To save gas in the main shard (and for the aesthetics of keeping things clean) we propose various collation header changes. For clarity of exposition we reduce the current header in two steps. The aim of this post is to brainstorm several changes to spark a discussion so that the best ideas eventually get cherry-picked to design an optimal collation header.

The currently specified 10-element collation header is:

[shard_id: uint256, expected_period_number: uint256, period_start_prevhash: bytes32, parent_hash: bytes32, transactions_root: bytes32, coinbase: address, state_root: bytes32, receipts_root: bytes32, number: uint256, sig: bytes]

with size 32+32+32+32+32+20+32+32+32+65 = 341 bytes. The first proposed reduced header is:

[collation_id: uint256 expected_period_number: uint256 log_root: bytes32 state_root: bytes32 parent_hash: bytes32]

The changes are:

1. Rename number

to collation_number

and then merge shard_id

and collation_number

into collation_id := (shard_id << 128) + collation_number

. The collation_id

naturally identifies a collation within the 2-dim (shard_id, collation_number)

collation vector space. Notice 128 bits suffice for the two coordinates. Even assuming a new shard is spawned every second or a new collation is added every second there's enough bit space for 10^42 years.

1. Remove period_start_prevhash

as it seems to be derivable from expected_period_number

.

1. Merge transactions_root

and receipts_root

into log_root

. Semantically transactions and receipts are both logs, just a different type. It is natural to merge them under the same accumulator. To distinguish types we suggest adding a corresponding prefix (such as TYPE_TX

and TYPE_RECEIPT

) to the log before hashing. Instead of using a Patria trie, we suggest using a Merkle tree with ordered leaves. By ordering the leaves we retain the power of tries (namely, non-membership proofs) and gain the following: * Exceptional/adversarial O(n) witnesses go away—more predictability, more fairness

- We don't have to suffer the 10% in trie witness overhead estimated by Vitalik
- Possibly a slight performance improvement in Merklelisation
- Exceptional/adversarial O(n) witnesses go away—more predictability, more fairness
- We don't have to suffer the 10% in trie witness overhead estimated by Vitalik
- Possibly a slight performance improvement in Merklelisation
- Assuming collation rewards are awarded in collations (as opposed to the main shard, c.fthis post) then I don't think it is necessary to expose coinbase

in the header.

1. As previously noticed by Vitalik the sig

can be optimised away by reusing the signature from the transaction calling the VMC.

The second proposed reduced header is:

[collation_id: uint256 collation_root: bytes32 parent_root: bytes32]

The changes are:

1. Put expected_period_number

somewhere else. Either: * Fit expected_period_number % (2 ** 32)

in collation_id

. Assuming 14 second block times and 5 blocks per period, then 2 ** 32 blocks corresponds to about 10,000 years.

- Grind expected_period_number % (2 ** 16)

into collation_root

. Assuming 14 second blocks times and 5 blocks per period, then 2 ** 16 blocks corresponds to 53 days which is enough to cover any kind of reasonable shard reorg, and 16 bits is small enough to grind into collation_root

.

1. Fit expected_period_number % (2 ** 32)

in collation_id

. Assuming 14 second block times and 5 blocks per period, then 2 ** 32 blocks corresponds to about 10,000 years.

1. Grind expected_period_number % (2 ** 16)

into collation_root

. Assuming 14 second blocks times and 5 blocks per period, then 2 ** 16 blocks corresponds to 53 days which is enough to cover any kind of reasonable shard reorg, and 16 bits is small enough to grind into collation_root

.

1. Merklelise log_root

, state_root

and the collation header hash into the collation_root

. Merklelise all the things!

1. Replace parent_hash

(a “dumb” hash) by parent_root

(a “smart” root that nicely mirrors collation_root

).

It seems we can make collation headers just 96 bytes (72% reduction). The collation_id

describes “the where”, the collation_root

describes “the what”, and the parent_root

allows for a hash chain.