Trustless access to Ethereum State

Inputs and reviews by Viktor Tron and Daniel Nagy better shaped this text.

ABSTRACT

This proposal addresses the problems related to accessing the current or historical state of blockchain data of the Ethereum network. The proposal offers a principled way to tackle the blockchain data availability issue in a way that ensure data access is verifiable, efficient, and scalable.

PROBLEM

When DApps need to access blockchain state, they call the API of an Ethereum client that is meant to provide the current state correctly. This condition is most directly satisfied if the DApp user runs their own Ethereum full node that synchronises with its peers on the same network to build and verify the state.

Very early on this proved unrealistic in practice: given the resource-intensive nature of running a full node, they struggled to effectively manage in real-time two essential tasks - actively participating in the consensus process and handling data uploads and queries.

Two distinct methods emerged to tackle this: light clients and centralised service providers.

Light clients are not meant to maintain and store the full state, but only synchronise the header chain, while they originally used the Ethereum wire protocol to retrieve state data with merkle proofs verifiable against the header chain. Over time, they have evolved to use a specialized subprotocol known as LES (Light Ethereum Subprotocol). However, LES still faces challenges related to efficiency and lacks robust incentive mechanisms to encourage full nodes to support light client services.

Another strategy is to use centralised service providers to offer remote endpoints for DApps to use. While this proved to work very well in practice, it reintroduced concerns of resilience, permission, and privacy which Ethereum itself was supposed to remedy in the first place.

Centralised provision exposes the network to single point of failure scenarios and also raises privacy concerns due to the ability to track users' query origins. Centralised control, on the other hand, makes the system susceptible to political pressure and regulations leading to censorship. On top of this, the lack of verifiable integrity proofs makes this solution not so different from using the API of a public block explorer running on centralised servers.

The current situation is elucidated by the following documented mishaps:

- State data integrity is not preserved,

because Block explorers digest state data and prune some of the collected data, lacking accuracy: "EtherScan misses nearly 15% of what are called 'appearances' "

(Source: Trueblocks - How Accurate is EtherScan?).

- Users' privacy is not protected

and information leaks from the originator of queries: "Infura will collect your IP address and your Ethereum wallet address when you send a transaction."

(Source: Unmasking Metamask — Is Web3 Really Decentralized And Private?).

- Centralised providers may block users or entire communities for purely political reasons

(Source: Infura Cuts Off Users to Separatist Areas in Ukraine, Accidentally Blocks Venezuela) or other reasons (Source: Infura, Alchemy Block Tornado Cash Following Treasury Ban).

Another burning issue is access to historical state data. Although planned since Ethereum's launch, archival nodes are non-existent to this day due to both technical challenges and lack of incentives.

Addressing this situation will strongly reinforce Ethereum's position as the

leading decentralized platform and advance the broader vision of a user-centric, decentralized Web3 infrastructure that promotes privacy-conscious and permissionless access to information.

HOW SWARM COULD HELP

Swarm network (web, wp) presents a practical and realistic solution to the full-node latency problem: by establishing a dense network of actively participating nodes that will ensure the storage and dissemination of these crucial pieces of data during the bootstrapping phase. The dataset exposed by Swarm is designed to be completely reproducible, ensuring the integrity and reliability of the data, allowing for verification and validation of its contents.

This approach draws inspiration from the peer-to-peer principles of torrenting, through queries that traverse the trie structure, nodes proactively replenish their caches along the path, therefore preserving and serving the most frequently accessed data. Noticeably, Swarm's caching mechanism would greatly accelerate the speed with which you can get the data, because of the bandwidth incentives present in the network.

If a node chooses to operate without relying on long-term storage from the network, this approach becomes feasible as long as it is active and consistently replenished. However, it's important to note that while it can be considered altruistic to some extent, it will inevitably fill the node with data over time. Storage incentives mechanisms (Postage stamps) address the long-term compensation for data storage, ensuring that those who store data for others are duly rewarded.

If light clients could request such data from Swarm, which, in turn would either serve it up from cache or, in the absence of a cached copy, request it from different Ethereum nodes (not necessarily light servers), it would considerably alleviate the load on light servers and thus improve the overall user experience.

More complex queries

Furthermore, a significant portion of DApps, developers, and users aspire to extract state information that transcends the limitations of ETH APIs. They seek to access blockchain data in a manner that reminds of flexible database queries. Functionalities such as filtering for 'more transactions exceeding 30 ETH'

or identifying 'stablecoin owners'

, or even seemingly straightforward tasks like 'transactions by sender'

often necessitate third-party platforms like Etherscan or The Graph.

As a result, there is a growing need for a database bootstrapping solution, one that assures data integrity, verifiability, and reliability. This becomes increasingly relevant for the forthcoming years, as the demand for diverse and tailored data access intensifies.

Notably, Swarm will allow reindexing of data that will allow alternative queries to be responded with verifiable resolution of the query response.

Through the integration of Swarm inclusion proofs, BMT proofs, feeds, and integrity protection structures, Swarm possesses the means to overcome these challenges and validate query responses in a trustless manner. Just as there exists a concept of 'proof of correctness,' Swarm is poised to employ a similar methodology for establishing 'proofs of indexation.'

Protocol outline with Swarm

The implementation of this approach is straightforward. Ethereum's state consists of many small binary blobs addressed by their Keccak hashes. The vast majority of these blobs are less than 4 kilobytes in size. Swarm has been specifically engineered to retrieve and cache binary blobs (called "chunks") that are at most 4 kilobytes in size. While Swarm's content addressing is different from Ethereum's, it is relatively easy to define an extension to Swarm that would allow retrieving specific chunk payload by its Keccak hash.

The Swarm network can possibly be used by Ethereum light clients as a state cache and load balancer.

1. When requesting a blob of state data, the light client first turns to Swarm.
2. The Swarm request either gets served from cache or gets routed to the neighborhood that is closest in XOR metric (as per Kademlia routing) to the content address of the blob.
3. The node in the neighborhood that received the request then requests the given blob from the Ethereum node to which it is connected. In case the blob is found and is no greater than 4 kilobytes, it is cached and served as a response.
4. If the node is not found or the blob exceeds 4kB in size, a response to this effect is routed back to the requesting light client, which requests the blob directly from a full client, if applicable.

The caching and serving of state data is incentivised by the bandwidth incentives already present in Swarm, while storage incentives ensure data availability.

This architecture will:

- Eliminate the light server verifiability problem and re-decentralise read access to ETH state for DApps.

- Eliminate the problem of full/archival nodes, allowing any storage node to serve historical requests.
- Unburden ETH miners from syncing traffic with potential improvement on transaction throughput.
- Combined with Swarm's decentralised database services, serve as a decentralised (backend for a) chain explorer.

In essence, this will allow light clients, browsers, dapps, or developers to query Ethereum's state more efficiently and trust the result about data structures and indexes.

SUPPORT TO ALTERNATIVE STORAGE SOLUTIONS

To be realistic, this proposal assumes some integration with execution clients, but always in a way to serve others, championing modularity, while assuming the minimal impact into current clients' roadmaps. For instance, the Go client can be readily adjusted through a minor modification of the ETH API and state trie-related libraries. A standard / generic hook might be designed by the client team so that the architecture is open and future facing to be extended to IPFS, Arweave, and others.

For alternative storage solutions, this adaptation involves the inclusion of API client calls in the decentralised storage network on top of the existing on-disk storage API calls.

Technical details are available upon request. We actively encourage feedback and comments from the community, and we remain committed to providing comprehensive responses.