This proposal deals with the following questions:

- How is the committee that can make a cross-link for shard i

determined?

- How is the committee that can participate in a CAS for shard i

determined?

- What messages get included in the beacon chain, and what do those messages commit to?
- When do committees make cross-links?
- What are the incentives?
- How does the Casper FFG cycle fit in?

It is designed to be mostly agnostic to whether or notBLS aggregation is used, though I will mention in a few places modifications specific to that proposal that would need to be made.

Desiderata

1. Every message is dual-used, so we can maximize the impact of each signature.
2. Every validator can participate in the Casper FFG cycle in every round (this ensures that during a quadratic leak phase, innocent validators will not lose)
3. Every validator's validation load is maximally stable
4. No centralization issues
5. Maximum simplicity

I will try to derive the scheme from the desiderata as directly as possible. First of all, (1) implies that we ideally want every message that gets included in the beacon chain to simultaneously carry two pieces of data: (i) Casper FFG-related epoch source, checkpoint, etc, and (ii) a shard collation hash for a cross-link. One of the implications of (3) is that we want each validator to be assigned to one shard at a time, and not a Poisson distribution; this would also deal with (2). Hence, the only solution for cross-link assignment is that during every Casper FFG epoch, we come up with a permutation of the N validators, assign the first k

to shard 1, the next k

to shard 2, etc. How k

is decided (and therefore how the shard count is decided) is an open question; I see the following three solutions:

- Fix k = 135

, set shard_count = N/135

- Fix shard_count = 100

, set k = N/100

- Fix k = shard_count = sqrt(N)

I am as yet undecided between these three

A possible simple algorithm is as follows:

def reshuffle(): while 1: x = random.randrange(validator_count) if gcd(x, validator_count) == 1: self.shuffling_factor = x return

def which_shard(validator_index): return ((validator_index * self.shuffling_factor) % validator_count) // validators_per_shard

Another challenge is this: if there has not been a successful cross-link with some shard for a long time (eg. 1 month), then the next cross-link will be confirming a month's worth of blocks. If a newly selected committee has only an hour-long epoch within which to verify all of these blocks, then they will not be able to verify them on time.

A natural fix is exponential backoff, described here. However, it would be difficult to create a shard allocation mechanism which can hold one shard's cross-link committee constant and not others (ie. it's certainly mathematically possible, but it would be fairly long in terms of lines of code), so the natural solution is to make all

shards exponentially backoff (that is, the committee switch frequency of all shards is dependent on the time since the oldest most-recent cross-link for a shard was included).

The algo could look something like this:

let current_epoch be the current epoch let last_cross_link[i] be the last epoch a cross-link from shard i was included let last_reshuffle be the last time committees were reshuffled

time_since_last_crosslink = current_epoch - min(last_cross_link[1]...last_cross_link[shard_count]) time_since_reshuffle = current_epoch - last_reshuffle

if time_since_reshuffle >= time_since_last_crosslink / 2: reshuffle_committees()

The next question is, who can participate in making cryptoeconomic aggregate signatures on each shard chain block? Here, the natural answer seems to be: the same committee as the committee that can make a cross-link. This achieves three nice results:

- The number of nodes that need to verify that shard is reduced, as we're merging the two functions
- CAS signatures are naturally also cross-link signatures at the same time
- We get signatures for every block that could be used for cross-links floating around "by default"

The next question after this is: incentives and coordination. Specifically, (i) how does the system get validators to agree on which block to confirm a cross-link for, and (ii) how does the system set up incentives to achieve this coordination?

There are two forms of aggregation: off-chain aggregation and on-chain aggregation. In on-chain aggregation, cross-link signatures are submitted separately, and counted on-chain. In off-chain aggregation, cross-link signatures must be submitted as a group. Pure off-chain aggregation is problematic because it depends on a single aggregator who has the ability to exclude votes; pure on-chain aggregation is problematic because it could easily lead to 50/50 splits between two different cross-link blocks in the same chain.

Here is a hybrid proposal:

- Suppose an epoch has epoch_length = 2 * shard_count

blocks, and in the current epoch there are remaining_blocks

blocks remaining.

- If no signatures for a given shard have yet been submitted, then new signatures can only come in a package of committee_size * remaining_blocks / epoch_length

or more

- If signatures for a given shard have already been submitted, then further signatures for that shard can be submitted later in the epoch but only if they confirm the same hash
- There's a per-block limit of 1.5 * committee_size

total signatures in every block.

The decreasing limit ensures that the aggregation mechanism works similarly even in cases where more than 1/3 of validators are offline, and so sufficiently sized committees for any given cross-link cannot be reached. The block proposer receives a reward for every signature submitted to encourage maximum inclusion; in the default (no quadratic rollback) case, the reward could even be fairly large, eg. 1/3 of the reward given to the signatures that participate in the cross-link. It can also be required that the first signature inclusion for some shard in some epoch must include all

of the signatures provided as part of some specific CAS.

This combines the benefits of off-chain aggregation with an on-chain ability to circumvent validators that maliciously exclude specific signatures.

Multiple signatures for the same shard can be aggregated via BLS on-chain aggregation in the short-term; in the longer term, it seems reasonable to expect that every block will have attached a STARK that attests to the validity of every transaction.