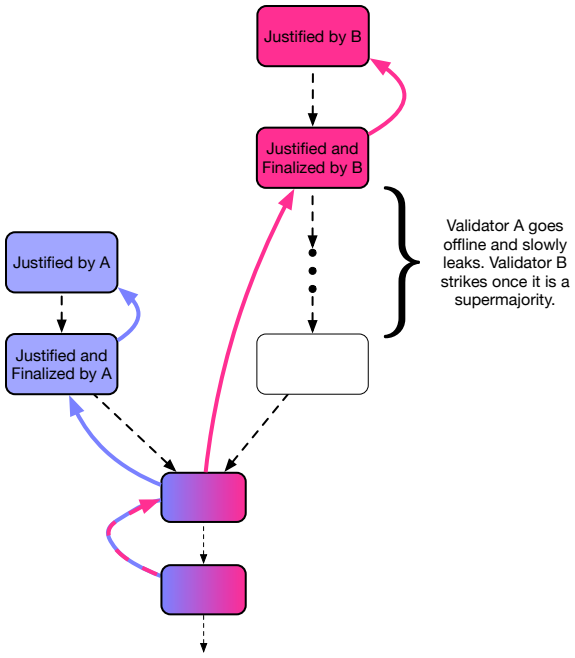


Figure 6: **Inactivity leak**. The checkpoint on the left can be finalized immediately. The checkpoint on the right can be finalized after some time, once offline validator deposits sufficiently deplete.

5. Conclusions

We presented Casper, a novel proof of stake system derived from the Byzantine fault tolerance literature. Casper includes: two slashing conditions, a correct-by-construction fork choice rule inspired by [11], and dynamic validator sets. Finally we introduced extensions to Casper (not reverting finalized checkpoints and the inactivity leak) to defend against two common attacks.

Casper remains imperfect. For example, a wholly compromised block proposal mechanism will prevent Casper from finalizing new blocks. Casper is an PoS-based strict security improvement to almost any PoW chain. The problems that Casper does not wholly solve, particularly related to 51% attacks, can still be corrected using user-activated soft forks. Future developments will undoubtedly improve Casper's security and reduce the need for user-activated soft forks.

Future Work. The current Casper system builds upon a proof of work block proposal mechanism. We wish to convert the block proposal mechanism to proof of stake. We wish to prove accountable safety and plausible liveness even when the weights of the validator set change with rewards and penalties. Another problem for future work is a formal specification of a fork-choice rule taking into account the common attacks on proof of stake. Future workpapers will explain and analyze the financial incentives within Casper and their consequences. A particular economic problem related to such automated strategies to block attackers is proving upper bounds on the ratio between the degree of disagreement between different clients and the cost incurred by the attacker.

Acknowledgements. We thank Jon Choi, Karl Floersch, Ozymandias Haynes, and Vlad Zamfir for frequent discussions.

References

[1] Pass, R. & Shi, E. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC 2017, 315–324 (ACM, New York, NY, USA, 2017). URL `http://doi.acm.org/10.1145/3087801.3087809`.

[2] Introducing dfinity crypto techniques (2017). URL `https://dfinity.org/pdf-viewer/pdfs/viewer.html?file=../library/intro-dfinity-crypto.pdf`.

[3] King, S. & Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake **19** (2012). URL https://decred.org/research/king2012.pdf. https://web.archive.org/save/https://decred.org/research/king2012.pdf.

[4] Vasin, P. Blackcoin's proof-of-stake protocol v2 (2014). URL http://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf.

[5] Bentov, I., Gabizon, A. & Mizrahi, A. Cryptocurrencies without proof of work. In Sion, R. (ed.) *International Conference on Financial Cryptography and Data Security*, 142–157 (Springer, 2016). URL http://www.cs.technion.ac.il/~idddo/CoA.pdf.

[6] Castro, M., Liskov, B. & et. al. Practical byzantine fault tolerance. In Leach, P. J. & Seltzer, M. (eds.) *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, vol. 99, 173–186 (1999). URL http://pmg.csail.mit.edu/papers/osdi99.pdf.

[7] Kwon, J. Tendermint: Consensus without mining (2014). URL https://tendermint.com/static/docs/tendermint.pdf.

[8] Chen, J. & Micali, S. ALGORAND: the efficient and democratic ledger. *CoRR* **abs/1607.01341** (2016). URL http://arxiv.org/abs/1607.01341.

[9] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system (2008). URL https://bitcoin.org/bitcoin.pdf.

[10] Buterin, V. Minimal slashing conditions (2017). URL https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c.

[11] Sompolinsky, Y. & Zohar, A. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. **2013** (2013). URL http://eprint.iacr.org/2013/881.