Many resources in the EVM have the property that they have very different limits for burst capacity

(how much capacity we could handle for one or a few blocks) and sustained capacity

(how much capacity we would be comfortable having for a long time). Examples include:

- EVM usage

: blocks occasionally taking 2s to process may be okay, but every

block taking that long would make it extremely difficult to keep a node synced

- Block data

: clients have enough bandwidth to process 2 MB blocks when they come, but not enough disk space to store them

- Witness data

: same concern as data - clients have enough bandwidth to process medium-big witnesses, but not enough disk space to store them

- State size filling

: there's basically no limit to how much it's okay to let state increase in a single

block (if state blows up from 45 GB to 46 GB in one block but further state growth goes back to normal, who will notice?) as long as the witnesses can handle it, but we can't have rapid state growth in every block

The scheme we have today, where all resources are combined together into a single multidimensional resource ("gas"), does a poor job at handling these differences. For example, on average transaction data plus calldata consumes ~3% of the gas in a block. Hence, a worst-case block contains ~67x (including the 2x slack from EIP 1559) more data than an average-case block. Witness size is similar: average-case witnesses are a few hundred kB, but worst case witnesses even with Verkle gas reforms would be a few megabytes in size, a 10-20x increase.

Shoehorning all resources into a single virtual resource (gas) forces the worst case / average case ratio to be based on usage, leading to very suboptimal gas costs when the usage-based ratio and the ratio of the burst and sustained limits that we know clients can handle are very misaligned.

This post proposes an alternative solution to this problem: multidimensional EIP 1559.

Suppose that there are n

resources, each with a burst limit

b_i

and a sustained target

s_i

(we require b_i >> s_i

). We want the quantity of resource i

in any single block to never exceed b_i

, and for long-run average consumption of resource i

to equal s_i

.

The solution is simple: we maintain a separate EIP 1559 targeting scheme for each resource! We maintain a vector of basefees f_1 ... f_n

, where f_i

is the basefee for one unit of resource i

. We have a hard rule that each block cannot consume more than b_i

units of resource i

. f_i

is adjusted by a targeting rule (we'll use exponential adjustment since we know now that it has better properties): f_{i,new} = f_{i,old} * exp(k * \frac{u_i - s_i}{s_i})

.

To make this work in an Ethereum context, where there is only one resource (gas) that gets passed from parent calls to child calls, we still charge everything in gas.

- Option 1 (easier but less pure)

: we keep gas costs of execution fixed, and we keep the current EIP 1559; let f_1

be the basefee. The gas prices of all "special" resources (calldata, storage use…) become \frac{f_i}{f_1}

. Blocks have both the current gas limit and the limit b_1 ... b_n

on each resource. Priority fee works in the same way as today.

- Option 2 (harder but more pure)

: the gas basefee is fixed to 1 wei (or if we want, 1 gwei). The gas price of using each resource (of which execution is one) becomes f_i

. There is no block gas limit; there are only the limits b_1 ... b_n

on each resource. In this model, "gas" and "ETH" become truly synonymous. Priority fee works by specifying a percentage; priority fees paid to the block producer equal base fees times that percentage (an even more advanced approach would be to specify a vector

of n

priority fees, one per resource).

Multidimensional pricing and knapsack problem objections

The main historical objection to multidimensional pricing models has been that they impose a difficult optimization problem on block builders: block builders would not be able to simply accept transactions in high-to-low order of fee-per-gas, they would have to balance between different dimensions and solve a multidimensional knapsack problem. This would create room for proprietary optimized miners that do significantly better than stock algorithms, leading to centralization.

This problem is much weaker than before in two key ways:

1. Miner extractable value (MEV) already

creates opportunities for optimized miners, so "the ship has sailed" on stock algorithms being optimal in a meaningful way. Proposer/builder separation (PBS) addresses this problem, firewalling the economies of scale of block production away from the consensus layer.

1. EIP 1559 means that any resource hitting a limit is an edge case and not an average case, so naive algorithms will underperform only in a few exceptional blocks.

To see why (2) is the case, we need to note one very important fact: in multidimensional EIP 1559, the "slack" parameter (\frac{maximum}{target}

) for each resource can be much higher than 2x

. This is because the 2x slack parameter of today creates a burst/sustained gap that stacks on top of the burst/sustained gap that comes from unpredictable usage, whereas in multidimensional EIP 1559, the slack parameter represents the entire

burst/sustained gap. For example, we could target calldata usage to ~256 kB (~8x more than today), have an 8x slack parameter (\frac{b_i}{s_i}

) on top of that, and still have comparable burst limits to today. If witness gas costs are unchanged, we could bound witness size to another ~2 MB and have a slack parameter for witness size of ~6x. A survey of 240 recent blocks suggests that only 1 of those blocks would have hit the limit under a calldata slack parameter of even 4x!

This shows a nice side effect of multidimensional EIP 1559: it would make the boundary case of priority fee auctions much more rare, and clear sudden burst of transactions more quickly.

What resources could be multidimensionally priced?

We can start with the basics:

- EVM execution
- Tx calldata
- Witness data
- Storage size growth

When sharding is added, shard data can also be added to this list. This would already give us a lot of gains in being able to support more scalability while mitigating risks from burst usage.

In the longer term, we could even make the pricing much more fine-grained:

- Split witness by read vs write
- Split witness by branch vs chunk
- Separately price each individual precompile
- Calls
- Each individual opcode

The main value of this is that it would add another layer of DoS protection: if each opcode is only allocated eg. 100 ms of max expected execution time, then if an attacker discovers a 10x slowdown in one opcode or precompile, they can only add 900 ms expected execution time to the block. This is in contrast to today, where they can fill the entire block with that opcode or precompile, so a 10x slowdown in any

single opcode or precompile could allow the attacker to create blocks that cannot be processed in time for a single slot.