Proof of Stake FAQ

Daniel Kronovet edited this page on Oct 31, 2017 · 50 revisions

What is Proof of Stake

Proof of Stake (PoS) is a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network. In proof of work (PoW) based public blockchains (e.g. Bitcoin and the current implementation of Ethereum), the algorithm rewards participants who solve cryptographic puzzles in order to validate transactions and create new blocks (i.e. mining). In PoS-based public blockchains (e.g. Ethereum's upcoming Casper implementation), a set of validators take turns proposing and voting on the next block, and the weight of each validator's vote depends on the size of its deposit (i.e. stake). Significant advantages of PoS include **security, reduced risk of centralization, and energy efficiency**.

In general, a proof of stake algorithm looks as follows. The blockchain keeps track of a set of validators, and anyone who holds the blockchain's base cryptocurrency (in Ethereum's case, ether) can become a validator by sending a special type of transaction that **locks up their ether into a deposit**. The process of creating and agreeing to new blocks is then done through a consensus algorithm that all current validators can participate in.

There are many kinds of consensus algorithms, and many ways to assign rewards to validators who participate in the consensus algorithm, so there are many "flavors" of proof of stake. From an algorithmic perspective, there are two major types: chain-based proof of stake and BFT-style proof of stake.

In **chain-based proof of stake**, the algorithm pseudo-randomly selects a validator during each time slot (eg. every period of 10 seconds might be a time slot), and assigns that validator the right to create a single block, and this block must point to some previous block (normally the block at the end of the previously longest chain), and so over time most blocks converge into a single constantly growing chain.

In **BFT-style proof of stake**, validators are randomly assigned the right to *propose* blocks, but *agreeing on which block is canonical* is done through a multi-round process where every validator sends a "vote" for some specific block during each round, and at the end of the process all (honest and online) validators permanently agree on whether or not any given block is part of the chain. Note that blocks may still be *chained together*; the key difference is that consensus on a block can come within one block, and does not depend on the length or size of the chain after it.

What are the benefits of proof of stake as opposed to proof of work?

See A Proof of Stake Design Philosophy for a more long-form argument.

In short:

- **No need to consume large quantities of electricity** in order to secure a blockchain (eg. it's estimated that both Bitcoin and Ethereum burn over $1 million worth of electricity and hardware costs per day as part of their consensus mechanism).
- Because of the lack of high electricity consumption, there is **not as much need to issue as many new coins** in order to motivate participants to keep participating in the network. It may theoretically even be possible to have *negative* net issuance, where a portion of transaction fees is "burned" and so the supply goes down over time.
- Proof of stake opens the door to a wider array of techniques that use game-theoretic mechanism design in order to better **discourage centralized cartels** from forming and, if they do form, from acting in ways that are harmful to the network (eg. like selfish mining in proof of work).
- **Reduced centralization risks**, as economies of scale are much less of an issue. $10 million of coins will get you exactly 10 times higher returns than $1 million of coins,

without any additional disproportionate gains because at the higher level you can afford better mass-production equipment.

- Ability to use economic penalties to **make various forms of 51% attacks vastly more expensive** to carry out than proof of work - to paraphrase Vlad Zamfir, "it's as though your ASIC farm burned down if you participated in a 51% attack".

How does proof of stake fit into traditional Byzantine fault tolerance research?

There are several fundamental results from Byzantine fault tolerance research that apply to all consensus algorithms, including traditional consensus algorithms like PBFT but also any proof of stake algorithm and, with the appropriate mathematical modeling, proof of work.

The key results include:

- **CAP theorem** - "in the cases that a network partition takes place, you have to choose either consistency or availability, you cannot have both". The intuitive argument is simple: if the network splits in half, and in one half I send a transaction "send my 10 coins to A" and in the other I send a transaction "send my 10 coins to B", then either the system is unavailable, as one or both transactions will not be processed, or it becomes inconsistent, as one half of the network will see the first transaction completed and the other half will see the second transaction completed. Note that the CAP theorem has nothing to do with scalability; it applies to sharded and non-sharded systems equally.
- **FLP impossibility** - in an asynchronous setting (ie. there are no guaranteed bounds on network latency even between correctly functioning nodes), it is not possible to create an algorithm which is guaranteed to reach consensus in any specific finite amount of time if even a single faulty/dishonest node is present. Note that this does NOT rule out "Las Vegas" algorithms that have some probability each round of achieving consensus and thus will achieve consensus within T seconds with probability exponentially approaching 1 as T grows; this is in fact the "escape hatch" that many successful consensus algorithms use.
- **Bounds on fault tolerance** - from the DLS paper we have: (i) protocols running in a partially synchronous network model (ie. there is a bound on network latency but we do not know ahead of time what it is) can tolerate up to 1/3 arbitrary (ie. "Byzantine") faults, (ii) deterministic protocols in an asynchronous model (ie. no bounds on network latency) cannot tolerate faults (although their paper fails to mention that randomized algorithms can with up to 1/3 fault tolerance), (iii) protocols in a synchronous model (ie. network latency is guaranteed to be less than a known `d`) can, surprisingly, tolerate up to 100% fault tolerance, although there are restrictions on what can happen when more than or equal to 1/2 of nodes are faulty. Note that the "authenticated Byzantine" model is the one worth considering, not the "Byzantine" one; the "authenticated" part essentially means that we can use public key cryptography in our algorithms, which is in modern times very well-researched and very cheap.

Proof of work has been rigorously analyzed by Andrew Miller and others and fits into the picture as an algorithm reliant on a synchronous network model. We can model the network as being made up of a near-infinite number of nodes, with each node representing a very small unit of computing power and having a very small probability of being able to create a block in a given period. In this model, the protocol has 50% fault tolerance assuming zero network latency, ~46% (Ethereum) and ~49.5% (Bitcoin) fault tolerance under actually observed conditions, but goes down to 33% if network latency is equal to the block time, and reduces to zero as network latency approaches infinity.

