Thanks to @timbeiko, @djrtwo, @gballet for discussions and comments.

This write up is the follow up to @djrtwo's Eth1+eth2 client relationship. Taking the separation of duties outlined in the previous document as the foundation, it aims to define the scope of work required to deliver the merger.

Alongside with the scope, it provides thoughts regarding the implementation of particular components of eth1-engine and eth2-client in the Eth1 shard context. Depending on the section, these thoughts have a different level of detail. This information intends to be a starting point for further discussions around specifications and implementation.

NOTE:

Eth1x outcomes are not strongly required for the technical side of the merger. Despite of this, prerequisites include Eth1x to highlight how substantial it efforts are for validators adoption and user experience after the merger.

Consensus

[

1656×1262 190 KB

](https://ethresear.ch/uploads/default/original/2X/f/f30e64ca36592aa093a4b59cc77e231eaf463a34.png)

- Prerequisites
- Phase 1
- State transition
- Fork choice
- Validator
- State transition
- Fork choice
- Validator
- Eth1x
- Stateless execution
- Stateless execution
- Phase 1
- State transition
- Fork choice
- Validator
- State transition
- Fork choice
- Validator
- Eth1x
- Stateless execution
- Stateless execution
- Scope
- Eth1 engine
- Block processing
- Block production

- External fork choice
- Block processing
- Block production
- External fork choice
- Eth1 shard
- State transition function
- Validator duties
- State transition function
- Validator duties
- Beacon chain
- Eth1 shard as a source of deposits
- Validator capabilities
- Eth1 shard committee selection
- Eth1 shard as a source of deposits
- Validator capabilities
- Eth1 shard committee selection
- Eth1 rewards
- Eth1 engine
- Block processing
- Block production
- External fork choice
- Block processing
- Block production
- External fork choice
- Eth1 shard
- State transition function
- Validator duties
- State transition function
- Validator duties
- Beacon chain
- Eth1 shard as a source of deposits
- Validator capabilities
- Eth1 shard committee selection
- Eth1 shard as a source of deposits
- Validator capabilities
- Eth1 shard committee selection
- Eth1 rewards

Eth1 engine

To become compatible to Eth1 shard, eth1-engine will have to expose block processing and block production via its API. Currently, eth_getWork endpoint produces a block but doesn’t return its full structure, and block processing seems trivial to be exposed as well (see InsertChain). Also, a scope of work on these two includes getting rid of Ethash and managing the concurrency case when block production depends on the processing.

The fork choice of Eth1 shard will be entirely up to eth2-client. Since forehanded knowledge of the head of the chain is essential for the transaction pool optimizations, eth1-engine must be aware of reorgs that occurred on the eth2-client side. There could be a setHead

API endpoint leveraging SetHead function to expose the fork choice. This change also implies reorgs to be removed from the block processing flow of eth1-engine.

Eth1 shard

Eth1 shard state transition will invoke eth1-engine to process a block and verify the result. To create a block, Eth1 shard block producers will have to call eth1-engine as well. Attesters assigned to Eth1 shard will have to execute Eth1 block and verify the result of the execution. These extensions to the shard transition are subjects for Eth1 shard specification.

Considering potential usage, a body of Eth1 shard block should at least contain the following data:

- Eth1-styled block hash.

It will likely be involved in block production to point out eth1-engine to the parent block.

- Eth1 state root.

With this data building proofs against Eth1 state becomes possible, making various use cases crossing eth1-eth2 border turn into reality.

- Transaction receipts.

The other way to establish eth1-eth2 communication is by utilizing the receipts.

- RLP encoded Eth1 block.

Primary payload of a block of Eth1 shard.

ShardState object will contain a hash tree root of this data in the data

field supplying beacon chain with all of Eth1 block essentials trough merkle proofs.

Note:

exposing transaction receipts turns into additional work for Eth1 shard block proposer. It will have to parse RLP-encoded receipts returned by eth1-engine and publish them in a format suitable for beacon chain.

