Vitalik Buterin Follow
Jan 15, 2017 · 8 min read · Listen

Parametrizing Casper: the decentralization/finality time/overhead tradeoff

As Casper continues to reach an increasingly stabilized form, there has been increased interest in the various parameters that are going to be set in the protocol, including the interest rate, fees, withdrawal period, speed, the minimum amount of ETH needed to stake, etc. This article will be the first in a series that attempts to describe the various tradeoffs involved, and will focus on decentralization and efficiency in protocols with the economic finality property.

First, we'll briefly define economic finality (we'll assume a threshold of 2/3; Vlad Zamfir will be quick to point out that we really want the ability for nodes to set their own thresholds).

Economic finality, version 1: state H is economically finalized if at least 2/3 of validators sign a message attesting to H, with the property that in any history not containing H, they lose their entire deposits.

Economic finality, version 2: state H1 is economically finalized if enough validators sign a message attesting to H1, with the property that if both H1 and a conflicting H2 are finalized, then there is evidence that can be used to prove that at least 1/3 of validators were malicious and therefore destroy their entire deposits.

Currently, both myself and Vlad are working on protocols that favor version 2. But regardless, note that both versions of economic finality require (i) at least 2/3 of validators to sign some message, and (ii) the network to verify these messages* (there

What this now means is that, **in order for state H to be finalized, the network must validate a message from at least 2/3 of the validator pool**. From this simple fact, we get a deep tradeoff between three things:

- **Time to finality:** how many seconds after H is proposed does H get finalized?
- **Decentralization** - defined here as the size of the validator pool, eg. a blockchain might have space for 1000 active validators). Note that this corresponds directly to **accessibility** - the minimum amount of ETH needed to become a validator, but more on this later.
- **Overhead**: how many messages per second do full nodes (including validators) need to verify?

Ideally, we want to minimize time to finality, maximize decentralization and minimize overhead. But unfortunately we can't do all three. Specifically, from the fact about validator messages given above, we get this mathematical equation:

$f * o \geq d * 2/3$

Where f is the time to finality in seconds, o is the overhead and d is the size of the validator pool (decentralization). This is a simple translation of the well-known law from physics "speed = distance / time": if you need to process d * 2/3 messages ("distance") in f seconds ("time"), then the number of messages per second ("speed") is d * 2/3 / f. Note that in reality, consensus protocols need multiple rounds of messaging, so $f * o \geq d * 2$ is likely, and in normal cases more than 2/3 of nodes will show up. Hence, for convenience, let's just use:

$f * o \geq d$

Now, let's stick some parameters into this to illustrate the point. Consider something PBFT-like where block n is always finalized before starting block n+1, and suppose the block time is 10 seconds. Suppose you want to support 500 validators. Then, $10* o \geq 500$, and so the overhead is at least 50 messages per second.

route is to go for 1000 validators, overhead of 1 message per second, and finality time of 1000 seconds (~15 minutes). You can feel free to stick your own numbers into this equation, and see what configurations are and are not possible. The important conclusion, once again, is that **we can't have all three nice things, at least completely, at the same time**.

So, there are four paths forward:

- Aim for **high decentralization and low overhead**, and let finality time be very long. Exchanges are used to finality times of over one hour, so tell them it's still going to be that way.
- Aim for **high decentralization and low finality time**, but require nodes to process high overhead. Tell full node operators to suck it up and accept this, and work on improving light client protocols.
- Aim for **low overhead and low finality time**, and make validating less accessible. Work on improving decentralized stake pooling software as a second-best to base-layer decentralization.
- Abandon the goal of economic finality.

Note that in the context of type 1 economic finality, you can try to get "partial" economic finality (eg. get 3.33% of validator deposits signing a message, which is still a lot of money, in 1/20 of the time that you need to get 67% of validator deposits), but in the context of type 2 economic finality you can't do this because you get no economic finality at all unless more than 50% of validators sign off on a state. Also, in either context, we can find ways to compromise between the three.

