authors: Seve, Eray, Tatu, and Lichu from Semiotic Labs, core dev of The Graph

Semiotic Labs is a core developer of The Graph protocol. The authors of this post are part of the team dedicated to developing and implementing mechanisms to verify that the data served by The Graph is derived from data committed on each indexed blockchain. The work presented in this post is a byproduct of our ongoing efforts.

Introduction

The Graph is a decentralized blockchain data indexing and querying protocol. Indexers are the heart of The Graph, indexing the blockchain's history for specific events and making the data available for Consumer querying. Indexers need access to historical blockchain data to do their job. In the future, The Graph's Indexers will extract data from the blockchain using a specially instrumented blockchain full node as part of the Firehose data extraction process. Firehose is an open-source tool created by StreamingFast, a core developer in The Graph. Firehose offers a flat file-based and streaming-first approach for processing blockchain data. Semiotic Labs is another core dev team in The Graph, focusing on research and development in the areas of artificial intelligence and cryptography.

Currently, blockchain data is extracted from the Firehose-enabled full node by performing a full archive sync, which requires peers in the blockchain's p2p network to provide historical data for the entire blockchain history. In a future where EIP-4444 is implemented and full nodes stop providing historical data over one year, The Graph's Indexers will need to change their blockchain data extraction process to continue providing the required data.

This post describes how The Graph's Indexers use Firehose to extract and store blockchain data in Google protobuf serialized object storage, called flat files.

When Firehose is synced from Genesis, these flat files store all the required information to preserve the complete history of data stored in the blockchain (specifically block bodies and receipts). We propose that these flat files can be used as an archival data format, similar to era and era1 files, and show how the data stored in flat files can be verified to belong to the canonical blockchain. For The Graph, once EIP-4444 is in effect, Indexers can share flat files to obtain the required historical data for indexing and querying. For Ethereum, The Graph's network of incentivized Indexers collectively ensures that historical data is preserved and available, providing a verifiable, decentralized, and independent solution for EIP-4444.

Interestingly, our initial goal was not explicitly designing a solution for EIP-4444. The tools and technology discussed here were developed to enhance the performance and verifiability of The Graph protocol. However, during this process, we unexpectedly discovered a solution for EIP-4444 "for free". Our vision is for this solution to coexist with other EIP-4444 solutions, much like the multiple implementations of Ethereum execution clients, without causing a heterogeneous distribution of block data.

Overview of The Graph Firehose Pipeline

The Graph is a dynamic protocol that is continuously evolving. To fulfill the need for quick access to blockchain data, a new technology called Firehose and Substreams was developed. Additionally, a new indexing pipeline using this technology was created specifically for Indexers.

Firehose is the first processing step in this pipeline, extracting data from the blockchain and storing it in flat files (more on this in the next section). Firehose then exposes a gRPC interface for streaming the blocks stored in the flat files. In The Graph, this stream can then be consumed by several endpoints. One example is a Substreams service, which filters for specific events of interest and performs any specified transformations on the filtered data. The output is then routed to an endpoint sink and made available to Consumers.

Figure 1 provides a visual representation of the Firehose and Substreams pipeline as described, demonstrating how data is extracted from the blockchain, processed, and made available to Consumers through various endpoints.

[

Figure 1: Firehose + Substreams Pipeline

3876×1538 81.6 KB

](https://ethresear.ch/uploads/default/original/2X/c/c910008af6edb0a6ff391ee3136ebcf8217236de.png)

Figure 1: Firehose + Substreams Pipeline

Regarding EIP-4444, we are particularly interested in the flat files that store the extracted blockchain data. We observe that flat files store all of the data necessary for preserving the history of the indexed blockchain, that the data can be verified to be included in the canonical history of the blockchain using straightforward techniques, and that The Graph, as a decentralized blockchain data indexing and querying protocol, naturally incentivizes ensuring that the entire history of the blockchain is available.

Flat Files as Shareable Ethereum Archives

Firehose extracts data from the blockchain by using aspecially instrumented full-node; Ethereum Firehose uses an instrumented version of the official Geth implementation *(*note that Firehose supports many chains, including non-EVM chains, and the full node used depends on the chain.). In particular, Firehose reads from the trace API of the full node during a full archive sync. Refer to Figure 2 for a detailed depiction of these steps.