Beacon chain

Eth1 data voting mechanism currently used by Phase 0 may get significant improvement after the merger. Moreover, it can become the first use case leveraging the new consensus. When Eth1 shard kicks off deposit root can get updated as frequent as cross-linking of Eth1 state root happens. Supplemented by the merkle proof deposit root becomes verifiable by any beacon chain client allowing for instant inclusion of new deposits. Note that to verify merkle proofs created against Eth1 state root or, e.g., polynomial commitments that could come instead, beacon client will have to be aware of the verification algorithm. For example, if beacon client had to verify Eth1 state root at the current moment, it would have to support Merkle Patricia Trie and keccak256

hash function.

There is a more convenient way of deposit processing that utilizes Eth1 transaction receipts. Beacon chain proposers will be able to parse receipts corresponding to the DepositEvent and induce new deposits upon cross-linking of Eth1 shard.

According to the current state of the arts, validator attesting to or producing on Eth1 shard will have to maintain full Eth1 state. Alleviating this requirement is one of the objectives of Eth1x research efforts. Once stateless execution is in place, attesters will become free from maintaining the state but not the producers. This kind of separation creates a demand for validator capability management. It could be implemented by adding capability flags as yet another field to the validator record. Such change implies a new operation on the beacon chain to update capability flags. The selection of Eth1 shard committee will have to keep track of validators capability, making committee selection yet another subject for Eth1 shard specification.

Eth1 rewards

After the merger takes place and PoW does no longer exist on the mainnet, there will be significant issuance cut by getting rid of block and uncle rewards.

Shard block proposer rewards in Phase 1 are described inthis issue and consist of the following parts:

- block proposer reward
- EIP-1559 fee per byte in the block data
- transaction fees

Beacon chain state transition is responsible for processing the first and the second steps, i.e., the core protocol pays the Eth1 block proposer for the proposal and subtracts the EIP-1559 fee from the proposer's balance. Transaction fees charging (if there is) assumed to be out of the core protocol.

In Phase 2, when a shard block becomes executable, gas used by a block is going to replace the size in the scheme above. It retains the EIP-1559 logic in the core protocol. Following the less invasiveness strategy of the merger, we might want to keep eth1-engine responsible for maintaining the EIP-1559 fees with a follow-up changes in Phase 2. In this case, the gas limit calculation could be moved to the core protocol and explicitly passed to eth1-engine during the block creation process.

Eth1 shard proposers could simply utilize the coinbase for the transaction fee payments. Another option would be to include transaction fees in a shard block explicitly and make the core protocol responsible for depositing these fees to the validator account. This trade-off between a validator maintaining several identities and the core protocol complication is yet to be resolved.

Yet another expenditure that Eth1 shard validators will have to take into account is a storage fee required to maintain Eth1 state. Fortunately, if Eth1-capable validator is not currently participating in the Eth1 shard committee, it can drop the state, diminishing the storage fees and sync it back when needed.

Network

[

1656×1302 154 KB

](https://ethresear.ch/uploads/default/original/2X/8/8f56b484b225ca1314c2bd481b90fd81f76707b1.png)

- Prerequisites
- Phase 1
- Subnet topics
- Shard chain sync
- Subnet topics
- Shard chain sync
- Eth1x
- State sync
- History data storage
- State sync
- History data storage
- Phase 1
- Subnet topics
- Shard chain sync
- Subnet topics
- Shard chain sync

- Eth1x
- State sync
- History data storage
- State sync
- History data storage
- Scope
- Discovery service
- Network reputation
- Eth1 shard sync
- Discovery service
- Network reputation
- Eth1 shard sync

Discovery service

In eth1+eth2 client relationship, Danny reasonably proposes to keep p2p interfaces of eth2-client and eth1-engine separately. These pair of interfaces should operate under the same network identity. Otherwise, it would be difficult to justify the reputation of peers constituting Eth1 shard network.

Working under a single identity implies the sharing of discovery service between eth2-client and eth1-engine. It would make sense if eth2-client took responsibility for network discovery, providing discovery service to eth1-engine. This approach to sharing discovery duties likely makes communication in the client-engine pair to be bi-directional

because of the pattern widely used in eth1 clients to request discovered peers on demand.

