@vbuterin in minimal sharding protocal post: Given the possibility of yet more changes to the sharding 1.1 spec, and developers' concerns that they are building something that could get changed again, I wanted to offer something that is worthwhile as a development target to shoot for right now

, and will be on the path toward implementing the final protocol.

This is an extension of minimal sharding protocal and a patch for Per period committee snapshot problem. Also, some SMC parameters and functions inherit from the retired spec 1.1.

1. To make this minimal sharding protocol as simple as possible at this stage, we set vote counting and collation head to on-chain in SMC.
2. Thank @NicLin , we present a new Per-period Sample Size Snapshot Mechanism

.

This document is mainly for establishing a more workable minimal sharding protocol for sharding client implementers in the short term

.

Any suggestions/feedback would be appreciated!

Parameters

// Shards SMC_ADDRESS := (TBD) NETWORK_ID := 0b1000_0001 SHARD_COUNT := 100 // shards PERIOD_LENGTH := 100 // block times

// Collations COLLATION_SIZE := 2 ** 20 // bytes CHUNK_SIZE := 32 // bytes

// Registries NOTARY_DEPOSIT := 1000 // ETH NOTARY_LOCKUP_LENGTH := 16128 // periods

// Notarization COMMITTEE_SIZE := 135 // notaries QUORUM_SIZE := 90 // notaries

Format

Collation

[header: CollationHeader, body: CollationBody]

CollationHeader

Updated: reordering fields again for (i) potentially combining shard_id

and period

in one bytes32 and (ii) matching on-chain storage format.

[shard_id: int128, period: int128, chunk_root: bytes32, proposer_address: address]

CollationBody

COLLATION_SIZE

bytes data contains blobs

Sharding Management Contract

Storages

Notary pool

- notary_pool: address[int128]
- array of active notary addresses
- notary_pool_len: int128
- size of the notary pool

- empty_slots_stack: int128[int128]
- stack of empty * notary slot indices
- empty_slots_stack_top: int128
- top index of the stack

Notary registry

- notary_registry: {deregistered: int128, pool_index: int128, deposit: wei_value}[address]
- notary registry (deregistered is 0 for not yet deregistered notaries)

Sampling

- current_period_notary_sample_size: int128
- the sample size for current current period
- next_period_notary_sample_size: int128
- the dynamic sampling size for next period
- notary_sample_size_updated_period: int128
- latest period in which current_period_notary_sample_size

has been updated

Collation

- collation_records: {chunk_root: bytes32, proposer: address, is_elected: bool}[int128][int128]
- the collation records that have been appended by the proposer. Mapping [shard_id][period]

to chunk_root

and proposer

.

- records_updated_period: int128[int128]
- the latest period in which new collation header has been submitted for the given shard
- head_collation_period: int128[int128]

: - head collation period number for the given shard

Notarization

- current_vote: bytes32[int128]

: current vote count of each shard * First 31 bytes: bitfield of who has voted and who has not. Each bit represent the iterating number in the get_member_of_committee

function.

- Last byte: a counter of the total number of eligible notaries that voted
- First 31 bytes: bitfield of who has voted and who has not. Each bit represent the iterating number in the get_member_of_committee

function.

- Last byte: a counter of the total number of eligible notaries that voted

Functions

Registering functions

- register_notary() returns bool

: * Checks

: * Deposit size

: msg.value >= NOTARY_DEPOSIT

- Uniqueness

: notary_registry[msg.sender]

does not exist

- Deposit size

: msg.value >= NOTARY_DEPOSIT

- Uniqueness

: notary_registry[msg.sender]

does not exist

- Updates

: * Calls update_notary_sample_size()

(see below)

- Adds an entry to notary_registry
- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Locks a deposit of size msg.value
- Calls update_notary_sample_size()

(see below)

- Adds an entry to notary_registry
- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Locks a deposit of size msg.value
- Returns

