Week 9 Research track - History Expiry

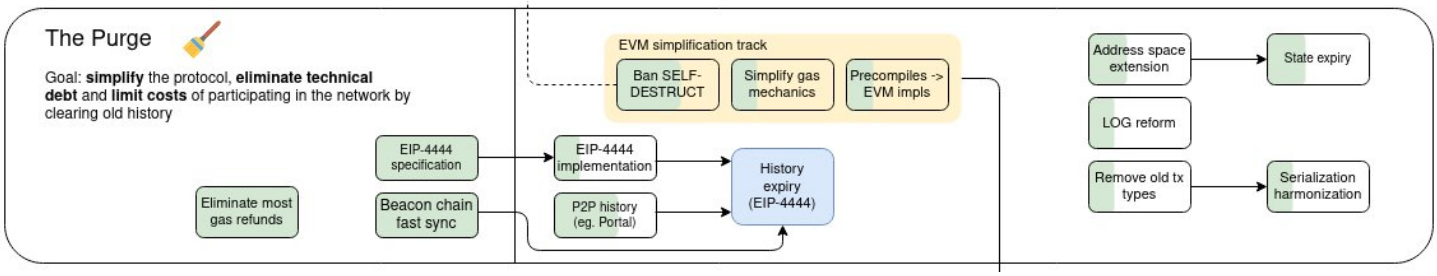
Guest speaker

Piper Merriam, Researcher at Ethereum Foundation, Founder of the Ethereum Protocol Fellowship program

Summary notes

- Edited by Chloe Zhu
- Online version: https://ab9jvcjkej.feishu.cn/docx/GXLCdUFL9oRpEIxTYJvcdLUMn7w

The Purge



Block execution bottleneck

- The problem
  - The Ethereum state currently is stored in Merkle Patricia Tree (MPT). To do block execution, clients need to access data through this large MPT with billions of data.
  - The original access pattern of stepping down through 7-10 layers of MPT is too heavy for clients.
- Early optimization by the client team
  - Flat database structure is designed to speed up the accessing of data from the state tree. But the problem of disk IO still persists for clients.

Statelessness

- The problem
  - The major problem remain to be the MPT itself as it has inefficient and attackable proofs
- Solution

- Verkle tree (VKT) provides the possibility for Ethereum to be statelessness, as VKT has more succinct proof shape comapred to MPT

History expiry

- The problem
  - Currently Etheruem has c.19 million blocks, which contain loads of data. Among the data, the early historical data is of little use for clients, but takes a lot of disk storage.
- Solution: EIP-4444
  - The main goal is for clients to drop the deep history of all the headers and blocks
  - 3 specific approaches
    - Torrent-based solution (used by Erigon)
    - Era1 file format (an archive format to service EL history premerge)
      - More details: https://github.com/ethereum/go-ethereum/pull/26621
    - Portal network: the P2P decentralized network for storing & retrieve data

Opcode SELFDESTRUCT

- The problem with SELFDESTRUCT
  - The function of SELFDESTRUCT in its current & historical form is to clear the storage data for a contract. In the MPT, the only thing that represents the storage of the contract is the storage root, and the contract code is only referenced as the codehash. **The code itself is not stored anywhere in the state tree.**
  - However, the EVM needs to read the contract code for implementation, which means the client has to either deduplicate contract code, or do reference counting, or leak contract code. If the client stores the contract code in its key value db under the code hash, and multiple accounts deploy the same code, there will be only one entry in the db.
  - The problem is when the account SELDESTRUCT & clears its data, the contract code cannot be cleared correspondently. That causes a long running accumulation of code storage.
- Solution
  - Depreciate SELFDESTRUCT
  - Transition to Verkle, so that the contract code can directly live inside the main state tree.

Precompiles -> EVM implementations

- The problem
  - Keccak and SHA 256 of precompiles make zkEVM really hard to implement

- Solution
  - Get rid of precompiled contracts in favor of direct EVM implementation through EOF (EVM Object Format)

State expiry

- The goal of state expiry is to let deep history of Ethereum (older than c. 6-12 months) go into hibernation.
- How state expiry works
  - 2 principles
    - Only the most recent tree can be modified
    - Full nodes (incl. Block proposers) are expected to only hold the most recent trees (within c.6-12 months), so only objects in the most recent trees can be read without a witness
  - Hybrid state regime
    - Consensus nodes need to store state that was accessed or modified recently
    - But can use the witness-base stateless client approach to verify older state

Address Space Expansion

- A slightly different addressing scheme is needed for state expiry to work. Addresses need to be extended from 20 to 32 bytes, and the new address format includes a concept of “address periods” (formerly called “address spaces”). This allows new contracts to fill new storage slots without needing to provide witnesses.
- The major challenge to implement address space expansion is that currently lots of clients' libraries strictly deal with 20-byte addresses, so 32-byte addresses would cause great amount of work on the client side.
- Relevant blog from Vitalik: https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposal

