Casper CBC roughly works as follows:

- Validators make messages.
- Each message specifies a block that the validator is voting on, and also specifies the most recent message that the validator received from each other validator.
- The block that the validator is voting on must be equal to or descended from the block that is the head under the GHOST fork choice rule using as inputs the other validators' latest messages.
- The only slashing conditions are (i) the above rule, (ii) a validator cannot make two messages with the same sequence number, (iii) a validator cannot in a later message refer to messages with an earlier sequence number than the messages that the validator referred to in an earlier message.
- Finality is endogenous: at some point, when there are multiple rounds of validators voting on descendants of X, it's mathematically impossible for the head to switch to being not-X without a large portion of validators making invalid messages. A lower bound on this amount can be detected and measured using various heuristics.

Here's an example of the GHOST fork choice rule in action. The letters A,B,C,D,E represent the five most recent votes.

[

%5B%20%20%5D%20-
%3E%20%5B%20%20%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%20%20%7Bbg%3Ayellow%7D%5D%20-
%3E%20%5BA%20B%5D%2C%5B%20%20%5D%20-
%3E%20%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%2C%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%20-
%3E%20%5BC%7Bbg%3Ablue%7D%5D%2C%5BC%7Bbg%3Ablue%7D%5D%20-
%3E%20%5BD%7Bbg%3Aorange%7D%5D%2C%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%20-
%3E%20%5BE%7Bbg%3Ared%7D%5D

258×568

](https://yuml.me/diagram/scruffy/class/%5B%20%20%5D%20-
%3E%20%5B%20%20%20%7Bbg%3Ayellow%7D%5D%2C%5B%20%20%20%7Bbg%3Ayellow%7D%5D%20-
%3E%20%5BA%20B%5D%2C%5B%20%20%5D%20-
%3E%20%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%2C%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%20-
%3E%20%5BC%7Bbg%3Ablue%7D%5D%2C%5BC%7Bbg%3Ablue%7D%5D%20-
%3E%20%5BD%7Bbg%3Aorange%7D%5D%2C%5B%20%20%20%20%7Bbg%3Agreen%7D%5D%20-
%3E%20%5BE%7Bbg%3Ared%7D%5D)

The first choice is between green and yellow. Green wins because there are three votes that descend from green, and only two that descend from its competitor yellow. The second choice is between red and blue. Blue wins by 2 vs 1. Blue has only one child, orange, so orange wins.

The major efficiency issue with doing this naively is obvious: every message needs to refer to every other recent message that it has seen, potentially leading to O(N^2)

data complexity.

This post explores one particular strategy for alleviating the data complexity. Instead of every validator's vote being the evaluation of the GHOST fork choice rule on every

other validator's message, validators are explicitly assigned private committees of m

other validators (likely 32 \le m \le 256

), and in their messages must include a reference to the signatures of these m

validators. This reference could be by sequence number, or by position in which those signatures were already included into the chain. The slashing condition can simply check that these messages actually do represent the GHOST fork choice evaluation of these m

other messages, and that the counters always increment.

Stated more concretely:

- For a chain to accept a message, either (i) the message must be voting for a block in the chain or (ii) the off-chain block that the message is voting for must have been included as an uncle
- For a chain to accept an uncle, the uncle's parent must either (i) be part of the chain or (ii) have already been included in the chain as an uncle

- For a chain to accept a message, all messages that the message refers to in its most-recent set must have been accepted.
- Every message has a sequence number. For a chain to accept a message with sequence number n

, it must have already accepted a message from that validator with sequence numbers 0....n-1

,.

- A validator can be slashed for two messages with the same sequence number, or for a message voting x

where the evidence included in the message does not justify voting x

.

If the committees are large enough, they will approximate the entire validator set, and you can heuristically determine the number of validators that will need to make attributable failures in order to illegally shift the fork choice from A to B. Here is some code that can calculate this:

github.com

ethereum/research/blob/659f0b31f9337b3e7ee4bde45cdb93c0ed4fd390/graph_cbc/graph_cbc.py

import random

VALIDATORS = 5000 EDGES = 255 FINALITY = 4000

assert EDGES % 2 == 1

neighbors = list(range(VALIDATORS)) edgelist = neighbors * EDGES random.shuffle(edgelist) edges = [edgelist[i*EDGES:i*EDGES+EDGES] for i in range(VALIDATORS)]

last_votes = '1' * FINALITY + '0' * (VALIDATORS - FINALITY)

while 1: new_zeroes = [] for i in range(VALIDATORS): votes_for_0 = len([e for e in edges[i] if last_votes[e] == '0']) if votes_for_0 * 2 > EDGES:

This file has been truncated. show original

The result is that with m \approx 256

, the fault tolerance seems to be close to ~20%, very close to the maximum 25% possible within two rounds with Casper CBC. Though we lose a few percentage points of safety, we gain a surprisingly simple and parsimonious representation of a protocol that may otherwise require some fairly complex data structures.

Also, note that this style of Casper CBC is fundamentally very similar to how Avalanche works, where each node gets consensus by polling a committee of other nodes. The main difference here is that the committee is selected by the protocol, slashing conditions enforce compliance, and GHOST is used as a fork choice rule to extend N-ary consensus to chains, achieving economic security efficiently. This suggests that there may be a more general framework that can include both Casper CBC and Avalanche effectively.

Further work:

- Casper CBC's fault tolerance can be improved up to \frac{1}{3} - \epsilon

by increasing the number of rounds that you wait. Can we use a similar technique to increase fault tolerance above 20%?

- Is there some way to make sharding happen naturally in this setup? In general, the goal would be to replace the chain with some kind of DAG, where each block is aware of the parent in its own shard and, say, the tenth most recent and older blocks in other shards, and expect validators to only full-validate blocks that have not yet been validated by a sufficiently large validator sample.