Background

EIP 2029/2030 centers around creating transaction level access lists for driving optimization through gas cost incentives. A transaction that provides an access_list

as an additional input while being included in a block, gets cheaper costs (~10%) on some of the EVM op codes, and new (first time) accesses outside the provided ones get charged at higher cost. This inadvertently leads to generation of transaction_access_list

which is also being standardized as a JSON-RPC

endpoint.

Motivation

Motivation is to generate a block_access_list

and make its Hash/Commitment

as the part of the block. This will serve as an index into the block where address is being accessed in the transaction,

1. enabling partial inspection of the block by light clients or by fast sync protocols to fetch relevant data
2. block level optimization while verifying the block by enabling the construction of partial order on transaction execution
3. Bundling the witness data for future stateless execution chain

Construction

Currently:

transaction_access_list= Set[AccessedAddress,List[AccessedStorageSlots]]

,

Proposal:

block_access_list=Set[Address, List[AccessedStorageSlots] ,Set[AccessedInBlockTransactionNumber,List[AccessedStorageSlots]]]

To bundle Witnesses, this can be evolved into:

block_access_list=Set[Address,List[AccessedStorageSlots],List[AccessedWitnesses],Set[AccessedInBlockTransactionNumber,List[AccessedStorageSlots],List[AccessedWitness]]]

For purposes of building a canonical_block_access_list

following sorting construction is suggested:

canonical_block_access_list=Set[Address, List[AccessedStorageSlots],Set[AccessedInBlockTransactionNumber,List[AccessedStorageSlots]]]

i.e. everything is just normally sorted. There was an idea to sort the base set of access_list by accessed by time, but above construction is simpler to generate, and on transaction level accessed by order doesn't really matter since its easy to preload all the transaction access data.

AccessListRoot

in Block Header

For this access_list

to have any relevance and serve its purpose as index, its hash

or commitment

needs to be included in the block header

to make it part of the verified chain

that can be used as integrity check against grieving attacks. Now there are two ways to go about it:

- Just hash/commit

this entire canonical_block_access_list

- Create a merkel/verkel

tree out of it and use its root as as the fingerprint

to be included in the block header

.

While the first construction is simpler, second one is more favorable as it will allow partial and/or distributed downloading of the data especially favorable in beam sync

, light sync

and/or stateless witness

protocols. This becomes important when the witness data would be needed to be bundled for achieving stateless ethereum.

Another point to note is, an AccessRootType

can be bundeled in the block header to specify the construction of this root as it evolve over time w.r.t. construction methodology as well as data.

Side Benefits

Despite it not encoding an access by sorting, above construction still allows us to generate a partial order on the transaction execution and accesses so that the block execution/validation can be parallelized. Also, transaction_access_list

s can be fast generated from the above construction, which can remove the need for transactions in block to rebundle this data with them or to atleast preempt fetching that transaction data.

Another side benefit of another construction is potential data compression that can be done while transmitting chunks of this tree as sorted lists can be compressed by sending the diffs rather over the wire.

PS: this work has not been funded or granted by anyone. if you think this adds value to the ecosystem you could tip the author at: 0xb44ddd63d0e27bb3cd8046478b20c0465058ff04