True

on success

- Checks

: * Deposit size

: msg.value >= NOTARY_DEPOSIT

- Uniqueness

: notary_registry[msg.sender]

does not exist

- Deposit size

: msg.value >= NOTARY_DEPOSIT

- Uniqueness

: notary_registry[msg.sender]

does not exist

- Updates

: * Calls update_notary_sample_size()

(see below)

- Adds an entry to notary_registry
- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Locks a deposit of size msg.value
- Calls update_notary_sample_size()

(see below)

- Adds an entry to notary_registry
- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Locks a deposit of size msg.value
- Returns

True

on success

- deregister_notary() returns bool

: * Checks

: * Authentication

: notary_registry[msg.sender]

exists

- Authentication

: notary_registry[msg.sender]

exists

- Updates

: * Calls update_notary_sample_size()

(see below)

- Sets the deregistered period in the notary_registry

entry

- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Calls update_notary_sample_size()

(see below)

- Sets the deregistered period in the notary_registry

entry

- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Returns

True

on success

- Checks

: * Authentication

: notary_registry[msg.sender]

exists

- Authentication

: notary_registry[msg.sender]

exists

- Updates

: * Calls update_notary_sample_size()

(see below)

- Sets the deregistered period in the notary_registry

entry

- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Calls update_notary_sample_size()

(see below)

- Sets the deregistered period in the notary_registry

entry

- Updates the notary pool (notary_pool

, notary_pool_len

, etc.)

- Returns

True

on success

- release_notary() returns bool

: * Checks

: * Authentication

: notary_registry[msg.sender]

exists

- Deregistered

: notary_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > notary_registry[msg.sender].deregistered + NOTARY_LOCKUP_LENGTH

- Authentication

: notary_registry[msg.sender]

exists

- Deregistered

: notary_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > notary_registry[msg.sender].deregistered + NOTARY_LOCKUP_LENGTH

- Updates

: * Removes an entry from notary_registry

- Releases the notary deposit
- Removes an entry from notary_registry
- Releases the notary deposit
- Returns

True

on success

- Checks

: * Authentication

: notary_registry[msg.sender]

exists

- Deregistered

: notary_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > notary_registry[msg.sender].deregistered + NOTARY_LOCKUP_LENGTH

- Authentication

: notary_registry[msg.sender]

exists

- Deregistered

: notary_registry[msg.sender].deregistered != 0

- Lockup

: floor(block.number / PERIOD_LENGTH) > notary_registry[msg.sender].deregistered + NOTARY_LOCKUP_LENGTH

- Updates

: * Removes an entry from notary_registry

- Releases the notary deposit
- Removes an entry from notary_registry
- Releases the notary deposit
- Returns

True

on success

Proposing functions

Updated: reordering parameters again for potentially combining shard_id

and period

in one bytes32 and (ii) matching on-chain storage format.

- add_header(int128 shard_id, int128 period, bytes32 chunk_root) returns bool

: * Checks

: * Shard

: shard_id

against SHARD_COUNT

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- First proposal

: period != records_updated_period[shard_id]

, which means only the first

header to get included for a given shard in a given period gets in, all others don’t.

- Shard

: shard_id

against SHARD_COUNT

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- First proposal

: period != records_updated_period[shard_id]

, which means only the first

header to get included for a given shard in a given period gets in, all others don’t.

- Updates

: * Calls update_notary_sample_size()

- collation_records[as_bytes32(shard_id, period)] = {chunk_root, proposer_address}
- records_updated_period[shard_id] = floor(block.number / PERIOD_LENGTH)

- Calls update_notary_sample_size()
- collation_records[as_bytes32(shard_id, period)] = {chunk_root, proposer_address}
- records_updated_period[shard_id] = floor(block.number / PERIOD_LENGTH)
- Returns

True

on success

- Checks

: * Shard

: shard_id

against SHARD_COUNT

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- First proposal

: period != records_updated_period[shard_id]

, which means only the first

header to get included for a given shard in a given period gets in, all others don't.

- Shard

: shard_id

against SHARD_COUNT

- Correct period

: period == floor(block.number / PERIOD_LENGTH)

- First proposal

: period != records_updated_period[shard_id]

, which means only the first

header to get included for a given shard in a given period gets in, all others don't.