LOG reform

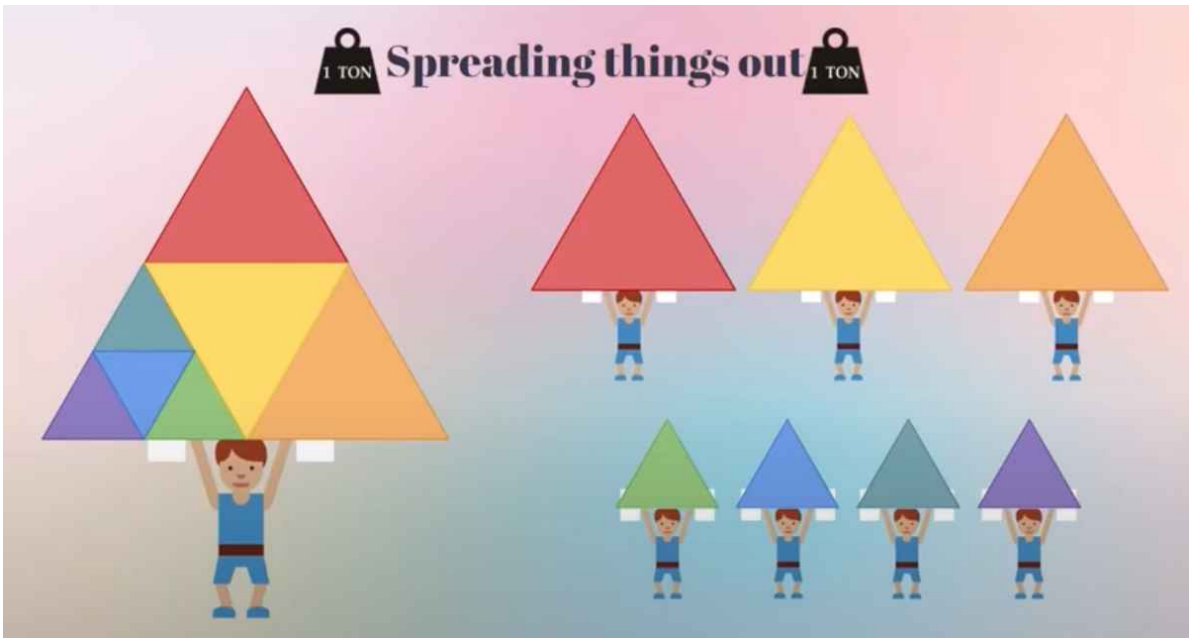
- Logs were originally introduced to give apps a way to record info about onchain events, which dapps would be able to easily query. In practice, the mechanism is far too slow. And almost all dapps that access history end up doing so not through RPC calls to an Ethereum node, but through centralized extra-protocol services.
- The solution is to simplify the LOG opcode so that all it does is create a value that gets hashes into the state.
- Relevant blog from Vitalik: https://notes.ethereum.org/@vbuterin/purge_2024_03_31

Gas observability

- The removal of gas observability is also in the roadmap

Portal network

- Portal network is a specialized storage engine for Ethereum's data
- The idea
  - The process of dealing with all of the data within Ethereum can be split and addressed separately in specific networks. And they are referenced and linked through cryptography.
  - So rather than having loads of full nodes, that are hard to run, nodes can be split and run separately on lighter clients



- Client - Network dynamics
  - Current model: The clients are very heavy, and the network is very light eg. devp2p
  - Portal network: The clients will be light, and the network will be complex
- 5 new decentralized storage networks
  - Beacon light client: beacon chain light protocol data
  - State network: account & contract storage
  - Transaction gossip: lightweight mempool
  - History network: headers, block bodies, receipts
  - Canonical txn index: TxHash > Hash, Index
- Relevant videos
  - https://www.youtube.com/watch?v=0stc9jnQLXA
  - https://www.youtube.com/watch?v=qqUOr6LgYy0&pp

Q&A

- Is it difficult because precompiles are not defined EVM's language/code for zkEVM?
  - Precompiles are not EVM code. Typically, they tend to be machine code. And the problem is still pretty much an open question.
- Regarding SelfDestruct, what's the storage leakage problem?
  - As Ethereum currently stores the state using MPT, only the reference, i.e. code hash and storage root, is stored in the state tree. A client who's doing selfdestruct has to decide whether or not it's going to delete the code itself afterwards.
  - The hygenic way to do it is delete the contract storage. But to implement the deletion naively means possible DDOS off the network for the client as for a large contract, there might be thousands/ millions of storage keys that need to be deleted. The realistic way for a client is just not delete them or put a strict limit on how many keys to be deleted each time, as this is not the most important thing most clients are working on.
  - Regarding the contract code, if the client chooses to store it under the code hash, there could be multiple accounts deploy the same code. And when a contract does the selfdestruct, the client cannot delete the code from its db. That's where the leakage happens.
- Will state expiry ever happen? Is it currently a low priority?
  - Don't have an ecosystem informed opinion here. It can be completely viable to never implemet state expiry if the growth of hardware performance outperformance the state growth.
  - But the state is a very hard dataset to work with and state expiry puts boundaries on the overall size of the state and let users have more choice to run on light clients, rather than turn into more centralized 3rd party providers.
- Does address space expansion have a high priority?
  - EOF team is doing some of the work to make it possible.
  - Overall, most things in our ecosystem get done because of a few very motivated people who decide to die on that hill, who decide to take up the flag and run with it and make it happen.
- What proofs is a portal client expected to serve, so it is not considered a leecher and kicked off?
  - Every piece of data that the portal serves is cryptographically anchored and canonically proven. For a client in our network to store anything, it will always guarantee to anchor itself to canonicleness. We also have designed all of our access patterns so that things are provable in the way that they are accessed.

    - But it's still unknown how that grows over time and many problems are expected to happen & to be solved.
- How much storage and bandwidth does a client need to participate in the network? Will mobile phones be allowed to participate state?
    - Mobiles are possibly viable in some limited context. But that is going to be client specific thing and we will react to it overtime as we see how the problem unfolds.
- How to participate in the portal network now?
    - Currently you can participate through bridges: https://blog.ethportal.net/posts/portal-bridge-4444s-guide
    - Or you can also run through a portal network clients, but this approach is not that ready to onboard lots of users currently.
- Will the portal network also be used on L2s?
    - Portal needs critical mass nodes to store data and run the system, so Ethereum L1 is ideal for that.
    - L2s are on the long horizon roadmap.