Proof of stake consensus fits more directly into the Byzantine fault tolerant consensus mould, as all validators have known identities (stable Ethereum addresses) and the network keeps track of the total size of the validator set. There are two general lines of proof of stake research, one looking at synchronous network models and one looking at partially asynchronous network models. "Chain-based" proof of stake algorithms almost always rely on synchronous network models, and their security can be formally proven within these models similarly to how security of proof of work algorithms can be proven. A line of research connecting traditional Byzantine fault tolerant consensus in partially synchronous networks to proof of stake also exists, but is more complex to explain; it will be covered in more detail in later sections.

Clone this wiki locally

`https://github.com/ethereum`

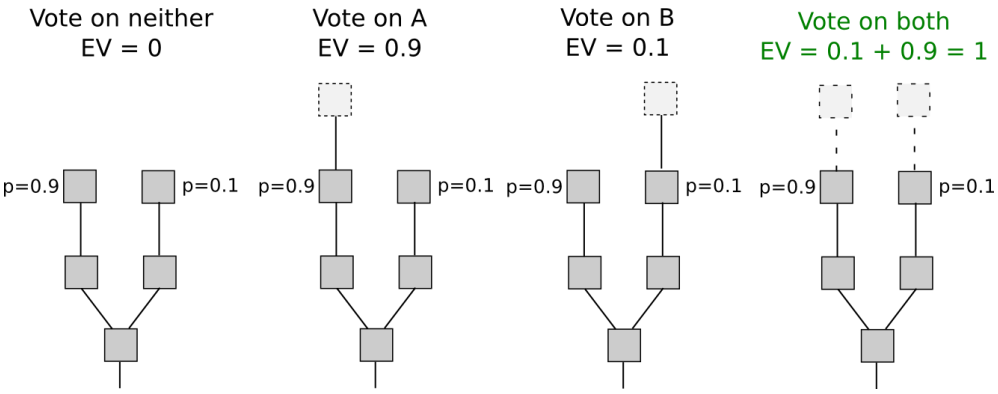
Clone in Desktop

Proof of work algorithms and chain-based proof of stake algorithms choose availability over consistency, but BFT-style consensus algorithms lean more toward consistency; Tendermint chooses consistency explicitly, and Casper uses a hybrid model that prefers availability but provides as much consistency as possible and makes both on-chain applications and clients aware of how strong the consistency guarantee is at any given time.

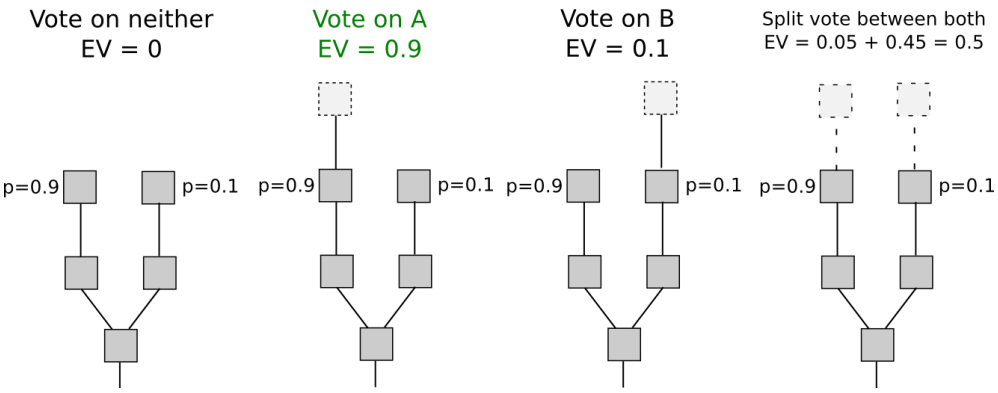
Note that Ittay Eyal and Emin Gun Sirer's selfish mining discovery, which places 25% and 33% bounds on the incentive compatibility of Bitcoin mining depending on the network model (ie. mining is only incentive compatible if collusions larger than 25% or 33% are impossible) has NOTHING to do with results from traditional consensus algorithm research, which does not touch incentive compatibility.

What is the "nothing at stake" problem and how can it be fixed?

In many early (all chain-based) proof of stake algorithms, including Peercoin, there are only rewards for producing blocks, and no penalties. This has the unfortunate consequence that, in the case that there are multiple competing chains, it is in a validator's incentive to try to make blocks on top of every chain at once, just to be sure:



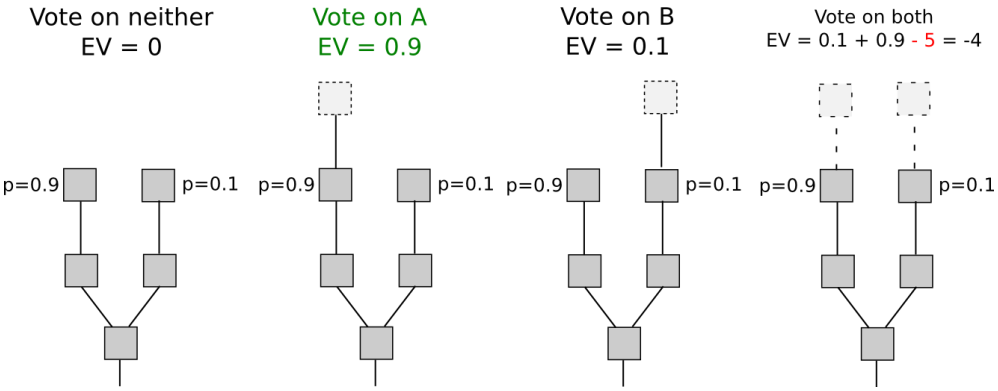
In proof of work, doing so would require splitting one's computing power in half, and so would not be lucrative:



The result is that if all actors are narrowly economically rational, then even if there are no attackers, a blockchain may never reach consensus. If there is an attacker, then the attacker need only overpower altruistic nodes (who would exclusively stake on the original chain), and not rational nodes (who would stake on both the original chain and the attacker's chain), in contrast to proof of work, where the attacker must overpower both altruists and rational nodes (or at least credibly threaten to: see P + epsilon attacks).

