TL;DR: a simple, flat type-length-value format for storing finalized block and consensus state data, for interop, archival and out-of-band dissemination - full spec here.

Intro

Today, consensus clients store and offer via P2P the entire beacon chain block history so as to allow syncing from genesis - to date this is in the order of 40-50GB of data, but with the merge, the growth of this data set will accelerate significantly with the inclusion of execution transactions in consensus blocks.

An alternative way of syncing is to start from an arbitrary point in time using a known trusted checkpoint - due to weak subjectivity, the security model remains the same (ie in both from-genesis and checkpoint sync, you need to know one valid block hash from within the weak subjectivity period to be safe against weak subjectivity attacks).

Regardless of sync method, clients conforming to the specmust make block data available for MIN_EPOCHS_FOR_BLOCK_REQUESTS

epochs (~5 months) but de facto clients store all blocks from genesis - in part this is to support from-genesis sync, but also because there exists no standardized and interoperable way to disseminate the data outside of the p2p specification and there is general concern that the data will be lost or difficult to recover should clients simply start dropping it with a replacement mechanism.

Enter ERA files - a simple, flat storage format for out-of-band interchange of historical block and consensus data.

Format

- Specification

The basic structure of an era file is a list of type-length-value records. Each era file starts with a version marker, followed by a number of entries: blocks, a beacon state and finally some indices that allow lookup of block data based on slot number. Blocks and states are stored as they typically appear in the p2p protocol of consensus clients: each record is encoded with SSZ and compressed with snappy.

Each era covers 8192 blocks (or roughly 27 hours), and includes a state snapshot of the beacon chain state at the end of the era.

The state snapshot serves two purposes: to validate the blocks contained in the era file (for which validator public keys as well as block roots are needed), as well as serve as a starting point to recreate history from there onward.

Finally, era files contain a slot-to-offset index allowing linear lookup of block data based on slot number.

Features

Era files are formed from finalized data: they are immutable and verifiable using a recent-enough state. Without making any additional security assumptions, they can be stored out-of-band and transferred using existing internet infrastructure such as plain cache-friendly HTTP, bittorrent etc.

Any era file can be verified using the historical_roots

field of any beacon state more recent than the era file, all the way back to genesis.

Because they contain a beacon state, they can also serve as starting point for recreating detailed historical data for each slot past the given era - for example to examine balances and states of validators at a particular point in time.

Clients wishing to use era files for syncing purposes can download them using a simple http client - with predictable naming, this can be used as a fast and efficient alternative to backfilling via P2P - in this sense, they replace and enhance checkpoint sync.

Finally, with an interoperable format for archival, clients are free to start dropping historical data altogether, because the needs of users requiring full archival remain met (albeit via a different mechanism than today).

Who will store the data?

Currently, the P2P network stores / supports the from-genesis history of blocks, but it is difficult to motivate this from a node perspective - it is not needed for computing consensus but costs significant bandwidth and storage.

Assuming that the historical data is of interest to the community, it is expected that existing large operators and data providers will fill the space of providing long-term storage - a single interested entity is enough to ensure that the data remains available - the data can also be backed up to decentralized storage (though in order to do so, an additional mapping mechanism is needed that allows the data to be found).

The current beacon chain history is available in era forma (additional sources will be added soon

):

- https://mainnet.era.nimbus.team/ - Mainnet consensus data from genesis

How will the data be made available?

Similar to the beacon API, providers of infrastructure can provide HTTP endpoints serving era files - a Beacon API PR is being considered to make era file access part of the “standard” api, likely via an “optional” extension.

Implementations

The era file format is fully documented and has as of writing, reference implementations inPython and Nim.

Nimbus supports using era files in lieu of its own database to serve p2p and API requests, allowing a single era file store to be shared among multiple Nimbus beacon nodes and avoiding the need to backfill via P2P after a checkpoint sync.

What’s missing?

With the merge, era files become complete stores of all data necessary to recreate the ethereum state history - however, pre-merge blocks are not covered by the current proposal.

Post-merge, the pre-merge history will be considered immutable / finalized as well and it is expected that a separate standard for the pre-merge data will be written so as to capture it, likely using the type mechanism of the ERA files.

What’s next?

This post serves to introduce the format and open it up for discussion - the next step for this proposal is to make it a standard (via EIP / consensus spec) - given how after the merge, blocks also contain transaction history, the format is of interest to both consensus and execution clients, allowing both to use the same archival storage format for raw block data.

Why now?

As of the merge hard fork, all consensus clients implement checkpoint sync allowing them to stop relying on full sync as the only mechanism to get going.

At the same time, the storage problem is becoming significant - it is widely recognized in the community that all nodes storing full block history is not a sustainable solution.

Delegating some of the storage work to participants better suited to handle the problem of long-term storage allows lowering the cost of running a node and those lowers the bar for running a consensus node in general.

Related work

- EIP-4444 - Bound Historical Data in Execution Clients
- [getPayloadBodiesByRangeV1

](https://github.com/ethereum/execution-apis/pull/218) - sharing block execution data between execution client and beacon node