We need to come up with a standard way to score the value of a block under the mev-boost architecture. This standard is a dependency for a multitude of components including relay monitoring, validator accounting, builder payments, block explorers, payment proofs, and mev hiding. Defining this early and correctly will help avoid a lot of pain and confusion in the future.

Note that a “block score” is not meant to be a formal definition of realized extractable value since this is a difficult metric to quantify. Rather, a block score is the standard metric recognized by the mev-boost system and used for profit switching between relays. An example of a validator payment which would clearly fall outside of the scope of a block score is a transfer to the validator account on an L2.

Some definitions

For clarity, lets refer to the feeRecipient

field set by the block builder in the block header as the coinbase address. We can refer to the address provided by the validator to collect fees as the fee recipient. The builder is not required to set the fee recipient to the coinbase address on the blocks they produce.

For context, here are the various payments methods on Ethereum which are relevant to keep in mind in this discussion:

- base_fee

: amount defined by protocol, always paid from user → network, user can define upper bound they are willing to pay using maxFeePerGas, the user needs to have sufficient ETH balance in their account before transaction execution - this is not MEV since there is no way for the validator to get this payment and therefore should not be included in the block score. The payment cannot be made within of EVM execution (unfortunately).

- priority_fee

: amount defined by the user as maxPriorityFeePerGas, always paid from user → coinbase address. The user needs to have sufficient ETH balance in their account before transaction execution. The priority fee is debited from the user account and credited to the fee recipient at the end of transaction execution.

- block.coinbase transfer

: the EVM has an opcode which allows end user to send funds directly to the coinbase address. The recipient of this payment is the same as the priority_fee

payment recipient, but this transfer triggers EVM execution including associated gas costs and security risks.

- regular transfer

: anyone can create a transaction which sends eth to any other Ethereum account. Since the proposer is publishing their desired fee recipient address to the public, it’s possible for any user to send payments directly to this account.

Some options

1. Block level scoring

Block level scoring is the difference in the balance of the fee recipient account before and after the block execution.

This is probably the most simple, intuitive, and generalized way to score a block. The main downside I see here is that a validator cannot withdraw fees in the same block they propose without impacting the block score.

Constructing a payment proof for this scoring method requires a merkle proof on the balance of the fee recipient in block n - 1 and a merkle proof on the balance in block n.

2. Transaction output scoring

Transaction output scoring is the difference in the balance of the fee recipient account before and after the execution of the last transaction in the block.

This is more restrictive than the block level scoring as it does not allow for the fee recpient address to be set as the coinbase address. It instead requires the builder to set their own address as the coinbase address and to include a transaction payment transaction in the block.

While it solves the withdrawl problem, it adds some gas overhead to every block and makes payment proofs non-trivial and maybe even impossible (since you can prove that the transaction was included, but not that it did any particular state transition).

3. Transaction input scoring

A slight modification to transaction level scoring could be to score based on the transaction inputs rather than the output. Specifically, the payment transaction can be required to be initiated from an EOA, to have the payment in the value

field, and the fee recipient address in the to

field of the transaction.

One advantage of this approach is that it allows for easily creating transaction inclusion proofs as well as querying the block score without needing to simulate the block (assuming the transaction did not revert).

4. Event level scoring

Event level scoring means looking at a standard event in a given transaction. For example:

event BlockScore(address builder, address validator, uint256 value);

This approach shares some of the benefits of transaction input scoring, but adds significant complexity and requires predefined conventions to determine which transaction in the block and which event index should be used.

Conclusion

Given the options outlined above, we are leaning towards the block level scoring as it is the most intuitive and least restrictive scoring method and provides a very specific definition to the block value. While it makes validator accounting slightly more difficult due to not being able to withdraw in their own blocks, it should not have a big impact beyond the possibility of delaying withdraw transactions by a few blocks and therefore is an acceptable tradeoff.

There are likely other options and tradeoffs I have not yet thought about and would like for other ideas to be submitted.