It has been a while since a comprehensive “roadmap” has been published or updated for Stateless Ethereum, and a lot has changed. Seems like a good time to “write it all down” again.

What we are NOT doing

We are NOT

solving stateless block mining

We are NOT

solving the DSA (Dynamic State Access) issue nor are we attaching witnesses to transactions for the purpose of execution.

Critical Path

It is difficult to nail down exactly what our goals are, but I think there is a natural hierarchy of goals that emerges when you look at what is “needed”.

Stateless validators for “The Merge” via block witnesses

We want/need validators to be able to validate blocks without burdening them with the state. The proposed way to do this is to enshrine block witnesses into the protocol such that a client could use the witness to validate the resulting state root from executing a block.

To do this we need

A: the witness to be much smaller than it is today (worst case witnesses are in the 100’s of MB with the current hexary patricia trie).

B: the witnesses to be reliably available along with the block.

We solve A with Verkle Tries which reduces proof overhead to a constant size giving us a theoretically upper bound of ~800k and an average witness size of ~200k (estimates based off of current gas limit of 12.5 million). See also the proposed scheme for the unified verkle trie. It is also worth noting that the unified verkle trie is dependent on either modifying the behavior of the SELFDESTRUCT

opcode, or removing it entirely.

We solve B by both enshrining witnesses into the protocol (probably as access lists in the block header) so that someone receiving a proof can verify it is the correct proof for that block. The actual responsibility for producing witnesses and making them available over gossip has not been defined yet.

Further reading on why statelessness is important in the Eth2 context

Mitigating state growth via State Expiry

Block proposers (or miners) will still need to generate blocks. We do not

propose trying to tackle stateless block mining, which leads us to the goal of reducing the ever growing burden of maintaining the state.

The goal is to have economic bounds on the total state size. The leading plan to accomplish this is referred to as “State Expiration” and is outlined here: Resurrection-conflict-minimized state bounding, take 2 - #17 by vbuterin

In broad terms, the plan is to have state become “inactive” after a period of time (~12 months). State that is inactive is no longer managed by the protocol. Any interactions with inactive state require

an accompanying proof to elevate the state back to being “active”. This scheme pleasantly doesn’t introduce any complex “rent” mechanics into the EVM, but still effectively imposes “rent”. The result is a firm upper bound on total state size.

Less Critical Path

Stateless Client Infrastructure via “Portal Clients”

More in-depth reading here:Complete revamp of the "Stateless Ethereum" roadmap - #2 by dankrad

The current DevP2P ETH

protocol which supports the network is not well suited to support stateless clients, nor is the protocol very easy to modify to

add this support. This means that with only the “critical path” items, we can

build clients that work for the Eth1+Eth2 merge infrastructure, but those clients will not be in any way useful for what most people use their client for (JSON-RPC).

A separate initiative is underway to build out the networking infrastructure necessary to support widely deployed ultra-lightweight “portal clients”. The term “portal” here means that these clients have a view

into the network and accompanying data, but don’t necessary participate in the protocol in any meaningful way.

A “Portal Client” will be a participant in a number of separate special purpose peer to peer networks designd specifically to serve the following needs:

1. retrieve arbitrary State on-demand.
2. State Network DHT - Development Update #2 - #5 by pipermerriam
3. State Network DHT - Development Update #2 - #5 by pipermerriam
4. retrieve arbitrary Chain History on demand.
5. Alexandria - HackMD (outdated but conceptually representative of the planned solution)
6. Alexandria - HackMD (outdated but conceptually representative of the planned solution)
7. participate in Transaction Gossip without access to the state.
8. Scalable Transaction Gossip - #3 by pipermerriam
9. Scalable Transaction Gossip - #3 by pipermerriam
10. participate in Block Gossip without the other implicit requirements imposed by the DevP2P ETH

protocol.

Any “stateless client” which wishes to expose the user facing JSON-RPC APIs would be a participant in these networks. We also expect existing clients to end up leveraging these networks to make themselves more lightweight.

This is not on the critical path for the primary goal of the Eth1+Eth2 merge, however, it is required for stateless clients to be useful beyond the validator use case.

Regenesis (but maybe without the state clearing)

Previously, the “Regenesis” concept wrapped up two distinct concepts:

1. Re-booting the chain with a new genesis block and agreed upon genesis state.
2. Moving state to be “inactive” and requiring proofs to move it back to “active”

The active/inactive mechanism is now covered by “State Expiry”.

There is still a lot of benefit to re-booting the chain at a new genesis block, primarily in eliminating the implicit need for all clients to implement all historical fork rules, and allowing clients to be simpler. This would also improve sync times for nodes that wish to do attain a full copy of the state.

Things no longer on the critical path

Binary Trie

Originally slated as the primary mechanism to reduce witness sizes. This has been replaced by Verkle Tries

Ethereum Improvement Proposals

EIP-3102: Binary trie structure

Details on Ethereum Improvement Proposal 3102 (EIP-3102): Binary trie structure

Code Merklization

Originally slated as the secondary mechanism to reduce witness sizes. This has been replaced by Verkle Tries

EIP-2926: Chunk-Based Code Merkleization

Details on Ethereum Improvement Proposal 2926 (EIP-2926): Chunk-Based Code Merkleization