- Updates

: * Calls update_notary_sample_size()

- collation_records[as_bytes32(shard_id, period)] = {chunk_root, proposer_address}
- records_updated_period[shard_id] = floor(block.number / PERIOD_LENGTH)
- Calls update_notary_sample_size()
- collation_records[as_bytes32(shard_id, period)] = {chunk_root, proposer_address}
- records_updated_period[shard_id] = floor(block.number / PERIOD_LENGTH)
- Returns

True

on success

Voting functions

- update_notary_sample_size() -> bool

:

- Only the first valid transaction call of register_notary

, deregister_notary

, and add_header

can trigger this private function of each period. This function will update current_period_notary_sample_size

and notary_sample_size_updated_period

. See Committee Size Snapshot Mechanism

section for more details.

- Checks

: * Time to update

: self.notary_sample_size_updated_period < floor(block.number / PERIOD_LENGTH)

- Time to update

: self.notary_sample_size_updated_period < floor(block.number / PERIOD_LENGTH)

- Updates

: * self.current_period_notary_sample_size = self.next_period_notary_sample_size

- self.notary_sample_size_updated_period = current_period
- self.current_period_notary_sample_size = self.next_period_notary_sample_size
- self.notary_sample_size_updated_period = current_period
- Returns

True

if the storages got updated in this function; otherwise, returns False

.

- Vyper code# Update notary_sample_size def update_notary_sample_size() -> bool: current_period: int128 = floor(block.number / self.PERIOD_LENGTH) if self.notary_sample_size_updated_period >= current_period: return False

  self.current_period_notary_sample_size = self.next_period_notary_sample_size
  self.notary_sample_size_updated_period = current_period

  return True

- Only the first valid transaction call of register_notary

, deregister_notary

, and add_header

can trigger this private function of each period. This function will update current_period_notary_sample_size

and notary_sample_size_updated_period

. See Committee Size Snapshot Mechanism

section for more details.

- Checks

: * Time to update

: self.notary_sample_size_updated_period < floor(block.number / PERIOD_LENGTH)

- Time to update

: self.notary_sample_size_updated_period < floor(block.number / PERIOD_LENGTH)

- Updates

: * self.current_period_notary_sample_size = self.next_period_notary_sample_size

- self.notary_sample_size_updated_period = current_period
- self.current_period_notary_sample_size = self.next_period_notary_sample_size
- self.notary_sample_size_updated_period = current_period
- Returns

True

if the storages got updated in this function; otherwise, returns False

.

- Vyper code# Update notary_sample_size def update_notary_sample_size() -> bool: current_period: int128 = floor(block.number / self.PERIOD_LENGTH) if self.notary_sample_size_updated_period >= current_period: return False

  self.current_period_notary_sample_size = self.next_period_notary_sample_size
  self.notary_sample_size_updated_period = current_period

  return True

- get_member_of_committee(shard_id: int128, index: int128) -> address

:

- Steps

: * Getting sample size

: if self.notary_sample_size_updated_period < period

, sets sample_size = self.next_period_notary_sample_size

; else if self.notary_sample_size_updated_period == period

, sets sample_size = self.current_period_notary_sample_size

.

- Pseudo-randomly sampling

: uses the hash of the last block of the privious period ++ shard_id ++ index

as seed to sample sample_index

from sample_size

.

- Getting sample size

: if self.notary_sample_size_updated_period < period

, sets sample_size = self.next_period_notary_sample_size

; else if self.notary_sample_size_updated_period == period

, sets sample_size = self.current_period_notary_sample_size

.

- Pseudo-randomly sampling

: uses the hash of the last block of the privious period ++ shard_id ++ index

as seed to sample sample_index

from sample_size

.

- Returns

self.notary_pool[sampled_index]

- Steps

: * Getting sample size

: if self.notary_sample_size_updated_period < period

, sets sample_size = self.next_period_notary_sample_size

; else if self.notary_sample_size_updated_period == period

, sets sample_size = self.current_period_notary_sample_size

.