The other approach to discovery service of client-engine pair would be to retain eth1-engine discovery and bootstrap it with the node id used by eth2-client. This way is less invasive to eth1-engine codebase and still allows for maintaining a shared network identity without requiring client-engine communication to be bi-bidirectional

.

Network reputation

Operating under the same network id client-engine pair becomes a single subject of the reputation. However, managing the reputation of the other peers on the network requires additional communication between eth2-client and eth1-engine cause they still share network responsibilities like transaction gossiping and state sync by eth1-engine against block gossiping by eth2-client. For instance, both parts of the pair will have to notify each other about bad behavior observed on their side of the network, resulting in disconnects and further bans of malicious peers.

Network specifications do not cover reputation heuristic, making it implementation-specific. Thus, it might differ not only between eth1 and eth2 clients but also across various eth2 client implementations. This discrepancy leaves a small room for the heuristic of shared reputation management that would probably be stripped down to disconnect signals only. This part of the protocol does also require bi-directional communication in the client-engine pair.

Eth1 shard sync

eth1-engine is going to be responsible for the state sync with either the currently existing algorithm or whatever else solution that Eth1x will come up. But the orchestration of the sync process should be done by eth2-client that will initially browse the network to find the most recent finalized checkpoint and send a command to eth1-engine to start syncing with a particular state root.

The other part of what is currently called Fast Sync in Eth1 is linking the downloaded state to the genesis block by Ethash-valid chain, verifying that the state belongs to the canonical chain. In Eth2 world, this kind of verification will look different due to the weak subjectivity. Thus, eth2-client will finalize Eth1 shard sync by checking that the state is canonical. It’s highly likely both parts of the sync process could be done simultaneously.

The other responsibility worth noting in this section is the access to Eth1 history data. In general, ancient data will be entirely handled by eth1-engine. Upon the finishing of the sync process, the eth2-client may send command to eth1-engine to obtain the whole chain of blocks and receipts. However, dealing with the history represented by the chain of blocks and their receipts is kind of burdensome. One of the objectives of Eth1x research is to improve history data management, which could significantly impact the UX of Eth1 shard.

Old fashioned regular block sync could still be utilized for quick state catch-ups after relatively short periods of downtime. However, it would require additional information obtained from Eth1 shard network like the number of Eth1 block corresponding to the head of the chain. If this number would be too far from the current client's state, then it worth falling back to the state sync or use other techniques probably borrowed from the beam sync. Anyway, from eth2-client perspective, it would be a single command to sync with the particular state root, and the heuristic of that sync should be entirely handled by eth1-engine.

Client

[

1656×1012 43 KB

](https://ethresear.ch/uploads/default/original/2X/7/7b61a0f2a195382bdd99e808c3c8eb61f1f425ac.png)

- Scope
- Code updates
- Getting rid of PoW
- validity conditions
- block production
- fork choice
- validity conditions
- block production
- fork choice
- Network stack
- discovery
- reputation
- discovery
- reputation
- State trie pruning
- Getting rid of PoW
- validity conditions
- block production
- fork choice
- validity conditions
- block production
- fork choice
- Network stack
- discovery
- reputation
- discovery
- reputation

- State trie pruning
- Eth1-Eth2 communication protocol
- Consensus
- set head
- create block
- insert block
- finalize block
- set head
- create block
- insert block
- finalize block
- Network
- discovery
- reputation
- sync
- discovery
- reputation
- sync
- Consensus
- set head
- create block
- insert block
- finalize block
- set head
- create block
- insert block
- finalize block
- Network
- discovery
- reputation
- sync
- discovery
- reputation
- sync
- UX
- API facade
- Light client

- Client tooling
- API facade
- Light client
- Client tooling
- Code updates
- Getting rid of PoW
- validity conditions
- block production
- fork choice
- validity conditions
- block production
- fork choice
- Network stack
- discovery
- reputation
- discovery
- reputation
- State trie pruning
- Getting rid of PoW
- validity conditions
- block production
- fork choice
- validity conditions
- block production
- fork choice
- Network stack
- discovery
- reputation
- discovery
- reputation
- State trie pruning
- Eth1-Eth2 communication protocol
- Consensus
- set head
- create block
- insert block
- finalize block

