Motivation:

TL;DR:

Light clients struggle to efficiently access and validate data on the mainnet due to challenges in obtaining concise witness proofs. Verkle trees help lightweight clients transition between blocks but can't prove new state accuracy. Stateless clients lack state data for actions beyond transitioning. The Portal Network doesn't fully address these issues. Our solution adds entities to the stateless verifier LC on the Portal Network with a cache to store important state fragments. We distribute the latest state using zero-knowledge proofs and propose a chase mechanism for efficient data retrieval. This addresses challenges in accessing state data for tasks like gas estimation and enhances lightweight client efficiency.

Project Description:

Light clients struggle to efficiently access and validate data. Currently, light clients, which rely on simplified verification, face challenges in accessing and validating the mainnet state due to the absence of concise witness proofs. These clients need to confirm blocks without having access to the full state.

Verkle trees allow very lightweight clients to consume proofs from other networks to transition from the last block to the new block. However, they cannot prove the accuracy of the new state root. If a stateless client discards all its stored information, it can still confirm the accuracy of new state roots. By doing so, a stateless client can still send transactions but cannot calculate gas estimates, perform ETH calls, or read Ethereum's state since it no longer maintains any state data. The client is limited to actions that involve transitioning from one state to the next state root, without any specific state-related functions.

This is where the Portal Network comes into play. While it allows the reading of random state data, it doesn't fully mitigate the core issue. The underlying challenge persists—efficiently accessing state data remains crucial for various tasks, including gas estimation. Additionally, Verkle trees, despite their benefits, don't inherently solve problems like federated access to the state.

To bridge this gap, an innovative solution comes in the form of the Portal Network introducing a stateless verifier LC (Lightweight Client) with a partial state caching mechanism to enhance the efficiency of accessing specific segments of the state. It achieves this by storing frequently accessed or important state fragments in a cache, enabling clients to retrieve them more quickly than repeatedly traversing Verkle trees.

Our proposal of partial state caching complements has following value propositions:

- Improved Retrieval for Stateless Clients:

Stateless clients lack the ability to store the full state and rely on external means to access data. By using partial state caching, we offer an efficient method for these clients to access vital state fragments, reducing their reliance on complex Verkle tree processes.

- Less Data Transfer and Computation:

Stateless clients struggle with data transfer and computation. Partial state caching lets them access pre-cached data, lessening the need for extensive data transfers and computational work, in line with the efficiency objectives of stateless clients.

-Decentralized, Trustless Verification:

Stateless clients aim for trustless Ethereum network interaction. Through partial state caching, clients can independently verify cached state fragment validity using zk proofs, preserving trustlessness by eliminating dependence on a central source.

-Swift Data Retrieval: Cached state fragments are readily available, bypassing the need to rebuild or navigate the Verkle tree for each request. This rapid access to cached data results in quicker retrieval times compared to direct tree fetching, especially for frequently needed data.

- Reduced Network Latency:

Cached fragments can be fetched locally, reducing the reliance on multiple network interactions for Verkle tree traversal. This minimizes network delay and enhances responsiveness.

- Efficient Resource Use:

Cached fragments reduce computational load during Verkle tree traversals, particularly for complex state structures. This optimizes computing resource utilization.

- Consistency and Validity:

The partial state caching mechanism ensures consensus-validated data, preventing caching of compromised or invalid data. This boosts integrity and data retrieval reliability.

-Optimized State Access:

Partial state caching can prioritize frequently accessed state fragments, catering to stateless clients' needs for specific data subsets. This speeds up necessary information access, elevating overall efficiency.

- Improved Security and Reliability:

Stateless clients face security risks with third-party state data. Incorporating cryptographic proofs and cached state fragments empowers clients to autonomously verify data integrity, boosting security and reliability in Ethereum network interactions.

Ps. I've added an issue in my GitHub to propose an initial draft of the Cashe mechanism design.