Deterministic threshold signatures for Stake Pools

Note that our work on elliptic curve pairing precompiles is important because, along with zk-SNARKs, the primitive can be used to implement deterministic threshold signatures. Hence, we could have a system where we have perhaps only 100 validators

of the participants in the pool approved the given hash. Each pool may itself have tens or even hundreds of participants.

Why can't we just use deterministic threshold signatures at the top level? First of all, they don't reveal *who* participated in them, just that enough people participated in them, and so we don't know whom to reward and whom to penalize. Second, they contradict the goal of cryptographic abstraction, where we want every user of the protocol, including validators, to be able to use whatever cryptography they want.

Theoretically, these pools could be managed in many different ways. The two most obvious paradigms are:

- **Fully open**: there exists a contract through which anyone can join.
- **Closed**: the pool is made between a group of friends who know each other.

Fully open pools are risky, because an attacker could join the pool with many identities and essentially 51% attack it; hence, participants in fully open pools run the risk of losing money at no fault of their own. However, the attackers would lose about as much money as their victims do (ie. the griefing factor is bounded by ~0.5–2). Closed pools are more secure, but of course they depend on people having friends whom they trust not to collude against them. Arguably, this kind of pool would allow for a rather tolerable level of decentralization in reality, as long as there are enough "slots" in the base validator set that many different pools can be created.

There is another type of stake pool, which uses trusted hardware instead of deterministic threshold signatures; see Loi Luu's recent blog post for more info. Note once again that such a design does not require the entire network to trust one particular brand of trusted hardware; rather, it only requires the participants in that particular pool to do so, and a robust network with heavy use of this scheme would likely include many different pools with many different combinations of trusted hardware solutions, as well as users that prefer deterministic threshold signatures or just "going solo".




From validator count to minimum staking ETH

Given a validator count (eg. d = 1000), the next question is: how does that translate into another variable that matters a lot for users personally - how much ETH do they need to become a (base-level) validator? There are a few formulas for this:

- **Set some minimum deposit size** (eg. 500 ETH), and let the amount of staking ETH, as well as the total number of validators (and hence either the finality time or the overhead) float
- **Set a maximum number of validators**, and increase the minimum deposit size toward infinity as the number of currently active validators approaches the maximum
- **Some intermediate policy between the two**, eg. one might consider setting the minimum deposit size to equal the number of currently staking validators

To examine the *worst case*, we once get a simple mathematical formula:

total_deposited_ether = min_deposit_size * number_of_validators

The reason why is hopefully obvious and uncontroversial. Now, what we can do is allow people who have more than the minimum deposit size to make a larger deposit, and count this deposit once. If this happens, then in reality we'll get results better than the worst-case scenario. Suppose Zipf's law is true, so the number of willing validators at a given deposit size is inversely proportional to that size, so eg. there are 10x more willing validators at 1000 ETH than there are at 10000 ETH. One quick way to assume there exists a willing validator of size largest_validator_size/n for every integer n ≥ 1, where largest_validator_size is itself an output of the model.

Now, fix some number_of_validators. The total amount of ETH deposited by the largest k validators, ie. the sum of largest_validator_size/n for 1 ≤ n ≤ k can be approximated as largest_validator_size * (ln(k) + 0.5) (where ln is the natural logarithm), so we get the equations:




If we fix any two variables, then we can solve for the other two. Suppose we fix the minimum to 500 ETH, and assume 10 million ETH deposited. Then, we can solve the equation, and get the results that we have:

- 2412 validators
- The largest validator has size 1206000 ETH

If we reduce the minimum to 50 ETH, then we get 19291 validators; if we increase it to 1500 ETH then we get 911 validators. Alternatively, we could do the math the other way: if we want to target 1000 validators, then the minimum will in equilibrium be ~1350 ETH. Note that this all assumes that it's possible to join with more than the minimum if you want to; if we don't have this property, then the tradeoffs worsen by a factor of ~log(n) as validators with larger deposits would have to split themselves into many accounts and make many signatures every time a state is finalized. For this reason, supporting variable depositor sizes above the minimum is considered to be a good thing.