- set head
- create block
- insert block
- finalize block
- Network
- discovery
- reputation
- sync
- discovery
- reputation
- sync
- Consensus
- set head
- create block
- insert block
- finalize block
- set head
- create block
- insert block
- finalize block
- Network
- discovery
- reputation
- sync
- discovery
- reputation
- sync
- UX
- API facade
- Light client
- Client tooling
- API facade
- Light client
- Client tooling

State trie pruning

Cleaning up outdated versions of the state trie is essential to keep Eth1 state of a sane size during the whole lifetime of the client run. Currently, this mechanism uses follow distance to decide which state versions are outdated and should undergo the pruning process. The distance parameter should be big enough to secure the client from accidental reorg requiring to proceed from the old version of the state that has been pruned already, for example, the geth client uses 128

blocks as a default follow distance on the mainnet.

For the Eth1 shard, more organic trigger for trie pruning would be checkpoint finalization. Once eth2-client gets the new finalized checkpoints, it could call eth1-engine with the hash of finalized Eth1 block and thereby trigger the process pruning the state trie up to that block.

Tooling

From a user perspective switching from Eth1 to Eth1 shard would mean maintaining one more client and setting up the connection between two parts of the new-fashioned client. It sounds like a significant UX flaw. Client tooling could be a remedy from this. This kind of tooling should provide support for all major eth1-engines and eth2-clients, be able to download and setup them, and establish a communication channel of the client-engine pair.

The other part of UX work would be in designing a unified API facade providing access to API interfaces of both eth2-client and eth1-engine. This facade should expose Eth1 JSON-RPC with as less changes as it could be achieved considering a major shift in consensus and updates of data structures.

Light client

The beacon state is very tiny and fits up to a hundred of megabytes. However, the network layer of the beacon node could be pretty much intensive due to attestation dissemination happening each slot.

Using a beacon chain light client instead of a full node would save a lot of network traffic to the user of Eth1 shard. A size of the state required by the light client is less than a megabyte and could be kept in memory all the way. When eth2-light-client starts, it downloads the most recent light client state within a single roundtrip and gives a command to eth1-engine to catch up with the most recent finalized Eth1 state. In the online mode, the light client would need to listen to beacon and shard blocks and execute Eth1 blocks to keep the tip of Eth1 shard chain with a pretty high level of security.

Note:

light client is highly desirable but not a prerequisite to the merger. There is another way of reducing beacon chain traffic by turning on a light mode for the full beacon chain client. In this mode, the client will listen to the beacon blocks channel without participating in attestation subnets. This approach retains the security of individual clients but could affect the network itself; hence is a subject for further investigation.

The next step

We propose a core consensus implementation as the next step towards the merger. A scope of the PoC doesn't include network and major client-related work. It minimizes consensus requirements to the following subset:

- Prerequisites
- Phase 1
- State transition
- Fork choice
- Validator
- State transition
- Fork choice
- Validator
- Phase 1
- State transition
- Fork choice
- Validator
- State transition
- Fork choice

- Validator
- Scope
- Eth1 engine
- Block processing
- Block production
- External fork choice
- Block processing
- Block production
- External fork choice
- Eth1 shard
- State transition function
- Validator duties
- State transition function
- Validator duties
- Eth1-Eth2 communication protocol
- Consensus
- Consensus
- Eth1 engine
- Block processing
- Block production
- External fork choice
- Block processing
- Block production
- External fork choice
- Eth1 shard
- State transition function
- Validator duties
- State transition function
- Validator duties
- Eth1-Eth2 communication protocol
- Consensus
- Consensus

The communication protocol is going to be as minimal as required by the core consensus functionality. Any beacon chain changes, including new deposit processing, are out of the scope, client updates are stripped down to the minimum subset needed for delivery.

A product of the PoC should be a client-engine pair capable of producing and importing shard blocks with Eth1 blocks as executable payload.