

State size management in Ethereum

Introduction guide

Outline - questions in sequence

- Why is state a valuable resource and to whom? (can this value be replicated)
- Why state size needs to be managed? (effect on system performance + possible partial mitigations)
- How can state size be managed? (feedforward vs feedback control)
- What metrics need to be regulated? (state size, state growth, or both)
- Do metrics need to be in-state or off-state?
- What parameters (control inputs) do we use? (cost of state expansion, charge per size unit per block)
- Can controlled system change to weaken or evade control? (dark rent, miner kick-backs)

Why is state a valuable resource and to whom?

Ethereum state provides these resources:

1. It keeps pieces of data (e.g. established claims, like a token balance) automatically available and their authenticity provable at any time. In this case, **value is provided to the beneficiaries of the claims.**
2. It enables ad-hoc permissionless stateful communication (i.e. beyond the boundaries of 1 transaction) across contracts (synergy), a higher order resource. Example is the synergy between DAI token system and decentralised exchanges. In this case, **value is provided to the users transacting via such synergies.**

Why state size needs to be managed?

Placing information into the Ethereum state, while providing value to users, leads to degradation of performance of the the Ethereum nodes:

1. Slower transaction processing (verification) due to reading from the state
2. Slower block sealing (verification) due to construction of the new state root after modifications
3. Longer snapshot synchronization due to bandwidth consumption to transmit the entire state
4. Lower success rate of snapshot sync due to state pruning by the peers

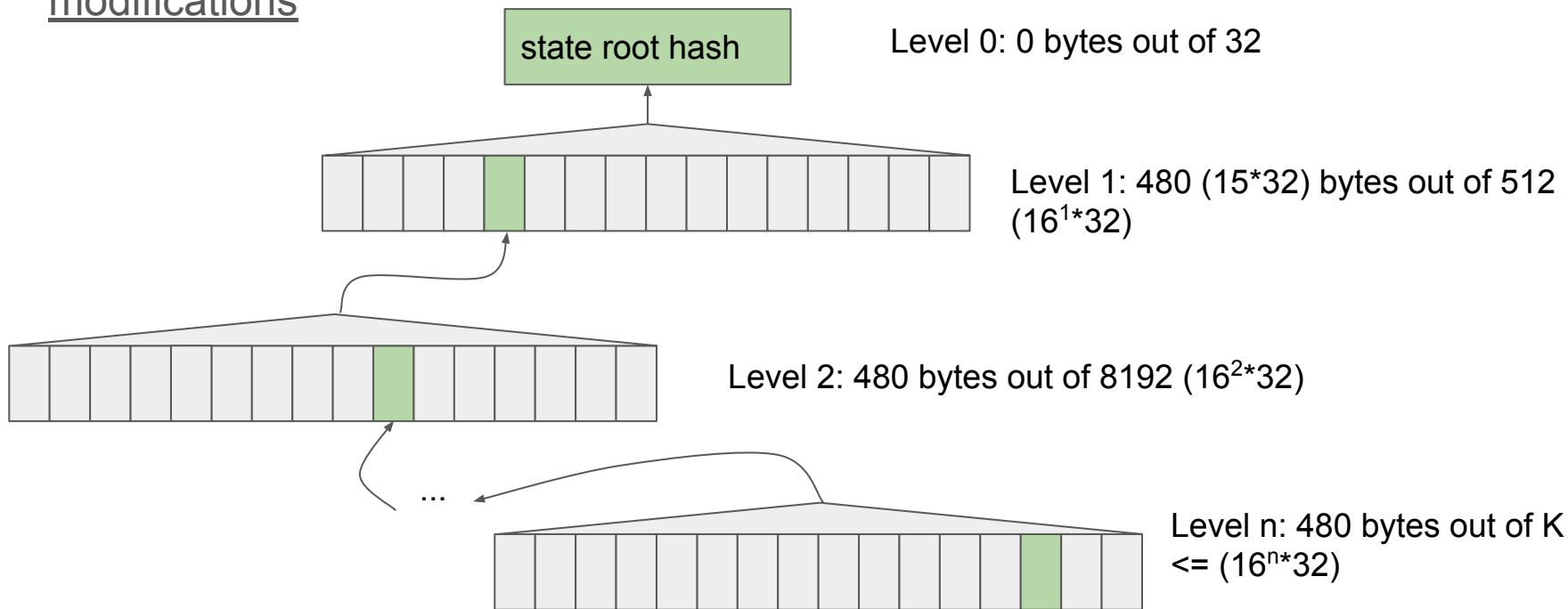
Why state size needs to be managed? (1)