- Pseudo-randomly sampling

: uses the hash of the last block of the privious period ++ shard_id ++ index

as seed to sample sample_index

from sample_size

.

- Getting sample size

: if self.notary_sample_size_updated_period < period

, sets sample_size = self.next_period_notary_sample_size

; else if self.notary_sample_size_updated_period == period

, sets sample_size = self.current_period_notary_sample_size

.

- Pseudo-randomly sampling

: uses the hash of the last block of the privious period ++ shard_id ++ index

as seed to sample sample_index

from sample_size

.

- Returns

self.notary_pool[sampled_index]

Updated: reordering parameters again for (i) potentially combining shard_id

and period

in one bytes32 and (ii) matching on-chain storage format.

- submit_vote(int128 shard_id, int128 period, bytes32 chunk_root, int128 index) -> bool

: sampled notaries can call this function to submit vote. * Checks

:

- Authentication

: notary_registry[msg.sender]

exists

- Notary deposit

: notary_registry[msg.sender].deposit > 0

- Eligible proposal

: collation_records[shard_id][period].chunk_root == chunk_root

- Voter qualification

: checks the bit that represents index

is 0.

- Eligible notary

: msg.sender == get_member_of_committee(index)

- Authentication

: notary_registry[msg.sender]

exists

- Notary deposit

: notary_registry[msg.sender].deposit > 0

- Eligible proposal

: collation_records[shard_id][period].chunk_root == chunk_root

- Voter qualification

: checks the bit that represents index

is 0.

- Eligible notary

: msg.sender == get_member_of_committee(index)

- Updates

:

- Updates current_vote

with the bit that represents index

and increase the number of last byte (vote_count

) by 1.

- Checks if the vote_count == QUORUM_SIZE

: if True

, sets head_collation_period[shard_id] = period

and updates collation_records[shard_id][period].is_elected = True

- Updates current_vote

with the bit that represents index

and increase the number of last byte (vote_count

) by 1.

- Checks if the vote_count == QUORUM_SIZE

: if True

, sets head_collation_period[shard_id] = period

and updates collation_records[shard_id][period].is_elected = True

- Emits

Vote

log

shard_id bytes2 # pointer to shard period bytes3 # (current block number / PERIOD_LENGTH) chunk_root bytes32 # pointer to collation body notary_address address # the address of notary

- Returns

True

on success

- Checks

:

- Authentication

: notary_registry[msg.sender]

exists

- Notary deposit

: notary_registry[msg.sender].deposit > 0

- Eligible proposal

: collation_records[shard_id][period].chunk_root == chunk_root

- Voter qualification

: checks the bit that represents index

is 0.

- Eligible notary

: msg.sender == get_member_of_committee(index)

- Authentication

: notary_registry[msg.sender]

exists

- Notary deposit

: notary_registry[msg.sender].deposit > 0

- Eligible proposal

: collation_records[shard_id][period].chunk_root == chunk_root

- Voter qualification

: checks the bit that represents index

is 0.

- Eligible notary

: msg.sender == get_member_of_committee(index)

- Updates

:

- Updates current_vote

with the bit that represents index

and increase the number of last byte (vote_count

) by 1.

- Checks if the vote_count == QUORUM_SIZE

: if True

, sets head_collation_period[shard_id] = period

and updates collation_records[shard_id][period].is_elected = True

- Updates current_vote

with the bit that represents index

and increase the number of last byte (vote_count

) by 1.

- Checks if the vote_count == QUORUM_SIZE

: if True

, sets head_collation_period[shard_id] = period

and updates collation_records[shard_id][period].is_elected = True

- Emits

Vote

log

shard_id bytes2 # pointer to shard period bytes3 # (current block number / PERIOD_LENGTH) chunk_root bytes32 # pointer to collation body notary_address address # the address of notary

- Returns

True

on success

Per-period Sample Size Snapshot Mechanism

Goals

1. The sample_size

should stay the same during one period.