Some argue that stakeholders have an incentive to act correctly and only stake on the longest chain in order to "preserve the value of their investment", however this ignores that this incentive suffers from tragedy of the commons problems: each individual stakeholder might only have a 1% chance of being "pivotal" (ie. being in a situation where if they participate in an attack then it succeeds and if they do not participate it fails), and so the bribe needed to convince them personally to join an attack would be only 1% of the size of their deposit; hence, the required combined bribe would be only 0.5-1% of the total sum of all deposits. Additionally, this argument implies that any zero-chance-of-failure situation is not a stable equilibrium, as if the chance of failure is zero then everyone has a 0% chance of being pivotal.

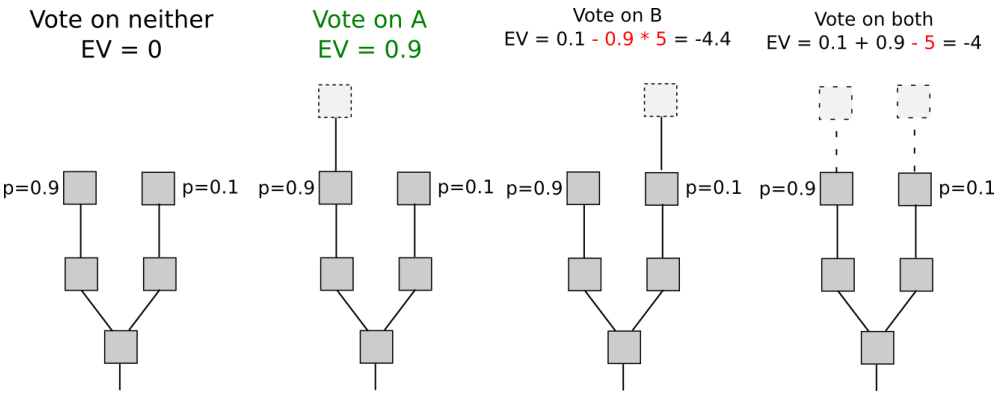
This can be solved via two strategies. The first, described in broad terms under the name "Slasher" here and developed further by Iddo Bentov here, involves penalizing validators if they simultaneously create blocks on multiple chains, by means of including proof of misbehavior (ie. two conflicting signed block headers) into the blockchain as a later point in time at which point the malfeasant validator's deposit is deducted appropriately. This changes the incentive structure thus:



Note that for this algorithm to work, the validator set needs to be determined well ahead of time. Otherwise, if a validator has 1% of the stake, then if there are two branches A and B then 0.99% of the time the validator will be eligible to stake only on A and not on B, 0.99% of the time the validator will be eligible to stake on B and not on A, and only 0.01% of the time will the validator will be eligible to stake on both. Hence, the validator can with 99% efficiency probabilistically double-stake: stake on A if possible, stake on B if possible, and only if the choice between both is open stake on the longer chain. This can only be avoided if the validator selection is the same for every block on both branches, which requires the validators to be selected at a time before the fork takes place.

This has its own flaws, including requiring nodes to be frequently online to get a secure view of the blockchain, and opening up medium-range validator collusion risks (ie. situations where, for example, 25 out of 30 consecutive validators get together and agree ahead of time to implement a 51% attack on the previous 19 blocks), but if these risks are deemed acceptable then it works well.

The second strategy is to simply punish validators for creating blocks on the *wrong* chain. That is, if there are two competing chains, A and B, then if a validator creates a block on B, they get a reward of +R on B, but the block header can be included into A (in Casper this is called a "dunkle") and on A the validator suffers a penalty of -F (possibly F = R). This changes the economic calculation thus:



The intuition here is that we can replicate the economics of proof of work inside of proof of stake. In proof of work, there is also a penalty for creating a block on the wrong chain, but this penalty is implicit in the external environment: miners have to spend extra electricity and obtain or rent extra hardware. Here, we simply make the penalties explicit. This mechanism has the disadvantage that it imposes slightly more risk on validators (although the effect should be smoothed out over time), but has the advantage that it does not require validators to be known ahead of time.

That shows how chain-based algorithms solve nothing-at-stake. Now how do BFT-style proof of stake algorithms work?

BFT-style (partially synchronous) proof of stake algorithms allow validators to "vote" on blocks by sending one or more types of signed messages, and specify two kinds of rules:

- **Finality conditions** - rules that determine when a given hash can be considered finalized.
- **Slashing conditions** - rules that determine when a given validator can be deemed beyond reasonable doubt to have misbehaved (eg. voting for multiple conflicting blocks at the same time). If a validator triggers one of these rules, their entire deposit gets deleted.

To illustrate the different forms that slashing conditions can take, we will give two examples of slashing conditions (hereinafter, "2/3 of all validators" is shorthand for "2/3 of all validators weighted by deposited coins", and likewise for other fractions and percentages). In these examples, "PREPARE" and "COMMIT" should be understood as simply referring to two types of messages that validators can send.

1. If `MESSAGES` contains messages of the form `["COMMIT", HASH1, view]` and `["COMMIT", HASH2, view]` for the same `view` but differing `HASH1` and `HASH2` signed by the same validator, then that validator is slashed.
2. If `MESSAGES` contains a message of the form `["COMMIT", HASH, view1]`, then UNLESS either view1 = -1 or there also exist messages of the form `["PREPARE", HASH, view1, view2]` for some specific `view2`, where `view2 < view1`, signed by 2/3 of all validators, then the validator that made the COMMIT is slashed.

There are two important desiderata for a suitable set of slashing conditions to have:

- **Accountable safety** - if conflicting `HASH1` and `HASH2` (ie. `HASH1` and `HASH2` are different, and neither is a descendant of the other) are finalized, then at least 1/3 of all validators must have violated some slashing condition.
- **Plausible liveness** - unless at least 1/3 of all validators have violated some slashing condition, there exists a set of messages that 2/3 of validators can produce that finalize some value.

If we have a set of slashing conditions that satisfies both properties, then we can incentivize participants to send messages, and start benefiting from economic finality.

What is "economic finality" in general?

