I'm taking the opportunity to post some some research that my colleague Surya Sankagiri and I did which relates to stateless cryptocurrencies. You can find a link to our paper here. While our project focused on applying statelessness to Bitcoin, it took inspiration from the use of the binary Merkle Tries that this forum has discussed for an Ethereum shift to statelessness, and I think that there are a few ideas that the Ethereum community will find interesting, especially in light of Vitalik's recent posts on paths forward for limiting the Ethereum state size.

The Locality Principle for Statelessness

The main idea of our work stems from theUTREEXO paper, which made the observation that in the Bitcoin network, coins tend to be spent very soon after they are sent. This has implications for stateless nodes that use hash tree accumulators, as UTREEXO and our work do, and as the Ethereum stateless initiative seems to be planning to do.

To summarize our findings, it is good to keep recently touched parts of the state nearby each other on the state tree, as this leads to smaller witness sizes

. Our paper constructs an accumulator which conforms to this principle by keeping the transactions in the Bitcoin state in a binary trie. Transactions are appended to this trie in the order of their insertion into the blockchain, and they are deleted when they are spent. This strategy makes witnesses tend to share proof data with each other and ultimately cuts down the witness size as compared to UTREEXO.

There is a good chance that this approach would work even better for Ethereum than it does for Bitcoin. In addition to direct externally-owned-account-to-externally-owned-account transfers, Ethereum also has popular smart contracts that are touched frequently. Under this proposal, accessing these smart contracts would contribute very little per-contract cost to the witness size, since these contracts would tend to be located nearby each other in the trie.

What would this look like as a stateless Ethereum proposal?

Current proposals for binarizing the state tree stateless ethereum involve storing the balance, nonce, code, and storage for an account at certain locations in the tree associated with the address of the account. What I would propose is to instead include the address itself

as a value to be stored alongside these other pieces of data, and have the location of the account in the tree depend on the time the account was last changed.

- location + 0x00

for address

- location + 0x01

for balance

- location + 0x02

for nonce

- location + 0x03 + chunk_id

for code

- location + 0x04 + storage_key

for storage.

Whenever an account is touched in a block, this subtree is deleted from its location and reinserted at new_location := current_block_height + index_within_block

How would state expiry work

Vitalik's post on State size management identified "Refresh by touching" as a nice way to do state expiry. This proposal integrates this idea rather naturally: If we are expiring all data that has not been touched since a given block height, we just forget the left part of the tree consisting of nodes in locations below that corresponding to the block height.

This can also be seen as a compromise between the "one-tree approach" of having one tree, some of which is expired, vs the "two-tree approach" of having a second separate tree for the expired data. In this case the "second tree" is just the left part of the main tree. We sidestep the problem of tree rot, where expired parts of the tree prevent new accounts from being created, by creating all new accounts on the right side of the tree, irrespective of address.

Drawbacks

There are a few drawbacks to this scheme, which I'll cover here.

- The subtree delete and move operation is a complicated primitive to implement.
- To prevent account collisions, it would be necessary to ensure that new accounts can't be made that have the same identification as old accounts. One could do something similar to the extended address scheme proposed here, but instead of appending the year to the account, you append the block number to the account.
- The proposal as I've stated it does not have storage slot level granularity but only contract level granularity. This would mean that if an old contract were resurrected it would only bring back the touched parts of the state, but if an contract were to stay alive in the state for a long time, it could accumulate storage indefinitely. This could be fixed by a separate inclusion of timestamps into the storage tree to expire parts of contract data that had not been recently touched.

Thanks

I'd be happy to know what you all think of this, and whether there are any other big drawbacks I may have missed.