This post requires understanding the general approach of publishing a Merkle root of erasure-coded data and having clients randomly sample Merkle branches to sample-check availability of the data, described here and here.

Describing the status quo

Currently, the de facto philosophy of eth2 is that, much like for non-sharded chains, the canonical chain is defined as the fully valid and available

chain that scores highest on the fork choice rule. In the non-sharded case, the "fully valid and available" criterion is trivial: full nodes just check it automatically. In the sharded case, however, nodes cannot check validity and availability directly, because "fully valid and available" includes shard chain data (ie. crosslinks) and there is too much shard chain data to check. Instead, nodes rely on a series of heuristics:

- If 2/3 of a randomly sampled shard committee does not support a particular crosslink, then it cannot be included in the first place
- If a node sees a fraud proof showing how a given crosslink is invalid, it rejects it
- A node can run a data availability sampling check (see https://arxiv.org/abs/1809.09044) on any crosslink before accepting its legitimacy

These heuristics are imperfect. Hence, while in an abstract definitional sense the canonical chain is defined

as being available and valid, it's possible that some nodes will wrongly believe that some block B

is the head, because they do not realize that some crosslink in B

's history points to unavailable or invalid data. However, we expect the combination of heuristics to be highly robust in practice.

Note that fraud proofs are a retroactive

heuristic: you may believe that some block B

is valid, but then you see a fraud proof and after that point block B

becomes invalid. In general, fraud proofs should be broadcasted quickly. However, in the worst worst case, where multiple security assumptions fail at the same time (there is an attacking majority in a crosslink AND there is a network latency of >6 minutes that affects fraud proofs but somehow NOT legitimate blocks), there's a risk that an invalid block will get "finalized", and then become "de-finalized" after the fact.

In this extreme case, because of Casper FFG's rules, the validators that voted on the finalized block will not be able to vote on the new "correct" chain, and so they will lose a large portion of their balances until the inactivity leak leads to a minority finalizing the new correct chain. Arguably, this is okay; if a disaster happens, then whoever was responsible for the mistake should be penalized, even if their mistake was simply not trying hard enough to stay connected to fraud proof providers, or not running heuristics that stop making finalizing votes if network latency is too high. But there is also a "friendlier" approach.

Layer 1 optimistic rollup

Consider a chain architecture that works as follows. The canonical chain is defined as being the fully available and valid beacon chain pointing to fully available shard blocks

that scores highest on the fork choice rule. That is, there are availability and validity requirements on the beacon chain, but there are only availability requirements on the data pointed to by the crosslinks; as long as the crosslink root actually is a Merkle root of data that is all available and of the right length, including that crosslink does not compromise a chain's validity.

We use a STARKed merkle root as a data availability proof mechanism. When a block is published, we can use the crosslink roots included in the block to compute a combined root of all crosslinks included in that block. We can then add a reward for someone to submit an erasure-coded extension of this Merkle root along with a STARK proving that the extended Merkle root is correct.

This proof may take minutes to produce; hence we allow and expect validators to at first build on top of some block B

"optimistically" without verifying data availability, and only expect and require data availability validation when validators make blocks that treat B

as justified

in the Casper FFG vote, as it's treating an invalid block as justified that is potentially very risky.

If a shard chain block with available data but “bad” contents (eg. wrong post-state root) gets included in a beacon chain, then this does not invalidate the chain. Instead, we add a mechanism where a fraud proof can be included in the beacon chain later, and inside the chain

this rewinds that shard chain block and all shard chain blocks that depend on it. Validator-related state does not get rewound; shard state roots do. A chain can rewind a maximum of 2 days; after 2 days, a bad state root included in a chain and not removed really does make that chain invalid. This allows validators to deposit (move shard state -> beacon chain state) with only a 2 day waiting period.

In this construction, the heuristics needed to “fully validate” the chain are much lighter:

- Fully validating the beacon chain
- Data availability check on crosslinks via random sampling
- Watching out for fraud proofs with a 2 day synchrony assumption

Note that the STARKed data availability root is required for this construction to work.Coded Merkle trees or the original 2D scheme are not acceptable, as in the case where an erasure coded root is fraudulent, different validators may durably disagree regarding whether or not it is valid. If an erasure coded root is correctly constructed, then any one client successfully validating availability implies that >\frac{1}{2}

of the data is available, which implies that the rest of the data can be recovered and so every other client’s availability checks will soon pass. But if an erasure coded root is incorrectly constructed, then some validators may see the errors and others may not see the errors, leading to a chain split. To avoid these issues, we need a STARKed data availability root to remove the possibility of incorrectly constructed data availability roots.