Slower transaction processing (verification) due to reading from the state

Theoretically, if state is stored as key-value pairs (like in Turbo-Geth), transaction processing is slowed down as \log (due to the search in the database index) of the state size. If, instead, the state is stored as a trie (like in geth, Parity, etc), the **transaction processing is slowed down as \log^2 of the state size** (one \log due to number of levels in the trie, another \log is due to the search in the database index), and **linearly with the number of unique items accessed in a block of transactions**.

Why state size needs to be managed (2)

Slower block sealing (verification) due to construction of the new state root after modifications



Why state size needs to be managed (2)

Comments to the previous slide:

To reconstruct merkle root of the state, it is required to know all the sibling hashes in the path from the state root to the modified item. This means fetching 480 bytes for each tree level (there are 15 sibling hashes on each level, 32 bytes each). On average, it could be 7-8 levels, meaning 3840 bytes per item. Some of these 3840 bytes will overlap. More reasoning and simulation is needed, but it can be expected that modifying 1 items in the state requires fetching about 2k of data out of the state trie. Therefore, **it slows down block processing as \log^2 with the size of the state and linearly with the number of items modified in a block.**

Why state size needs to be managed (3)

Longer snapshot synchronization due to bandwidth consumption to transmit the entire state

General idea of snapshot sync is to transmit the entire state from group of peers to a new node. Depending on the scheme, state authenticated either once at the end of the sync (warp sync), or piece by piece (fast sync).

Naively, **sync duration grows linearly with the size of the state.**

Why state size needs to be managed (4)

Lower success rate of snapshot sync due to state pruning by the peers

With the growing state size, most nodes start pruning aggressively.

If a new peer initiates a fast sync (downloading the state), and does not complete within 1 hour, all peers that only keep last 256 blocks ($256 \times 15s \sim 1h$), would have pruned the data necessary for the new peer to complete the sync. So it has to restart the sync with the risk of never completing.

Intuitively, rate of snapshot sync success would be highly non-linear, and dependent on the ratio between prevailing network bandwidth and prevailing pruning threshold. The larger is the state, the lower is the threshold.

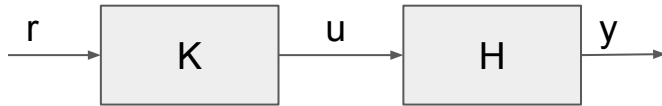
Possible partial mitigations

It might be possible to mitigate some of the performance effects of the state size with short terms measures.

- Latency (but not throughput) of block processing can be improved by pre-announcing “half-mined” (half of PoW difficulty) blocks, letting nodes to warm up the state caches for when the fully mined block arrives.
- All state items (accounts and storage items) can include “last modified block” field, allowing more elaborate syncing protocols that do not need to start from scratch when the sync fails.