1. Any action that influences the “real” pool size would be recorded and then applied in the next period.

Description

- (i) We defined two storages current_period_notary_sample_size

(for short, current_s

) and next_period_notary_sample_size

(for short, next_s

)for recording “sample size of current period” and “sample size of next period” on-chain. Also a notary_sample_size_updated_period

(for short, update_p

) to log the last period where current_s

was updated.

- (ii) Only the first valid transaction call of register_notary

, deregister_notary

, and add_header

per period

can trigger update_notary_sample_size

function to update current_s

storages.

- (iii) For the notaries, they can check if they are sampled with sample_size

: if update_notary_sample_size

has already been called, use current_s

as sample_size

; otherwise, use next_s

. (The detailed detecting process is in Simple Proposing-Voting Procedure - Notary section

) After this process, the notaries will get a list of the positions that they should vote. Each item contains:

- shard_id

: the shard_id

of the sampled position

- period

: the period

of the sampled position

- index

: the index number of the sampled committee member position

- shard_id

: the shard_id

of the sampled position

- period

: the period

of the sampled position

- index

: the index number of the sampled committee member position

- (iv) The notary send submit_vote

transaction with (shard_id, period, chunk_root, index)

as the sampling seed. The logic of selecting sample_size

is the same as (iii).

FAQ

1. Is that possible that the same notary got sampled twice?

Yes, but we assume that the notary pool size would be more than 100 times of COMMITTEE_SIZE

.

1. Is that possible that an empty slot got sampled?

Yes, returns 0x0. We assume that normally, seldom notaries would deregister.

1. Is that possible that an out-of-range index?

Yes, returns 0x0.

1. Can a notary vote and deregister, and then another notary takes the empty slot?

Yes, but the second notary shouldn't be able to vote since we can check bitfield to know if this index has already voted.

Simple Proposing-Voting Procedure

Proposer

- Creates collation
- Signs collation
- Sends add_header

transaction to SMC

- Broadcasts the CollationBody

to shard network

Notary

- Maintains individual notary_index

which is generated in register

function.

- When a new period starts, check if they are sampled and in which shards, periods, and the committee indicesref Pseudo code:@to_list def get_sample_result(smc_handler, notary_index): web3 = smc_handler.web3 current_period = web3.eth.blockNumber // smc_handler.config['PERIOD_LENGTH']

  if smc_handler.notary_sample_size_updated_period() < current_period: sample_size = smc_handler.next_period_notary_sample_size() elif smc_handler.notary_sample_size_updated_period() == current_period: sample_size = smc_handler.current_period_notary_sample_size() else: raise Exception("notary_sample_size_updated_period is larger than current period")

  for shard_num in range(SHARD_COUNT): shard_id = to_shard_id(shard_num) bytes32_shard_id = int_to_bytes32(shard_id) entropy_block_number = current_period * smc_handler.config['PERIOD_LENGTH'] - 1 entropy_block_hash = web3.eth.getBlock(entropy_block_number)['hash'] for index in range(smc_handler.config['COMMITTEE_SIZE']): pool_index = big_endian_to_int(keccak(entropy_block_hash + bytes32_shard_id + int_to_bytes32(index))) % sample_size if pool_index == notary_index: yield (shard_id, period, index)

Selected notary

- Queries SMC.head_collation

and calculates the collation hash.

- Tries to download the CollationBody

and verifies the data availability of the collation. * If available, sends a submit_vote

transaction to SMC.

- If available, sends a submit_vote

transaction to SMC.

Shard full node

Full syncing

- Traces SMC.collation_records

to get the elected CollationHeaders

.

- Tries to download the CollationBodies

and verifies the data availability of the collations. * If available, accepts them as part of the canonical chain.

- If available, accepts them as part of the canonical chain.

After syncing

- Monitors the SMC, polls SMC.head_collation

and calculates the collation hash per period.

- Tries to download the CollationBody

and verifies the data availability of the collation. * If available, accepts it as part of the canonical chain.

- If available, accepts it as part of the canonical chain.

Sub-protocol

[TBD]

At least GetCollationBodies

and CollationBodies

messages for syncing.