Recap

A follow up of [A minimal sharding protocol that may be worthwhile as a development target now]

In that post, Vitalik illustrated a stub scheme for the basic committee mechanism that could be the testing backbone before we have a more stable specification of the protocol design.

There are two emitting-log-only functions (I renamed/reorder the parameters):

1. add_header(shard_id, chunk_root, period) returns bool

: anyone can call this function at any time. The first header to get included for a given shard in a given period gets in, all others don't. This function just emits a log. Returns True

on success.

1. submit_vote(shard_id, chunk_root, period) returns bool

: Sampled notaries call this function to submit a vote. This function just emits a log. Returns True

on success.

And since the notaries have to know if they're sampled committee members, they need to use message call to invoke a function which is similar to the outdated get_eligible_collator

:

- get_committee(shard_id, period) returns a list of addresses

: use the last block hash h

before this period as the seed. Selecting notary_pool[sha3(h) % notary_pool_size], notary_pool[sha3(h + 1) % notary_pool_size], ... notary_pool[sha3(h + (COMMITTEE_SIZE - 1)) % notary_pool_size]

notaries from notary_pool

.

This function is also a stub before we have a certain RNG mechanism.

In this scheme, most calculations happen in client side. Shard clients have to query SMC receipt log to decide which collations are included in the canonical shard chain.

Here are some descriptions of this scheme:[Minimal sharding protocol · Issue #539 · ethereum/py-evm · GitHub]

Problem in full sync

When the shard clients want to sync with the shard, before the client calculates the votes, they have to check who are the selected committee members via calling SMC.get_committee(shard_id, period)

. But by the time, notary_pool_len

may be already changed because some notaries registered or deregistered. So the return value from SMC.get_committee(shard_id, period)

may be different from the real selected committee that was generated right after the period started.

Possible solutions

Solution 1 - client side maintaining notary_pool

In Vyper implementation, we already have logs for updating notary pool:

RegisterNotary: **log**({index_in_notary_pool: int128, notary: address}) DeregisterNotary: **log**({index_in_notary_pool: int128, notary: address, deregistered_period: int128}) ReleaseNotary: **log**({index_in_notary_pool: int128, notary: address})

During the full sync, make the client also lookup every RegisterNotary

and DeregisterNotary

log and sample the "committee" off-chain

. So when the client checks the votes, they will know they should sample with which committee.

- Pros:
- No extra gas cost.

- No extra gas cost.

- Cons:
- Higher client-side complexity.
- Lack of a trustworthy committee list snapshot on-chain. How do clients apply other fast sync mechanisms?

- Higher client-side complexity.
- Lack of a trustworthy committee list snapshot on-chain. How do clients apply other fast sync mechanisms?

Solution 2 - on chain checking with adding receipt logs and costing a lot gas cost for adding header

1. Add one storage in SMC: last_period: public(int128)

logs the period of the most recent success add_header

message.

1. Add one check in SMC.add_header(shard_id, chunk_root, period)

:def add_header(shard_id, chunk_root, period) -> bool: assert last_period < period emit HeaderAdded log committee = SMC.get_committee(shard_id, period) emit Committee log return True

- Pros:
- Easy.
- This assertion also forbids the second proposer to send HeaderAdded

log. So no loser-proposer log.

- Easy.
- This assertion also forbids the second proposer to send HeaderAdded

log. So no loser-proposer log.

- Cons:
- High gas cost for proposer to invoke sampling

COMMITTEE_SIZE :=135

notaries and log them.

- High gas cost for proposer to invoke sampling

COMMITTEE_SIZE :=135

notaries and log them.

I think solution 2 is bad and but very trivial. Solution 1 is better if the trend is moving most calculation off-chain.

Since right now, the implementation might be just a quick fix for the stub scheme, I'd like to know which solution would be more compatible with the future scheme integration. Or any other solution can fix this problem more perfectly.