Hybrid Casper FFG uses proof of stake to finalize

blocks, but still leaves to proof of work the responsibility of proposing

them. To reach full PoS, the task is thus to come up with a proof of stake block proposer.

Here is a simple one, and the one that is currently used for the sharding validator manager contract:

- All validators have the same deposit size
- Suppose there are N active validators. Take some source of randomness, R; compute R % N

, and take the validator at that index as the next validator.

This has the benefit of being extremely concise to implement and reason about, but has several flaws:

- If a validator gets the ability to propose a block, not only the validator but also everyone else will know ahead of time
- It does not support variable validator sizes
- In the case of Casper FFG, if it is intended for blocks to double as votes, then it's ideal to have all validators appear within each epoch, ideally in a permutation without repetitions, so that finalization can happen

The first problem is out of scope of this post (seehere for an ingenious solution using linkable ring signatures). To solve the second problem in isolation, the main challenge is developing a random sampling algorithm where each validator has a probability of being selected proportional to their deposit balance. This can be solved using a binary tree structure, as implemented here. This has O(log(n)) overhead, but the largest overhead only arises when balances are changed or validators are inducted or removed; simple random selection only takes O(log(n)) SLOAD calls, so <5000 gas.

For the third problem, the easiest solution is to set the epoch length to equal the number of active validators, and select a permutation at the start of the epoch. If there are N validators, this may be done by randomly selecting a P coprime to N (that is, something like P = sha3(R) % N; while gcd(P, N) != 1: P += 1

), and then cycling through (i * P) % N

for all 0 <= i < N

. One can also use Las Vegas algorithms (keep randomly picking validators until you find one that has not already been used in the current period); this reduces "lookahead transparency" at the cost of adding a bit more complexity and runtime.

Solving the second and third at the same time, however, is more challenging. There is a conflict between:

1. The needs of the consensus algorithm, which requires (almost) every validator to "speak up" at least once per period
2. The goal of fairness, which requires the expected number of blocks created by a validator to be proportional to their deposit size
3. The goal of shortening period length to reduce time-to-finality.

Optimally reconciling (1) and (3) requires each validator to have exactly one slot, but this contradicts fairness.

One can combine (1) and (2) with a hybrid algorithm; for example, one can imagine an algorithm that normally just randomly selects validators, but if it notices that there are too many not-yet-selected validators compared to the number of blocks left in the period it continues looking until it finds a not-yet-selected validator. Alternatively, one could use the (i * P) % N

round robin interspersed with a random algorithm where the probability of getting selected is proportional to balance - min_balance

. In all cases, the expected period length is total_balance / min_balance

. If we assume account balances are distributed according toZipf's law, that's roughly N * log(N).

One can achieve an even better result with an exponential backoff scheme: use any form of randomness to select validators, then start the epoch length at some very small number. Chances are, the first epoch will be so small that only a small percentage of validators will be able to participate, and so hitting the 2/3 threshold is impossible. However, every time we fail to come to consensus, we simply double the epoch length until eventually we get to 2/3. This mechanism has the advantage that it does not rely on a high min_balance

; if there is a large number of very-small-balance validators but they only make up a small portion of the total deposit, most of the time the larger validators will be able to finalize by themselves.

Currently, I favor this latter scheme.

The other important topic that needs to be covered is parametrizing the[overhead / finality time / node count tradeoff](#) See also Vlad's expression in triangle form:

Mathematically speaking, it's the result of a simple inequality:

overhead \ge \frac{2 * NumValidators}{TimeToFinality}

Achieving finality requires two messages from every validator that must be processed by the blockchain, so the rest follows trivially. There are two ends of the tradeoff that are obvious:

1. PBFT et al: every node sends a "vote" every block, so low time to finality, hence low node count or high overhead
2. Chain-based: votes are

blocks, so same low overhead and high node count potential as any other chain-based algo, but high time to finality

I believe that the optimal solution lies in the middle. Casper FFG already does this, supporting:

- A medium amount of nodes (1000-3000, not an unlimited amount like "naive PoS" chains bit also not ~20-30 like many DPOS chains)
- A medium time to finality (not 5s, but also not "never" like naive PoS chains) and a medium
- A medium amount of overhead (~1-2 messages per second, not ~0.07 per sec like the current ethereum chain but also not tens or hundreds per second like many DPOS chains)

But with full proof of stake, there is a binary divide: either votes are blocks, in which case you go full chain-based, or votes are not blocks, in which case you have a redundancy between the two, with the protocol having to handle the overhead of both. One could simply accept the redundancy, accepting how Casper FFG works, but then adding in a PoS block proposer, but we may be able to do better. If we are okay with having 1-2 messages per second of consensus overhead, then we can have these consensus messages contain references to a small list of transactions, and use fancy directed acyclic graph techniques to gather these transactions into a history, ensuring convergence happens as quickly as possible (insert coindesk article and reddit posts saying "Ethereum foundation is exploring hashgraph/DAG technology!!!1!!" here).

There are two other possibilities:

1. With sharding, Casper votes can also serve double-duty as collation headers. Because there are 100 shards, with a block time of ~75 seconds (period length 5 blocks) on each shard, the total overhead of the system just happens to also be ~1-2 messages per second.
2. When a block is created, a random set of N validators is selected that must validate that block for it to be possible to build another block on top of that block. At least M of those N must sign off. This is basically [proof of activity](#) except with both layers being proof of stake; it brings the practical benefit that a single block gives a very solid degree of confirmation - arguably the next best thing to finality - without adding much complexity. This could be seen as bringing together the best of both worlds between committee algorithms and full-validator-set algorithms.

Currently, I favor (2).