The purpose of this document is to give a “mini-spec” for the beacon chain mechanism for the purpose of security analysis, safety proofs and other academic reasoning, separate from relatively irrelevant implementation details.

(Thanks @josojo for edits!)

Beacon chain stage 1 (no justification, no dynasty changes)

Suppose there is a validator set V = {V_1 ... V_n}

(we assume for simplicity that all validators have an equal amount of “stake”). We divide time into six-second-long slots

; if the genesis timestamp of the system is T_0

, then slot i

consists of the time period [T_0 + 6i, T_0 + 6(i+1))

, and consider each set of 64 adjacent slots to be an “epoch”. For each epoch, we pseudorandomly choose disjoint subsets of V

, S_1 .... S_{64}

with |S_i| = \frac{|V|}{64}

, where |x|

refers to set size (ie. the number of validators, or whatever other kind of object, in x

), and within each S_i

we select one particular member as the “proposer” for slot i

in that epoch.

Note: if an attacker controls less than \frac{1}{3}

of the stake, then if |S_i| \ge 892

there is a less than 2^{-80}

chance that the attacker controls more than \frac{1}{2}

of S_i

, and there is a less than 2^{-100}

chance that an attacker controls all 64 indices in a given span i_k .... i_{k+63}

. We can assume that it is certain that neither of these things will happen (that is, we can assume there exists a substring of validator indices p_{i_1}, p_{i_2} ...

with p_{i_{k+1}} - p_{i_k} < 64

and that every S_i

is majority honest).

When (from the point of view of the validator’s local clock) slot i

begins, the proposer at slot i

is expected to create (“propose”) a block, which contains a pointer to some parent block that they perceive as the “head of the chain”, and includes all of the attestations

that they know about that have not yet been included into that chain.

After receiving a valid block at slot i

, or after waiting for 3 seconds after the start of slot i

and not receiving such a valid block, validators in S_{i\ mod\ 64}

are expected to determine what they think is the “head of the chain” (if all is well, this will generally be the newly published

block), and publish a (signed) attestation, [current_slot, h]

, where h

is the hash of the head of the chain that they know about, and current_slot

is the current slot number.

The fork choice used is “latest message driven GHOST”. The mechanism is as follows:

1. Set H

to equal the genesis block.

1. Let M = [M_1 ... M_n]

be the most-recent messages (ie. highest slot number messages) of each validator.

1. Choose the child of H

such that the subset of M

that attests to either that child or one of its descendants is largest; set H

to this child.

1. Repeat (2) until H

is a block with no descendants.

Claims:

- Safety

: assuming the attacker controls less than \frac{1}{3}

of V

, and selected the portion of V

to control before the validators were randomly sorted, the chain will never revert (ie. once a block is part of the canonical chain, it will be part of the canonical chain forever).

- Incentive-compatibility

: assume that there is a reward for including attestations, and for one’s attestation being included in the chain (and this reward is higher if the attestation is included earlier). Proposing blocks and attesting to blocks correctly is incentive-compatible.

- Randomness fairness

: in the long run, the attacker cannot gain by manipulating the randomness

Beacon chain stage 2 (add justification and finalization)

We cluster slots into epochs: Every 64th slot is an epoch_boundary_slot

and all 63 consecutive slots of an epoch_boundary_slot

belong to the same epoch. The chain state keeps track of a map \mathit{justified_hashes}

, which starts at {(0, genesis)}

, and will add new (slot, hash)

pairs over time. A valid attestation now references two additional variables: an epoch boundary hash

(ie. the hash of the highest-slot block

in the chain such that floor(\frac{block.slot}{64}) < floor(\frac{attestation.current_slot}{64})

), and a latest justified hash

(the highest-epoch hash in the justified_hashes

of the block referenced by the epoch boundary hash). Attestations published can be included in the chain, but only if the attestation's latest justified hash equals the highest-slot hash in \mathit{justified_hashes}

.

Note that because the state of the chain keeps track of the latest justified hash implicitly, all honest validators that vote for the same epoch boundary hash will vote for the same latest justified hash.

At each epoch boundary (ie. when processing a block where floor(\frac{b.slot}{64}) > floor(\frac{b.parent.slot}{64})

), the state transition function performs the following steps.