Economic finality is the idea that once a block is finalized, or more generally once enough messages of certain types have been signed, then the only way that at any point in the future the canonical history will contain a conflicting block is if a large number of people are willing to burn very large amounts of money. If a node sees that this condition has been met for a given block, then they have a very economically strong assurance that that block will always be part of the canonical history that everyone agrees on.

There are two "flavors" of economic finality:

1. A block can be economically finalized if a sufficient number of validators have signed cryptoeconomic claims of the form "I agree to lose X in all histories where block B is not included". This gives clients assurance that either (i) B is part of the canonical chain, or (ii) validators lost a large amount of money in order to trick them into thinking that this is the case.
2. A block can be economically finalized if a sufficient number of validators have signed messages expressing support for block B, and there is a mathematical proof that *if some B' != B is also finalized under the same definition* then validators lose a large amount of money. If clients see this, and also validate the chain, and validity plus finality is a sufficient condition for precedence in the canonical fork choice rule, then they get an assurance that either (i) B is part of the canonical chain, or (ii) validators lost a large amount of money in making a conflicting chain that was also finalized.

The two approaches to finality inherit from the two solutions to the nothing at stake problem: finality by penalizing incorrectness, and finality by penalizing equivocation. The main benefit of the first approach is that it is more light-client friendly and is simpler to reason about, and the main benefits of the second approach are that (i) it's easier to see that honest validators will not be punished, and (ii) griefing factors are more favorable to honest validators.

Casper follows the second flavor, though it is possible that an on-chain mechanism will be added where validators can voluntarily opt-in to signing finality messages of the first flavor, thereby enabling much more efficient light clients.

So how does this relate to Byzantine fault tolerance theory?

Traditional byzantine fault tolerance theory posits similar safety and liveness desiderata, except with some differences. First of all, traditional byzantine fault tolerance theory simply requires that safety is achieved if 2/3 of validators are *honest*. This is a strictly easier model to work in; traditional fault tolerance tries to prove "if mechanism M has a safety failure, then at least 1/3 of nodes are faulty", whereas our model tries to prove "if mechanism M has a safety failure, then at least 1/3 of nodes are faulty, *and you know which ones, even if you were offline at the time the failure took place*". From a liveness perspective, our model is the easier one, as we do not demand a proof that the network *will* come to consensus, we just demand a proof that it does not get *stuck*.

Fortunately, we can show the additional accountability requirement is not a particularly difficult one; in fact, with the right "protocol armor", we can convert *any* traditional partially synchronous or asynchronous Byzantine fault-tolerant algorithm into an accountable algorithm. The proof of this basically boils down to the fact that faults can be exhaustively categorized into a few classes, and each one of these classes is either accountable (ie. if you commit that type of fault you can get caught, so we can make a slashing condition for it) or indistinguishable from latency (note that even the fault of sending messages too early is indistinguishable from latency, as one can model it by speeding up everyone's clocks and assigning the messages that *weren't* sent too early a higher latency).

What is "weak subjectivity"?

It is important to note that the mechanism of using deposits to ensure there is "something at stake" does lead to one change in the security model. Suppose that deposits are locked for four months, and can later be withdrawn. Suppose that an attempted 51% attack happens that reverts 10 days worth of transactions. The blocks created by the attackers can simply be imported into the main chain as proof-of-malfeasance (or "dunkles") and the validators can be punished. However, suppose that such an attack happens after six months. Then, even though the blocks can certainly be re-imported, by that time the malfeasant validators will be able to withdraw their deposits on the main chain, and so they cannot be punished.

To solve this problem, we introduce a "revert limit" - a rule that nodes must simply refuse to revert further back in time than the deposit length (ie. in our example, four months). Note that this rule is different from every other consensus rule in the protocol, in that it means that nodes may come to different conclusions depending on when they saw certain messages. The time that a node saw a given message may be different between different nodes; hence we consider this rule "subjective" (alternatively, one well-versed in Byzantine fault tolerance theory may view it as a kind of synchrony assumption).

However, the "subjectivity" here is very weak: in order for a node to get on the "wrong" chain, they must receive the original message four months later than they otherwise would have. This is only possible in two cases:

1. When a node connects to the blockchain for the first time.
2. If a node has been offline for more than four months.

We can solve (1) by making it the user's responsibility to authenticate the latest state out of band. They can do this by asking their friends, block explorers, businesses that they interact with, etc. for a recent block hash in the chain that they see as the canonical one. In practice, such a block hash may well simply come as part of the software they use to verify the blockchain; an attacker that can corrupt the checkpoint in the software can arguably just as easily corrupt the software itself, and no amount of pure cryptoeconomic verification can solve that problem.

Note that all of this is a problem only in the very limited case where a majority of previous stakeholders collude to attack the network and create an alternate chain; most of the time we expect there will only be one canonical chain to choose from.

Also, note that weak subjectivity assumptions exist in proof of work chains as well if the chain does any hard forks. Bitcoin has previously pulled off a hard fork with 2 months lead time through bitcoind 0.8.1, which fixed a database issue that made certain kinds of large blocks invalid and thereby allowed clients to process blocks that bitcoind 0.7 could not process, and so users had to download a new version of the software with 2 months' notice. This is itself a kind of weak subjectivity assumption, as users had to "log on" within that two-month period to download the update to stay on the correct chain.

Additionally, the social authentication can if needed even be automated in several ways. One is to bake it into natural user workflow: a BIP 70-style payment request could include a recent block hash, and the user's client software would make sure that they are on the same chain as the vendor before approving a payment (or for that matter, any on-chain interaction). The other is to use Jeff Coleman's universal hash time. If UHT is used, then a successful attack chain would need to be generated secretly *at the same time* as the legitimate chain was being built, requiring a majority of validators to secretly collude for that long.

Does weak subjectivity mean that a proof of stake chain must be "anchored" into a proof of work chain to be secure?

In short, no.

Elaborate?