[

Figure 2: Firehose data extraction

2420×1399 230 KB

](https://ethresear.ch/uploads/default/original/2X/3/3e352b69d66b55877047307ff7bb9873253cf49a.png)

Figure 2: Firehose data extraction

The output of the trace API is what gets stored in flat files. It's worth noting that the trace API can be instrumented to provide a varying degree of introspection into the EVM execution, e.g., as a callTracer

, which tracks call frames executed during a transaction (this includes the top-level transactions calls as well as calls internal to the transaction), or the trace API can be configured as struct/opcode logger

which would emit the opcode and execution context for every step of transaction execution. To demonstrate the viability of Firehose flat files as a historical archive for Ethereum, we only care about preserving the top-level transaction calls and receipts, as this is the data that is committed to and is verifiable in the blockchain. In this context, Firehose stores all transaction data (block bodies) and receipts associated with transaction execution in the blockchain.

While Firehose flat files were not specifically designed to export historical Ethereum archive data, they store all the necessary historical data. These files are shareable and, as we will see, verifiable, which makes them a viable candidate for an archive format.

Verifiable Flat Files

Flat files store all top-level transaction data (block bodies) and receipts from transaction execution. For example, a (JSON) decoded flat file using Semiotic's flat-files-decoder for block 15,000,000 is shown in Figure 3. We can see the stored block header information and transaction traces for top-level transactions, e.g., for transaction 0x7ecdee…e3222 below. Ethereum blockchain data extracted using Firehose is stored in flat files according to this schema.

[

Figure 3: Decoded Blockchain Data from Flat File for Block 15,000,000

1920×1202 210 KB

](https://ethresear.ch/uploads/default/original/2X/1/191550b9aa546e099bf457c62184d8638d2e3d7f.jpeg)

Figure 3: Decoded Blockchain Data from Flat File for Block 15,000,000

Flat files store all the information necessary to reconstruct the transaction and receipt tries. These tries are committed to every block in the canonical blockchain. To verify that this data belongs to the canonical blockchain, all that is needed is to confirm that the roots of these constructed tries match the roots stored in a canonical block header.

Post-merge, using information from the Consensus Layer. For example, Ethereum provides the sync-committee subprotocol, which can be used to validate block headers from the canonical blockchain. To verify that the data stored in a post-merge flat file is correct, all that is needed is to reconstruct the receipt and transaction tries for the block in the flat file and then compare these roots to a valid sync committee signed block header. Valid block headers can be obtained from a consensus light client.

For an example of how this can be done, see our proof-of-concept codehere. According to our benchmarks, a single block's data extraction and trie construction takes about 5ms on an M1 Mac.

Pre-merge requires a little more work. Fortunately, we are not alone. The Ethereum community has been developing solutions for this problem. Specifically, The Portal network uses Header Accumulators to verify that a block header has been included in the canonical chain. Importantly, these accumulators are agnostic to the storage format of the archive data, so we can align ourselves with this approach and compute the same Header Accumulator values from the data stored in flat files. When an Indexer shares a flat file containing pre-merge blocks, they must also provide an inclusion proof for the header containing the receipt and transaction trie roots.

Flat files

Era files

The archival data format and verification mechanisms presented for The Graph's Firehose flat files demonstrate compatibility with established data formatting approaches, such as the era/era1 data format described here. A key difference what blockchain data gets stored. Specifically, The era file format specifies including a snapshot of the Beacon chain state with every era file, with the goal of bootstrapping access to state data for processing the chain starting from that file. However, Firehose flat files intentionally do not contain any Beacon chain state data, as the purpose is solely focused on archiving historical transactional data and receipts from the execution layer. Since Firehose flat files are not aimed at bootstrapping an actual Ethereum node, preserving the state is outside the scope. The Graph products focus specifically on indexing event data for querying rather than stateful execution of an Ethereum node.

Table 1 compares the structural nuances and use cases of the flat files format and the era1/era format.

Attribute

Flat files

Era1

Serialization format

Binary

Binary

Schema-based

Yes

No

Type support

Strings, integers, floats, dates, and more

Not specified

Encoding scheme

Wire formats (Varint, fixed64, length-delimited, and fixed32)

SSZ, Varint, snappy

Unknown data

Requires to understand the object structure to decode it

Can skip, using TLV

Each file

Optional, 1 block or 100 blocks

8192 blocks (or roughly 27 hours)

Verifiability

Yes, using Semiotic's tool

Yes, using recent-enough state.

Beacon chain state data

No, The Graph products focus on contracts, traces, and events.

Yes, includes a snapshot of the Beacon Chain state

Table 1: Flat files and Era1 attributions

Incentivized Flat File Retrievability

As we have seen, new Indexers in The Graph running Firehose must perform a full archive sync of a full node to extract the required historical blockchain data for indexing. This process introduces two challenges, and we will explain how having shareable, verifiable flat files paired with The Graph's incentivized marketplace solves each of these problems.

