One of the challenges of the current phase 2 spec is as follows. The abstraction model has ensured that it's possible for dozens of different kinds of environments (eth1-like, eth1-like-improved, ZEXE, UTXO-based…) to be run on top of the chain, but keeping the consensus code that validators need to run very small. A node only needs to have knowledge of one of these environments if it is personally interested in it. But this only solves the problem of validation; it does not solve another large problem that validators have, which is optimal fee collection: how do validators know what to include that will pay them the most fees, when there are 74 different environments with 33 different ways of representing currency, which the validator does not understand? A highly abstracted spec that makes validation universal and simple is of limited value if the code needed to run to effectively propose blocks and collect fees still requires knowledge of 74 different sub-protocols

.

One proposed mitigation in the earlier spec design was relayer markets

. Essentially, there would be a category of intermediaries that each understand one more more of the different environments and aggregate transactions for them and collect them into packages that pay fees using some standard method that the validators understand. These relayers could also perform tasks like adding or aggregating Merkle branches or other witnesses. Alternatively, transaction senders that do not want to go through relayers could simply "self-package" their transactions, making a package with only their transaction included (potentially, transaction senders could create packages containing both their transaction and

other transactions that they have seen). But what is the universal fee payment medium that these systems would use?

The basic universal fee market

Here is one proposal. For each transaction package, it would be standard for an EE to create a Merkle branch that is verifiable against the state root in a standard position that contains a receipt with the fee to be paid (eg. state = SSZContainer(real_state: RealState, fee: uint64)

would make fee

easy to verify with an SSZPartial). It would be the EE's responsibility to charge the fees to the actual transaction senders; the fee

specified in this fee

value would be charged to the execution environment

itself.

There would be one EE called the "fee market EE", where other

EEs (as well as block proposers) could hold accounts with balances. If anyone submits a Merkle branch showing that some EE issued a receipt paying some fee then the fee market would deduct that fee from that EE's account and add it to the account of the proposer address of that block. The fee market EE could require branches to be submitted in slot order for each EE, removing the need for any more complicated replay protection.

The process for a block proposer looking to accept transaction packages would be to do two things:

1. Run the transaction package and verify that it produces a fee receipt (as an optimization, EE execution could pre-signal that it will produce a certain fee receipt as soon as it knows this, and if an EE is found to even once lie with these pre-signals then it could be automatically blacklisted)

2. Verify that the EE has an account with the fee market EE with enough funds to pay for the fee specified in the receipt.