Weak subjectivity by itself is a rather small addition to the security assumptions in a blockchain in practice, and definitely does not necessitate some proof-of-work-based outside source of truth to supplement it. To see why, consider the kind of situation where weak subjectivity by itself would compromise a blockchain's security. In such a world, powerful corporate or nation state actors would have the ability to somehow convince an entire community that block hash B was the block hash of block XXXYYY when most of them saw at the time and have stored in their own computers that the block hash of block XXXYYY was A, but for some reason such powerful actors **would not** have the ability to trick users into accepting a different location from where they download their client software.

Furthermore, the "anchoring" that advocates of such a scheme describe is not even all that secure. All anchoring proves is that a given block hash was *produced* at time T' < T; it does not prove that it was *published* at that time. Hence, a PoS chain anchored into a PoW chain could simply be attacked by a majority coalition that produces both chains in parallel, anchors both, publishes one, and then four months later publishes the other one.

One could get around this by embedding a fully-functional "light client" of the PoS chain into the PoW chain, which would reject the double-anchoring, but this would require the PoW chain to be feature-rich enough *to be able* to implement such a client - a property which most actually existing proof of work chains do not possess.

Can one economically penalize censorship in proof of stake?

Unlike reverts, censorship is much more difficult to prove. The blockchain itself cannot directly tell the difference between "user A tried to send transaction X but it was unfairly censored", "user A tried to send transaction X but it never got in because the transaction fee was insufficient" and "user A never tried to send transaction X at all". However, there are a number of techniques that can be used to mitigate censorship issues.

The first is censorship resistance by halting problem. In the weaker version of this scheme, the protocol is designed to be Turing-complete in such a way that a validator cannot even tell whether or not a given transaction will lead to an undesired action without spending a large amount of processing power executing the transaction, and thus opening itself up to denial-of-service attacks. This is what prevented the DAO soft fork.

In the stronger version of the scheme, transactions can trigger guaranteed effects at some point in the near to mid-term future. Hence, a user could send multiple transactions which interact with each other and with predicted third-party information to lead to some future event, but the validators cannot possibly tell that this is going to happen until the transactions are already included (and economically finalized) and it is far too late to stop them; even if all future transactions are excluded, the event that validators wish to halt would still take place. Note that in this scheme, validators could still try to prevent **all** transactions, or perhaps all transactions that do not come packaged with some formal proof that they do not lead to anything undesired, but this would entail forbidding a very wide class of transactions to the point of essentially breaking the entire system, which would cause validators to lose value as the price of the cryptocurrency in which their deposits are denominated would drop.

The second, described by Adam Back here, is to require transactions to be timelock-encrypted. Hence, validators will include the transactions without knowing the contents, and only later could the contents automatically be revealed, by which point once again it would be far too late to un-include the transactions. If validators were sufficiently malicious, however, they could simply only agree to include transactions that come with a cryptographic proof (eg. ZK-SNARK) of what the decrypted version is; this would force users to download new client software, but an adversary could conveniently provide such client software for easy download, and in a game-theoretic model users would have the incentive to play along.

Perhaps the best that can be said in a proof-of-stake context is that users could also install a software update that includes a hard fork that deletes the malicious validators and this is not that much harder than installing a software update to make their transactions "censorship-friendly". Hence, all in all this scheme is also moderately effective, though it does come at the cost of slowing interaction with the blockchain down (note that the scheme must be mandatory to be effective; otherwise malicious validators could much more easily simply filter encrypted transactions without filtering the quicker unencrypted transactions).

A third alternative is to include censorship detection in the fork choice rule. The idea is simple. Nodes watch the network for transactions, and if they see a transaction that has a sufficiently high fee for a sufficient amount of time, then they assign a lower "score" to blockchains that do not include this transaction. If all nodes follow this strategy, then eventually a minority chain would automatically coalesce that includes the transactions, and all honest online nodes would follow it. The main weakness of such a scheme is that offline nodes would still follow the majority branch, and if the censorship is temporary and they log back on after the censorship ends then they would end up on a different branch from online nodes. Hence, this scheme should be viewed more as a tool to facilitate automated emergency coordination on a hard fork than something that would play an active role in day-to-day fork choice.

How does validator selection work, and what is stake grinding?

In any chain-based proof of stake algorithm, there is a need for some mechanism which randomly selects which validator out of the currently active validator set can make the next block. For example, if the currently active validator set consists of Alice with 40 ether, Bob with 30 ether, Charlie with 20 ether and David with 10 ether, then you want there to be a 40% chance that Alice will be the next block creator, 30% chance that Bob will be, etc (in practice, you want to randomly select not just one validator, but rather an infinite sequence of validators, so that if Alice doesn't show up there is someone who can replace her after some time, but this doesn't change the fundamental problem). In non-chain-based algorithms randomness is also often needed for different reasons.

"Stake grinding" is a class of attack where a validator performs some computation or takes some other step to try to bias the randomness in their own favor. For example:

1. In Peercoin, a validator could "grind" through many combinations of parameters and find favorable parameters that would increase the probability of their coins generating a valid block.
2. In one now-defunct implementation, the randomness for block N+1 was dependent on the signature of block N. This allowed a validator to repeatedly produce new signatures until they found one that allowed them to get the next block, thereby seizing control of the system forever.
3. In NXT, the randomness for block N+1 is dependent on the validator that creates block N. This allows a validator to manipulate the randomness by simply skipping an opportunity to create a block. This carries an opportunity cost equal to the block reward, but sometimes the new random seed would give the validator an above-average number of blocks over the next few dozen blocks. See here for a more detailed analysis.

(1) and (2) are easy to solve; the general approach is to require validators to deposit their coins well in advance, and not to use information that can be easily manipulated as source data for the randomness. There are several main strategies for solving problems like (3). The first is to use schemes based on secret sharing or deterministic threshold signatures and have validators collaboratively generate the random value. These schemes are robust against all manipulation unless a majority of validators collude (in some cases though, depending on the implementation, between 33-50% of validators can interfere in the operation, leading to the protocol having a 67% liveness assumption).

