For background on CBC Casper, see: https://vitalik.ca/general/2018/12/05/cbc_casper.html

There are two key implementation challenges with CBC Casper as described in this post:

- Evaluating whether or not a block is valid requires executing the LMD GHOST fork choice rule, which takes n * log^2(t)

time with n

validators and t

slots after genesis (possibly shortenable to n * log(t)

with better algorithms, but the dominant factor making things difficult is the n

- Evaluating whether or not two messages M_1

and M_2

constitute a slashable equivocation

Here, I present viable (both computation-wise and human-complexity-wise) algorithms for doing both.

Basic chain structure

The chain structure is generally identical to the beacon chain: there are blocks, and there are attestations of blocks, and attestations of blocks can get re-included into the chain. A validator can legally make a maximum of one attestation per epoch.

However, we add one difference: block headers that are not part of the chain can be added into the main chain as uncles. An attestation can only be included if it is attesting to either a block that is in the chain or a block that has been already included as an uncle.

The header of each block contains a list of 32 items where the k

'th item is the ancestor of that block at the most recent multiple of 2^k

. This allows us to compute, using data in the state, the ancestor of an arbitrary block at an arbitrary height in log time, by hopping down successively lower powers of 2 to reach any desired height.

Validity

The above changes to the chain allow us to theoretically execute the LMD GHOST fork choice rule (eg. see here for a python implementation) using data in the state. However, this is still expensive, because we are dealing with a graph calculation involving potentially a million validators.

There are two ways to simplify this. First, in the beacon chain, we only have 64 blocks per epoch, and so generally most validators' most recent messages will be on a few blocks. Hence, if we treat all most-recent-messages on one block as a single unit, we can greatly decrease computation costs in the normal case, but not in the exceptional case where many validators go offline at different times.

Second, we can make a more unconditional improvement to O(log^2(n))

efficiency, see Bitwise LMD GHOST

Slashing conditions

In the state of the chain, we store an object that represents the epoch in which each validator most recently made an attestation that the chain knows about. The intended slashing condition is that if you make an attestation M_n

, then any previous attestations M_1 ... M_{n-1}

that you made must have been already included in the chain.

Suppose a scenario like this happens, where blocks are in yellow, and m1, m2, m3 are attestations of one particular validator (in grey).

[

%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BA%7Bbg%3Ayellow%7D%5D----
%3E%5Bm1%5D%2C%5Bm1%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BB%7Bbg%3Ayellow%7D%5D----

%3E%5Bm2%5D%2C%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%2C%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5Bm2%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%7Bbg%3Ayellow%7D%5D%20-----
%3E%20%5BC%7Bbg%3Ayellow%7D%5D%2C%5BC%7Bbg%3Ayellow%7D%5D%20-----%3E%20%5Bm3%5D

194×1070

](https://yuml.me/diagram/scruffy/class/%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BA%7Bbg%3Ayellow%7D%5D----
%3E%5Bm1%5D%2C%5Bm1%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BB%7Bbg%3Ayellow%7D%5D----
%3E%5Bm2%5D%2C%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%2C%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5Bm2%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%7Bbg%3Ayellow%7D%5D%20-----
%3E%20%5BC%7Bbg%3Ayellow%7D%5D%2C%5BC%7Bbg%3Ayellow%7D%5D%20-----%3E%20%5Bm3%5D)

This is the validator simply legitimately changing their mind and following the fork choice, as we know that m2 was included in the chain. The state of C, which m3 contains a signature of, will show the validator's last signed epoch as being epoch(m2)

.

Now, consider this case (see the arrow from m2 to the block before C removed). This is now equivocation, as the validator is signing messages that do not string together into a single history.

[

%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BA%7Bbg%3Ayellow%7D%5D----
%3E%5Bm1%5D%2C%5Bm1%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BB%7Bbg%3Ayellow%7D%5D----
%3E%5Bm2%5D%2C%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%2C%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%7Bbg%3Ayellow%7D%5D%20-----
%3E%20%5BC%7Bbg%3Ayellow%7D%5D%2C%5BC%7Bbg%3Ayellow%7D%5D%20-----%3E%20%5Bm3%5D

215×735

](https://yuml.me/diagram/scruffy/class/%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BA%7Bbg%3Ayellow%7D%5D----
%3E%5Bm1%5D%2C%5Bm1%5D%20----
%3E%20%5BB%7Bbg%3Ayellow%7D%5D%2C%5BB%7Bbg%3Ayellow%7D%5D----
%3E%5Bm2%5D%2C%5BA%7Bbg%3Ayellow%7D%5D%20----
%3E%20%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%2C%5B%20%20%7Bbg%3Ayellow%7D%20%20%5D%20----
%3E%20%5B%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%7Bbg%3Ayellow%7D%5D%20-----
%3E%20%5BC%7Bbg%3Ayellow%7D%5D%2C%5BC%7Bbg%3Ayellow%7D%5D%20-----%3E%20%5Bm3%5D)

Here, the state that m3 signs over will show the last signed epoch as being epoch(m1)

.

So now our slashing condition is as follows: you can slash a validator by providing:

- Their signature of some block C with epoch e3
- A Merkle branch from C showing that in that block their last signed known epoch was e1
- Their signature of some block B with epoch e2

Where e3 > e2 > e1. So this is simply a different kind of no-surround rule, where the last-justified-epochs are per-validator values stored in a Merkle tree instead of a single value used by everyone.