ps: this is an iteration over block access list with inputs and refinements from @pipermerriam

Background

EIP 2929/2930

centers around normalizing (low) gas costs of data/storage accesses

made by a transaction as well as providing for (and encouraging) a new transaction type format:

0x01 || rlp([chainId, nonce, gasPrice, gasLimit, to, value, data, access_list, yParity, senderR, senderS])

that makes upfront access_list

declarations

, where

access_list

is some [[{20 bytes}, [{32 bytes}...]]...]

map

of AccessedAddress=> AccessedStorageKeys

.

The first accesses

of these upfront declarations

are charged at discounted price (roughly ~10%

) and first accesses

outside this list are charged higher price. Reason is, upfront access declaration provides for a way to preload/optimize/batch

loading these locations while executing the transaction.

This inadvertently leads to generation of transaction

access_list

that has all first accesses

(declared or not) made by a transaction. A JSON-RPC

api endpoint for creating and fetching this list is being standardized.

Motivation

Motivation is to collate these transaction access_list

s for all the transactions in a block's access_list

document, that can serve as sort of access index

of the block with following benefits:

1. Block execution/validation optimizations/parallelization by enabling construction of a partial order

for access and hence execution (hint: chains

in this poset

can be parallelized).

1. Enabling partial inspection and fetching/serving of a block data/state by light sync

or fast sync

protocols concerned with a subset of addresses.

1. Possible future extension of this list to serve as index for bundling, serving and fetching witness data for stateless protocols.

To serve the above purpose, and prevent any grieving

attacks, an AccessListRoot

which could be a urn encoding Hash/Commitment

of a canonical

access_list

as well as the construction type (sha256/kate/merkel/verkel

) will need to be included in the block header

.

Forming the tree structure (merkel/verkel

) rather than a simple Hash/Commitment

of the entire canonical

access_list

will be a bit more expensive, but it will enable partial downloading, inspection and validation of the access_list

and is recommended.

Construction

Currently a transaction's access_list

:

Set [AccessedAddress, List [AccessedStorageKeys]]

Proposal a block's access_list

:

Set [AccessedAddress, List [AccessedStorageKeys] , Set [AccessedInBlockTransactionNumber, List [AccessedStorageKeys]]]

But as mentioned, we need to define a canonical

access list construction for purposes of building AccessListRoot

Canonical Block Access List

An access_list

is comprised of many access_list_entry

elements:

access_list := [access_list_entry, ...]

An access_list_entry is a 3-tuple of:

- address
- sorted list of storage keys of the address accessed across the entire block
- sorted list of 2-tuples of:
- transaction index in which the address or any of its storage keys were accessed
- sorted list of storage keys which were accessed

- transaction index in which the address or any of its storage keys were accessed
- sorted list of storage keys which were accessed

access_list := [access_list_entry, ...] access_list_entry := [address, storage_keys, accesses_by_txn_index] address := bytes20 accesses_by_txn_index := [txn_index_and_keys, ...] txn_index_and_keys := [txn_index, storage_keys] txn_index := uint64 # or uint256 or whatever storage_keys := [storage_key, ...] storage_key := bytes32

Additional sorting rules for the above are that:

- access_list

is sorted by the address

- storage_keys

is sorted

- accesses_by_txn_index

is sorted by txn_index

Additional validation rules for the above are that:

- Each unique address

may only appear at most once in access_list

- Each storage_key

may only appear at most once in storage_keys

- Each txn_index

may only appear at most once in txn_index_and_keys

Side Note: Other sorting orders like ordering the elements by accessed by

order were considered but didn’t seem to provide any significant benefit over the simpler lexicographic ordering.

AccessListRoot

For a very simple start we can do a SHA256

of the canonical

access_list

and include it in the block header with the following the urn scheme:

AccessListRoot := "urn:sha256:${ SHA256(access_list) }"

This AccessListRoot

could evolve to urn:merkel:...

or to urn:verkel:...

or to any other scheme as per requirement.

Additional Block Validation

Full nodes while accepting, executing the new block have an additional validation check that the block’s urn

encoded AccessListRoot

matches the one calculated from the one generated by executing the block.

A JSON RPC Endpoint

Recommended implementation of fetching the block access list of a given block something of the sorts of eth_BlockAccessList(BlockNumberOrBlockHashOrAccessListURN, ListOfAddressesOrOffsetOrNull) which can return paginated access_list

, paginated at the top address level.

Future optional extensions

We can extend the notion of a block's access_list

to:

access_list := Set[Address, List [AddressWitnesses], Set [AccessedStorageKey, List [StorageKeyWitnesses]], Set [AccessedInBlockTransactionNumber, List [AccessedStorageKeys]]]

This will allow an incremental path to partial or full stateless chain protocols, where it would be easy to bundle/request witnesses using this access_list

.