The second is to use cryptoeconomic schemes where validators commit to information (ie. publish `sha3(x)`) well in advance, and then must publish `x` in the block; `x` is then added into the randomness pool. There are two theoretical attack vectors against this:

1. Manipulate `x` at commitment time. This is impractical because the randomness result would take many actors' values into account, and if even one of them is honest then the output will be a uniform distribution. A uniform distribution XORed together with arbitrarily many arbitrarily biased distributions still gives a uniform distribution.
2. Selectively avoid publishing blocks. However, this attack costs one block reward of opportunity cost, and because the scheme prevents anyone from seeing any future validators except for the next, it almost never provides more than one block reward worth of revenue. The only exception is the case where, if a validator skips, the next validator in line AND the first child of that validator will both be the same validator; if these situations are a grave concern then we can punish skipping further via an explicit skipping penalty.

The third is to use Iddo Bentov's "majority beacon", which generates a random number by taking the bit-majority of the previous N random numbers generated through some other beacon (ie. the first bit of the result is 1 if the majority of the first bits in the source numbers is 1 and otherwise it's 0, the second bit of the result is 1 if the majority of the second bits in the source numbers is 1 and otherwise it's 0, etc). This gives a cost-of-exploitation of `~C * sqrt(N)` where `C` is the cost of exploitation of the underlying beacons. Hence, all in all, many known solutions to stake grinding exist; the problem is more like differential cryptanalysis than the halting problem - an annoyance that proof of stake designers eventually understood and now know how to overcome, not a fundamental and inescapable flaw.

What would the equivalent of a 51% attack against Casper look like?

The most basic form of "51% attack" is a simple **finality reversion**: validators that already finalized block A then finalize some competing block A', thereby breaking the blockchain's finality guarantee. In this case, there now exist two incompatible finalized histories, creating a split of the blockchain, that full nodes would be willing to accept, and so it is up to the community to coordinate out of band to focus on one of the branches and ignore the other(s).

This coordination could take place on social media, through private channels between block explorer providers, businesses and exchanges, various online discussion forms, and the like. The principle according to which the decision would be made is "whichever one was finalized *first* is the real one". Another alternative is to rely on "market consensus": both branches would be briefly being traded on exchanges for a very short period of time, until network effects rapidly make one branch much more valuable with the others. In this case, the "first finalized chain wins" principle would be a Schelling point for what the market would choose. It's very possible that a combination of both approaches will get used in practice.

Once there is consensus on which chain is real, users (ie. validators and light and full node operators) would be able to manually insert the winning block hash into their client software through a special option in the interface, and their nodes would then ignore all other chains. No matter which chain wins, there exists evidence that can immediately be used to destroy at least 1/3 of the validators' deposits.

Another kind of attack is **liveness denial**: instead of trying to revert blocks, a cartel of >=34% of validators could simply refuse to finalize any more blocks. In this case, blocks would never finalize. Casper uses a hybrid chain/BFT-style consensus, and so the blockchain would still grow, but it would have a much lower level of security. If no blocks are finalized for some long period of time (eg. 1 day), then there are several options:

1. The protocol can include an automatic feature to rotate the validator set. Blocks under the new validator set would finalize, but clients would get an indication that the new finalized blocks are in some sense suspect, as it's very possible that the old validator set will resume operating and finalize some other blocks. Clients could then manually override this warning once it's clear that the old validator set is not coming back online. There would be a protocol rule that under such an event all old validators that did not try to participate in the consensus process take a large penalty to their deposits.
2. A hard fork is used to add in new validators and delete the attackers' balances.

In case (2), the fork would once again be coordinated via social consensus and possibly via market consensus (ie. the branch with the old and new validator set briefly both being traded on exchanges). In the latter case, there is a strong argument that the market would want to choose the branch where "the good guys win", as such a chain has validators that have demonstrated their goodwill (or at least, their alignment with the interest of the users) and so is a more useful chain for application developers.

Note that there is a spectrum of response strategies here between social coordination and in-protocol automation, and it is generally considered desirable to push as far toward automated resolution as possible so as to minimize the risk of simultaneous 51% attacks and attacks on the social layer (and market consensus tools such as exchanges). One can imagine an implementation of (1) where nodes automatically accept a switch to a new validator set if they do not see a new block being committed for a long enough time, which would reduce the need for social coordination but at the cost of requiring those nodes that do not wish to rely on social coordination to remain constantly online. In either case, a solution can be designed where attackers take a large hit to their deposits.

A more insidious kind of attack is a **censorship attack**, where >= 34% of validators refuse to finalize blocks that contain certain kinds of transactions that they do not like, but otherwise the blockchain keeps going and blocks keep getting finalized. This could range from a mild censorship attack which only censors to interfere with a few specific applications (eg. selectively censoring transactions in something like Raiden or the lightning network is a fairly easy way for a cartel to steal money) to an attack that blocks all transactions.

There are two sub-cases. The first is where the attacker has 34-67% of the stake. Here, we can program validators to refuse to finalize or build on blocks that they subjectively believe are clearly censoring transactions, which turns this kind of attack into a more standard liveness attack. The more dangerous case is where the attacker has more than 67% of the stake. Here, the attacker can freely block any transactions they wish to block and refuse to build on any blocks that do contain such transactions.

There are two lines of defense. First, because Ethereum is Turing-complete it is naturally somewhat resistant to censorship as censoring transactions that have a certain effect is in some ways similar to solving the halting problem. Because there is a gas limit, it is not literally impossible, though the "easy" ways to do it do open up denial-of-service attack vulnerabilities.

This resistance is not perfect, and there are ways to improve it. The most interesting approach is to add in-protocol features where transactions can automatically schedule future events, as it would be extremely difficult to try to foresee what the result of executing scheduled events and the events resulting from those scheduled events would be ahead of time. Validators could then use obfuscated sequences of scheduled events to deposit their ether, and dilute the attacker to below 33%.