Challenge 1:

Performing a full archive sync can be time-consuming and depends on factors such as the file system and Indexer hardware. At Semiotic Labs, syncing a Firehose node (Geth + Firehose) took approximately two months.

Solution 1:

In cases where the flat files become corrupted during the syncing process (as happened to us after two months), it is possible to bootstrap the Firehose sync by downloading flat files from other Indexers and directing Firehose to use them. Using the tools described above, the Indexer can verify that the flat files downloaded from peers belong to the canonical blockchain. Rather than performing a full archive sync, the Indexer will seed Firehose with flat files downloaded from peers and verified against the canonical blockchain. The bottleneck becomes download speed, e.g., days to download ~2TB for the entire archive from Genesis rather than months for a full archive sync.

Challenge 2:

Once EIP-4444 is in effect, mechanisms are needed to preserve history.

Solution 2:

We note that flat files are both shareable and verifiable, so Indexers can establish an incentivized marketplace for sharing flat files. Consumers of The Graph specify the events and the desired range of history. This history range is not known in advance. An Indexer who wishes to be future-proof will ensure they have data spanning the entire blockchain history. There is currently an effort within The Graph ecosystem to build this marketplace as a data service, a "flat file sharing data service" enabled by The Graph.

Regarding EIP-4444, we are answering two questions, "Who will store the data?" and "How will the data be made available?". The Graph's Indexers will store historical data in the form of flat files, and the data will be made available through subgraphs/substreams in The Graph protocol and a flat file-sharing data service currently in development.

Conclusion

The flat file format generated by The Graph's Firehose provides a decentralized and verifiable solution for preserving Ethereum's historical data after EIP-4444 is implemented. We have demonstrated how flat files contain all necessary blockchain data and can be verified against the canonical chain.

We are working to mature our open-source flat file verification code, which can be used to verify downloaded flat files. We envision Indexers in The Graph protocol using this code to distribute and monetize flat files in a data service marketplace instead of syncing a full node using Ethereum's p2p network.

The Graph network naturally enables and incentivizes a unique EIP-4444 solution. In particular, the flat files generated and used by Indexers can also be used as a shareable and verifiable archival format. We have demonstrated how flat files contain all necessary blockchain data and can be verified and provided proof-of-concept code. Our proposed approach aligns with existing efforts like Portal Network's header accumulators and the era/era1 file format. It provides a complementary solution focused on the execution layer data that can coexist alongside other EIP-4444 solutions without causing a heterogeneous distribution of block data. We are working to mature our open-source verification tools so Indexers can easily validate downloaded flat files. Soon, The Graph will launch a flat p2p file-sharing data service, opening up alternative methods to access historical data.

This decentralized, verifiable flat file solution requires no protocol-level changes and can help protocols like The Graph continue functioning smoothly after EIP-4444. We offer it as a viable approach to preserving Ethereum's history to benefit the entire ecosystem.

Call to action

- Feedback loop

With this post, we hope to share how the technology we are developing on The Graph protocol can be used to solve problems from the broader community. We are looking for feedback, so if you have any questions, recommendations, etc., please get in touch with us.

- Distribution and accessibility of flat files

We feel confident about the claims that Firehose flat files, generated by syncing a Firehose-enabled node from Genesis,

contain all the necessary information to preserve the entire blockchain history and that the data in these flat files can be verified using the mechanisms above. One question is how do we manage the distribution of flat files and ensure that the historical data is easily retrievable? We presented two arguments for incentivizing distribution and retrievability: Indexers will maintain a complete history to maximize their ability to provide open-ended support for future subgraphs/substreams in The Graph protocol, and the future flat file sharing data service will ensure a marketplace for incentivizing Indexers to sell this data.

- Support Firehose as a feature for full nodes

An ongoing effort is to integrate Firehose as a feature in full nodes.Firehose provides a novel method for interacting with blockchain data, as evidenced by its use in The Graph to enable performant indexing. As seen in this post, the flat files output from Firehose can also serve as a shareable and verifiable storage format for historical blockchain data. This further demonstrates that enabling Firehose as a feature in full nodes unlocks new utility dimensions for full node operators. It's also worth noting that Firehose supports many chains, and the list is growing, providing a common interface across chains.

Appendix and Related Works

EIP-4444: Bound Historical Data in Execution Clients

Example flat files

Example era files

StreamingFast Firehose data storage

Era files specification

Log In ERA - archival files for block and consensus data

The Graph Network In Depth

https://ethereum-magicians.org/t/eip-7542-eth-69-available-blocks-extended-protocol/

https://substreams.streamingfast.io/