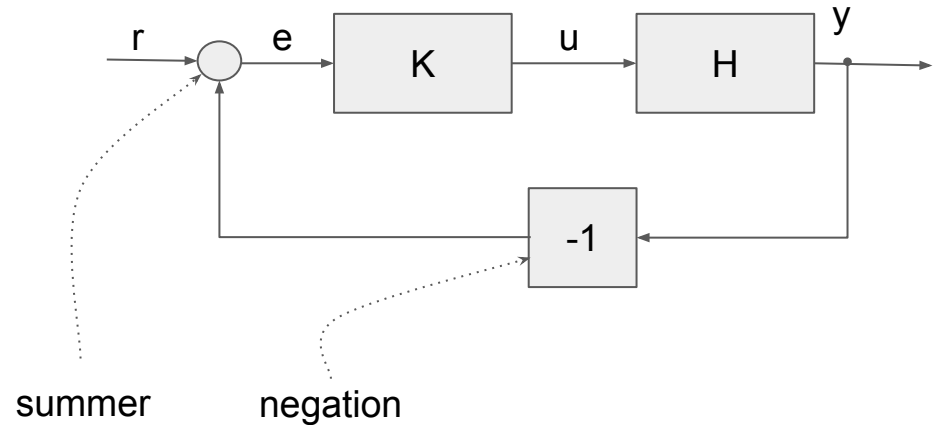
How can state size be managed?

Feedforward control (open loop)

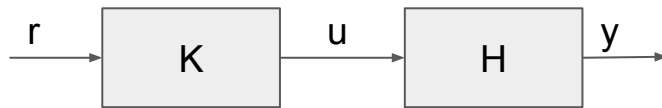


K - controller
H - controlled system
y - regulated output (metric)
r - reference value (parameter)
u - controller output
e - tracking error

Feedback control (closed loop)



Feed-forward control

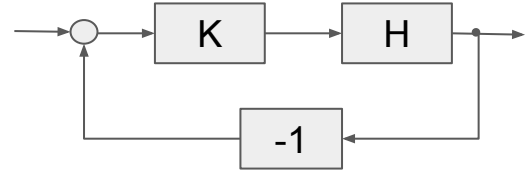


Would be useful if we wanted to regulate something that the protocol cannot observe within itself, for example, latency/throughput of block processing, snapshot sync duration, or snapshot sync success.

Gas schedule of EVM instructions is an example of feed-forward control - the gas costs get adjusted in the hope to eliminate mispricing, which are not observable from the protocol currently.

Obvious drawback of feed forward control is that it can be unreliable, especially if the controlled system (i.e. Ethereum network) keeps evolving, and requires regular changes in the controller (via hard forks).

Feedback control



Obvious example of feedback control in Ethereum is mining difficulty, and it is possible because protocol can perform time measurements using block timestamp.

If we wanted to use feedback control, we would mostly probably target not the performance degradation effects directly, but the state size (or state size growth rate) as a proxy.

What metrics needs to be regulated?

If we choose to use Feed Forward control, we should be targeting the performance effects directly (measured under some standardised benchmarks):

- Block processing latency
- Block processing throughput
- Duration of snapshot sync (via network simulation/emulation)
- Success of snapshot sync (via network simulation/emulation)

If we chose to use Feedback control, we should be targeting size of the state, or a form of moving average of state growth, or both.

Do metrics need to be in-state or off-state?

For feed-forward control, it is obvious that the metrics do not need to be in-state.

For feedback control, the state size (or state growth) needs to be embedded into the state. This is to ensure that the correctness of controller can be verified for any block in isolation (plus some number of previous blocks for moving average).

State growth metric is easier to introduce into the state - it requires an additional field in a block (or in the state under the state root hash). The value of this field can be computed efficiently when processing a block.

State size can be introduced in 2 stages (hard forks): 1) cumulative state growth starts to be recorded. 2) state size at hard-fork 1 is added.

What parameters (control inputs) do we use?

We know what type of actions change the state size (and state growth rate):

- Creation of an account by sending non-zero ETH to an empty
- Creation of a contract by transaction without destination but with data
- Creation of a contract from another contract using CREATE or CREATE2
- Allocation of a new storage item using SSTORE
- Removing a storage item by setting it to zero using SSTORE
- Removing contract by using SELFDESTRUCT

First 4 increase state size, last 2 decrease state size

What parameters (control inputs) do we use?

Since we must not be able to force users to stop making transactions at the times of our choosing, there are 3 possible ways to control state size and state growth

1. Make state increasing actions more costly
2. Make state decreasing actions more rewarding
3. Reduce the state directly (eviction)