Second, one can introduce the notion of an "active fork choice rule", where part of the process for determining whether or not a given chain is valid is trying to interact with it and verifying that it is not trying to censor you. The most effective way to do this would be for nodes to repeatedly send a transaction to schedule depositing their ether and then cancel the deposit at the last moment. If nodes detect censorship, they could then follow through with the deposit, and so temporarily join the validator pool en masse, diluting the attacker to below 33%. If the validator cartel censors their attempts to deposit, then nodes running this "active fork choice rule" would not recognize the chain as valid; this would collapse the censorship attack into a liveness denial attack, at which point it can be resolved through the same means as other liveness denial attacks.

That sounds like a lot of reliance on out-of-band social coordination; is that not dangerous?

Attacks against Casper are extremely expensive; as we will see below, attacks against Casper cost as much, if not more, than the cost of buying enough mining power in a proof of work chain to permanently 51% attack it over and over again to the point of uselessness. Hence, the recovery techniques described above will only be used in very extreme circumstances; in fact, advocates of proof of work also generally express willingness to use social coordination in similar circumstances by, for example, changing the proof of work algorithm. Hence, it is not even clear that the need for social coordination in proof of stake is larger than it is in proof of work.

In reality, we expect the amount of social coordination required to be near-zero, as attackers will realize that it is not in their benefit to burn such large amounts of money to simply take a blockchain offline for one or two days.

Doesn't MC => MR mean that all consensus algorithms with a given security level are equally efficient (or in other words, equally wasteful)?

This is an argument that many have raised, perhaps best explained by Paul Sztorc in this article. Essentially, if you create a way for people to earn $100, then people will be willing to spend anywhere up to $99.9 (including the cost of their own labor) in order to get it; marginal cost approaches marginal revenue. Hence, the theory goes, any algorithm with a given block reward will be equally "wasteful" in terms of the quantity of socially unproductive activity that is carried out in order to try to get the reward.

There are three flaws with this:

1. It's not enough to simply say that marginal cost approaches marginal revenue; one must also posit a plausible mechanism by which someone can actually expend that cost. For example, if tomorrow I announce that every day from then on I will give $100 to a randomly selected one of a given list of ten people (using my laptop's /dev/urandom as randomness), then there is simply no way for anyone to send $99 to try to get at that randomness. Either they are not in the list of ten, in which case they have no chance no matter what they do, or they are in the list of ten, in which case they don't have any reasonable way to manipulate my randomness so they're stuck with getting the expected-value $10 per day.
2. MC => MR does NOT imply total cost approaches total revenue. For example, suppose that there is an algorithm which pseudorandomly selects 1000 validators out of some

very large set (each validator getting a reward of $1), you have 10% of the stake so on average you get 100, and at a cost of $1 you can force the randomness to reset (and you can repeat this an unlimited number of times). Due to the central limit theorem, the standard deviation of your reward is $10, and based on other known results in math the expected maximum of N random samples is slightly under `M + S * sqrt(2 * log(N))` where `M` is the mean and `S` is the standard deviation. Hence the reward for making additional trials (ie. increasing N) drops off sharply, eg. with 0 re-trials your expected reward is $100, with one re-trial it's $105.5, with two it's $108.5, with three it's $110.3, with four it's $111.6, with five it's $112.6 and with six it's $113.5. Hence, after five retrials it stops being worth it. As a result, an economically motivated attacker with ten percent of stake will inefficiently spend $5 to get an additional revenue of $13, though the total revenue is $113. If the exploitable mechanisms only expose small opportunities, the economic loss will be small; it is decidedly NOT the case that a single drop of exploitability brings the entire flood of PoW-level economic waste rushing back in. This point will also be very relevant in our below discussion on capital lockup costs.

3. Proof of stake can be secured with much lower total rewards than proof of work.

What about capital lockup costs?

Locking up X ether in a deposit is not free; it entails a sacrifice of optionality for the ether holder. Right now, if I have 1000 ether, I can do whatever I want with it; if I lock it up in a deposit, then it's stuck there for months, and I do not have, for example, the insurance utility of the money being there to pay for sudden unexpected expenses. I also lose some freedom to change my token allocations away from ether within that timeframe; I could simulate selling ether by shorting an amount equivalent to the deposit on an exchange, but this itself carries costs including exchange fees and paying interest. Some might argue: isn't this capital lockup inefficiency really just a highly indirect way of achieving the exact same level of economic inefficiency as exists in proof of work? The answer is no, for both reasons (2) and (3) above.

Let us start with (3) first. Consider a model where proof of stake deposits are infinite-term, ASICs last forever, ASIC technology is fixed (ie. no Moore's law) and electricity costs are zero. Let's say the equilibrium interest rate is 5% per annum. In a proof of work blockchain, I can take $1000, convert it into a miner, and the miner will pay me $50 in rewards per year forever. In a proof of stake blockchain, I would buy $1000 of coins, deposit them (ie. losing them forever), and get $50 in rewards per year forever. So far, the situation looks completely symmetrical (technically, even here, in the proof of stake case my destruction of coins isn't fully socially destructive as it makes others' coins worth more, but we can leave that aside for the moment). The cost of a "Maginot-line" 51% attack (ie. buying up more hardware than the rest of the network) increases by $1000 in both cases.

Now, let's perform the following changes to our model in turn:

