For background, see Tradeoffs in Account Abstraction Proposals

In this post I'll make an account abstraction proposal that I would argue achieves significantly greater simplicity, and greater generality, than anything proposed so far, but at a price: transactions whose gas costs are not completely static may end up overcharging gas.

The proposal is simple. We add an opcode BREAKPOINT

, with the property that a function call that fails after a BREAKPOINT

reverts only up to the BREAKPOINT

. User account code would in general be structured as follows:

verify_nonce() verify_signature() send(coinbase, x) breakpoint() do_stuff() do_more_stuff()

After the execution hits the BREAKPOINT

opcode, the block proposer is certain that they will get compensated for including the transaction. Note that in this model, refunds for unused gas are not

possible.

To add more flexibility, we can add another opcode, DECREASE_LIMIT

, which decreases the remaining gas limit without consuming gas. This would allow for account code where the gas limit of a transaction can be determined in the "header" (ie. before the send

and BREAKPOINT

).

Consequences

- apply_msg

and apply_tx

become identical (fee market reform can be done at the per-block level), greatly reducing complexity

- The ABI would need to specify the max gas consumption of each function call, so that tight maximums can be more easily computed
- Does not require static analysis of code
- Would lead to some inefficiency in cases where gas costs truly are variable (the most common use case being CREATE2'ing a contract if and only if it does not exist yet), as the user would need to pay for the higher level of consumption even if the lower level of consumption is more frequent
- Implements full abstraction, so we would lose the tx hash uniqueness guarantee