Suppose that in a chain, the most recent epoch boundary blocks are B1, B2, B3, B4 (B4 being the most recent), with the epoch boundary corresponding to B4 having slot B4_slot

(then B3 has slot B3_slot = B4_slot - 64

, etc). If the chain has accepted attestations from \frac{2}{3}

of all

validators that specify B4 as the epoch boundary block, then it admits (B4_slot, B4)

into \mathit{justified_hashes}

.

That is, an epoch boundary block is "justified" if \frac{2}{3}

of all

validators sign a message whose epoch boundary hash is that block, and the block becoming part of \mathit{justified_hashes}

in some chain means that the block is justified and the information proving that the block is justified has been included in that chain.

In the following three cases, we "finalize" a block:

- If B4 and B3 are in \mathit{justified_hashes}

and the attestations that justified B4 used B3 as the latest justified hash, then we finalize B3.

- If B4, B3 and B2 are in \mathit{justified_hashes}

and the attestations that justified B4 used B2 as the latest justified hash, then we finalize B2.

- If B3, B2 and B1 are in \mathit{justified_hashes}

and the attestations that justified B3 used B1 as the latest justified hash, then we finalize B1.

Note that the algorithm can work if only the first rule exists, but we add the other two rules to satisfy the cases where it takes longer for attestations to get included in the chain.

We change the fork choice rule above so that instead of starting H

from the genesis block, it works as follows:

- Set H_F

to the highest-slot finalized block.

- Set H_J

to the highest-slot block which is justified and the client has known is justified for some time (precisely, if T_F

is the time the client learned about the highest-slot finalized block, T_J

is the time the client learned about some given justified block, then the client takes that block into account starting at slot T_F + (T_J - T_F) * 3

- Start the fork choice from H_J

.

We then add two slashing conditions:

- A validator cannot make two distinct attestations in the same epoch
- A validator cannot make two attestations with epoch boundary slots t1

, t2

and justified slots s1

, s2

such that s1 < s2 < t2 < t1

and s2

& t2

are consecutive epoch boundary slots.

Claims:

- Safety

: once a block becomes finalized, it will always be part of the canonical chain as seen by any node that has downloaded the chain up to the block and the evidence finalizing the block, unless at least a set of validators V_A

with |V_A| \ge |V| * \frac{1}{3}

violated one of the two slashing conditions (possibly a combination of the two).

- Plausible liveness

: given an “honest” validator set V_H

with |V_H| \ge |V| * \frac{2}{3}

, V_H

by itself can always finalize a new block without violating slashing conditions.

- Real liveness

: assuming that after some time T

network latency < 6 sec, the network will justify blocks, and an attacker with less than \frac{1}{3}

of V

cannot prevent this.

Arguments for safety and plausible liveness are equivalent tohttps://arxiv.org/abs/1710.09437. The argument for real liveness is roughly as follows. If no “new justified block” events happen, then the LMD GHOST rule is stable (this is assumed to be already proven) and clients will finalize a new block after 2 epochs (that is, within 3 epochs of any given point in time, as specific points in time could be in the middle of an epoch). The T_F + (T_J - T_F) * 3

rule ensures that “new justified block” events can only take place once every three epochs, so eventually a block will get finalized.

Beacon chain stage 3: adding dynamic validator sets

Every block B

comes with a subset of validators S_B

, with the following restrictions:

- Define the dynasty

of a block recursively: dynasty(genesis) = 0

, generally dynasty(B) = dynasty(parent(B))

except

when the processing of B

finalizes a block, in which case dynasty(B) = dynasty(parent(B)) + 1

.

- Each block B

has a local validator set

LVS(B)

. For two blocks in the chain, if B_1

and B_2

, dynasty(B_2) - dynasty(B_1) = k

, then |LVS(B_1)\ \cap\ LVS(B_2)| \ge LVS(B_1) * (1 - \frac{k}{64})

(and likewise wrt LVS(B_2)

). That is, at most \frac{1}{64}

of the local validator set changes with each dynasty.

Claims:

- All of the above claims hold, with appropriate replacements of V

with LVS(...)

, except with fault tolerance possibly reduced from \frac{1}{3}

to approximatively 30%.