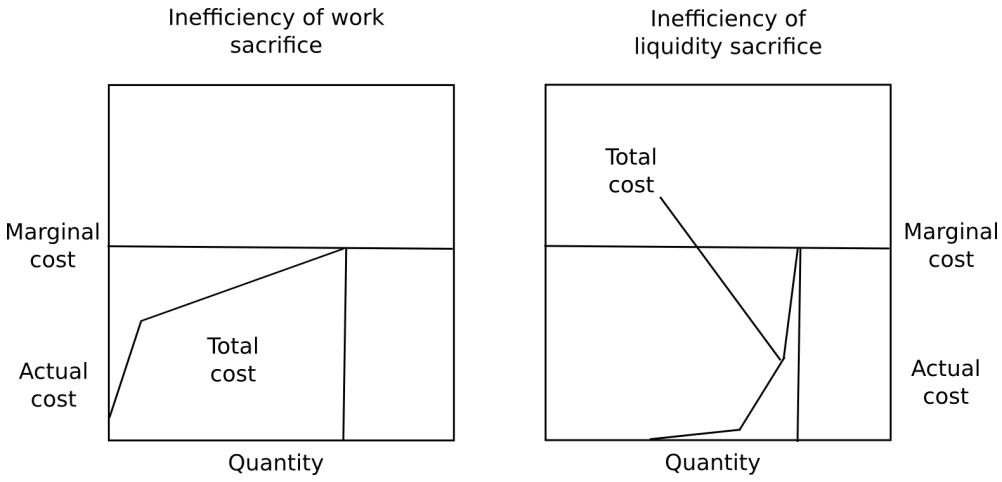
1. Moore's law exists, ASICs depreciate by 50% every 2.772 years (that's a continuously-compounded 25% per annum; picked to make the numbers simpler). If I want to retain the same "pay once, get money forever" behavior, I can do so: I would put $1000 into a fund, where $167 would go into an ASIC and the remaining $833 would go into investments at 5% interest; the $41.67 dividends per year would be just enough to keep renewing the ASIC hardware (assuming technological development is fully continuous, once again to make the math simpler). Rewards would go down to $8.33 per year; hence, 83.3% of miners will drop out until the system comes back into equilibrium with me earning $50 per year, and so the Maginot-line cost of an attack on PoW given the same rewards drops by a factor of 6.
2. Electricity plus maintenance makes up 1/3 of mining costs. We estimate the 1/3 from recent mining statistics: one of Bitfury's new data centers consumes 0.06 joules per gigahash, or 60 J/TH or 0.000017 kWh/TH, and if we assume the entire Bitcoin network has similar efficiencies we get 27.9 kWh per second given 1.67 million TH/s total Bitcoin hashpower. Electricity in China costs $0.11 per kWh, so that's about $3 per second, or $260,000 per day. Bitcoin block rewards plus fees are $600 per BTC * 13 BTC per block * 144 blocks per day = $1.12m per day. Thus electricity itself would make up 23% of costs, and we can back-of-the-envelope estimate maintenance at 10% to give a clean 1/3 ongoing costs, 2/3 fixed costs split. This means that out of your $1000 fund, only

$111 would go into the ASIC, $55 would go into paying ongoing costs, and $833 would go into hardware investments; hence the Maginot-line cost of attack is 9x lower than in our original setting.

3. Deposits are temporary, not permanent. Sure, if I voluntarily keep staking forever, then this changes nothing. However, I regain some of the optionality that I had before; I could quit within a medium timeframe (say, 4 months) at any time. This means that I would be willing to put more than $1000 of ether in for the $50 per year gain; perhaps in equilibrium it would be something like $3000. Hence, the cost of the Maginot line attack on PoS *increases* by a factor of three, and so on net PoS gives 27x more security than PoW for the same cost.

The above included a large amount of simplified modeling, however it serves to show how multiple factors stack up heavily in favor of PoS in such a way that PoS gets *more* bang for its buck in terms of security. The meta-argument for why this perhaps suspiciously multifactorial argument leans so heavily in favor of PoS is simple: in PoW, we are working directly with the laws of physics. In PoS, we are able to design the protocol in such a way that it has the precise properties that we want - in short, we can *optimize the laws of physics in our favor*. The "hidden trapdoor" that gives us (3) is the change in the security model, specifically the introduction of weak subjectivity.

Now, we can talk about the marginal/total distinction. In the case of capital lockup costs, this is very important. For example, consider a case where you have $100,000 of ether. You probably intend to hold a large portion of it for a long time; hence, locking up even $50,000 of the ether should be nearly free. Locking up $80,000 would be slightly more inconvenient, but $20,000 of breathing room still gives you a large space to maneuver. Locking up $90,000 is more problematic, $99,000 is very problematic, and locking up all $100,000 is absurd, as it means you would not even have a single bit of ether left to pay basic transaction fees. Hence, your marginal costs increase quickly. We can show the difference between this state of affairs and the state of affairs in proof of work as follows:



Hence, the *total* cost of proof of stake is potentially much lower than the marginal cost of depositing 1 more ETH into the system multiplied by the amount of ether currently deposited.

Note that this component of the argument unfortunately does not fully translate into reduction of the "safe level of issuance". It does help us because it shows that we can get substantial proof of stake participation even if we keep issuance very low; however, it also means that a large portion of the gains will simply be borne by validators as economic surplus.

Will exchanges in proof of stake pose a similar centralization risk to pools in proof of work?

From a centralization perspective, in both Bitcoin and Ethereum it's the case that roughly three pools are needed to coordinate on a 51% attack (4 in Bitcoin, 3 in Ethereum at the time of this writing). In PoS, if we assume 30% participation including all exchanges, then three exchanges would be enough to make a 51% attack; if participation goes up to 40% then the required number goes up to eight. However, exchanges will not be able to participate with all of their ether; the reason is that they need to accomodate withdrawals.

Additionally, pooling in PoS is discouraged because it has a much higher trust requirement - a proof of stake pool can pretend to be hacked, destroy its participants' deposits and claim a reward for it. On the other hand, the ability to earn interest on one's coins without oneself running a node, even if trust is required, is something that many may find attractive; all in all, the centralization balance is an empirical question for which the answer is unclear until the system is actually running for a substantial period of time. With sharding, we expect pooling incentives to reduce further, as (i) there is even less concern about variance, and (ii) in a sharded model, transaction verification load is proportional to the amount of capital that one puts in, and so there are no direct infrastructure savings from pooling.

A final point is that centralization is less harmful in proof of stake than in proof of work, as there are much cheaper ways to recover from successful 51% attacks; one does not need to switch to a new mining algorithm.

Can proof of stake be used in private/consortium chains?

Generally, yes; any proof of stake algorithm can be used as a consensus algorithm in private/consortium chain settings. The only change is that the way the validator set is selected would be different: it would start off as a set of trusted users that everyone agrees on, and then it would be up to the validator set to vote on adding in new validators.

[English | Deutsch | Español | Français | 日本語 | Română | فارسی | Italiano | 한국어 | 中文]