Special thanks to @barnabe for suggesting a similar idea at some point earlier

As discussed at length in my older position paper on resource pricing, gas usage in Ethereum actually pays for three distinct kinds of resources:

- Bandwidth

(data in transactions that must be downloaded)

- Computation

(time to verify and execute transactions)

- Storage

(history but more importantly state, eg. account balances, nonces, contract code, contract storage)

Storage is not like the other two costs. Bandwidth and computation are ephemeral costs

, that brush up against ephemeral limits

: there is a bound on how much computation or data downloading a node can do within the span of one block, and once that block passes, the effort needed to download and verify that block is mostly gone (only a few syncing nodes will need to process it in the future). Storage, on the other hand, is a persistent cost

. If a single block increases state size by 100 MB, that block can be processed just fine in the moment

, but a series of blocks like that for an entire month will render Ethereum entirely unusable. The “burst” impact of heavy state growth is negligible, but the long-term impact is the most severe of all, because a piece of state created once burdens the network forever.

With state expiry and weak statelessness, the long-term impact of state will certainly be greatly reduced: instead of burdening the network forever, a piece of state will only burden it for about a year, and even for that year only a smaller portion of nodes would need to actually store that state. But even still, this long-term cost continues to be real, and will continue to need to be priced.

Average case vs worst case storage size

One of the weaknesses of modeling state, both in the current protocol (broadly agreed to be unsustainable) and the improved protocol with state expiry, is the enormous difference between average

and worst-case

state growth. Consider the current protocol. Today, the total size of the state is about 550 million objects, or about 32 GB (not including trie overhead). If we took out all state that was not touched in the previous year, that would easily drop by more than half.

Now, what is the worst case? Contract code creation is charged at 200 gas per byte, so if we split a block into three transactions each of which creates a contract, we can make three 20558-byte contracts for 12334800 gas plus 3 * 55000 gas for contract creation overhead. Thus, in a single block storage size can increase by ~20600 * 3, or 61800 bytes. Assuming an average block time of 13.1s, there’s 31556925 / 13.1 = 2408925

blocks per year, so in total, the state can grow by ~61800 * 2408925 = 148871600381.67938

bytes, or about 138 GB.

This difference, a factor of ~10, is very significant! Particularly, 16 GB can fit within realistic consumer hardware RAM (if not we can tweak gas prices or the state expiry period to make it fit), 138 GB cannot. It would be nice if we could force the worst case to be closer to the average case.

Dual-track EIP 1559

A natural solution to the problem is to use EIP 1559 pricing for both ephemeral and permanent costs, but make the adjustment period different

. In the ephemeral case, prices can adjust by over 10% in a single block. For permanent costs, however, we would make the price adjust much more slowly. If we take the AMM cost curve mechanism as a base, for storage we could consider a curve where the target rate is 1 GB per month, and costs increase depending on how far above the target we are: for example, for every 1 GB over target, storage costs could double. It would take ~3 days of worst-case blocks for the price of storage to double in this parameter. If storage growth were to go 10 GB over target, storage costs would be 1000x higher than normal, making it economically infeasible to fill storage any further.

There are two ways to implement this:

1. Purchase storage with gas

. That is, using SSTORE to create a new storage slot would consume gas as it does today, but the quantity of gas consumed would be variable. This has the weakness that it preserves timing mis-incentives (users would prefer to fill storage on weekends when gas prices are lower, despite the fact that this does not benefit the network)

1. Purchase storage with ETH

. Transactions (and calls) would be required to supply another resource in addition to gas (we could call it mana

), and this resource would be charged with a similar mechanism to gas, except with different parameters. This has the weakness that it complicates calling rules and requires adding a new CALL opcode.

There are also two hybrid options:

1. We can price

storage in ETH, but charge

in gas

(so if the basefee went up by 2x then the gas needed to fill a storage slot would automatically halve). We can exclude ETH used to expand storage from EIP 1559 gasprice update rules or even the block gas limit.

1. Reform gas more comprehensively, into three concepts: gas, execution points, and storage points

. 1 gas = 1 wei; a transaction allocating gas just means that it's transforming some of its wei into a special form that can be used to pay for resources. This form works the same way as gas in terms of how it passes between calls and subcalls. However, there are now two costs that are managed by the AMM: the cost of an execution point and the cost of a storage point. When the execution processes an opcode that currently costs N gas, it instead costs N execution points, meaning that N * execution_point_cost

gas is charged. Filling a storage slot costs 1 storage point, so storage_point_cost

gas is charged.

Note also that the state expiry roadmap is expected to remove refunds. This is because for technical reasons storage slots cannot "become empty" and become eligible for a refund; they can only be set to zero, and the zero record has to remain in the state until that epoch ends and the state can expire. This greatly reduces the complexities that plagued older attempts at storage rent.