Beacon Chain Light Client Classification

A beacon chain light client allows access to Ethereum on resource-constrained environment, e.g. phones, metered VM, or browsers. A light-client should be defined with respect to the set of APIs that it could handle.

I could not find any posts or spec documents discussing the full scope of functionality that the beacon-chain light client is supposed to cover. Instead of narrowing down a single scope, I am listing out a reasonable progression of the functionalities that a light-client could. It ranges from the minimal client to a client resembling a full node.

For each of the level of functional completeness, I also sketch out what kind input data is required for the client to compute the answers of the API queries.

Level 1: Minimum

The minimum beacon chain light client syncs to the latest header using sync_committee signatures. The client can only answer questions with the information contained in the BeaconBlockHeader.

Supported APIs

- /eth/v1/beacon/headers
- /eth/v1/beacon/states/{state_id}/root

Input Data

The client needs to have access the following two message types specified in the consensus-spec aboutminimal light client.

LightClientSnapshot LightClientUpdate

Level 2: Access Beacon State

The client is able to answer most of the questions with regard to theBeaconState.

Supported APIs

The following list not exhaustive, but it is representative of the set of functions that this client intends to support.

/eth/v1/beacon/genesis /eth/v1/beacon/states/{state_id}/root /eth/v1/beacon/states/{state_id}/finality_checkpoints /eth/v1/beacon/states/{state_id}/validators /eth/v1/beacon/states/{state_id}/validators/{validator_id} /eth/v1/beacon/states/{state_id}/validator_balances /eth/v1/beacon/states/{state_id}/committees /eth/v1/beacon/states/{state_id}/sync_committees

Input Data

- Type 1: Updating to latest header

LightClientSnapshot LightClientUpdate

- Type 2: Answering query on BeaconState

- It is not likely that the client wants to keep an up-to-date copy of BeaconState locally. It only keeps track of the state root.

- To answer a query, the client needs access to the corresponding leave node of the beacon state. It gets the leave nodes and multiproof from a network layer.

- The client validates the proof and use the data to compute an answer.

class BeaconStateProof(Container): leaves: Sequence[Bytes32], leave_indices: Sequence[Bytes32], proof: Sequence[Bytes32],

- It is not likely that the client wants to keep an up-to-date copy of BeaconState locally. It only keeps track of the state root.

- To answer a query, the client needs access to the corresponding leave node of the beacon state. It gets the leave nodes and multiproof from a network layer.

- The client validates the proof and use the data to compute an answer.

Level 3: Support Attestation Client

The beacon chain client is able to support a validator that is only performing attestation tasks.

Attestation makes up the bulk of the workload for a validator client. If a “attestation client” could be supported by a light weight beacon chain client, it could open the door to allow a majority of the validation tasks to be run on resources constrained devices. The rest of the validation tasks, e.g. block proposing and sync committee signing, could be backed by a smaller set of full nodes with higher resources requirements.

Supported APIs

The client needs to support all the queries required to complete the attestation duties.

/eth/v1/beacon/states/{state_id}/validators/{validator_id} /eth/v1/validator/duties/attester/{epoch} /eth/v1/validator /attestation_data

Input Data

- Type 1: Updating to latest header

LightClientSnapshot LightClientUpdate

- Type 2: Accessing BeaconState
- The client needs to calculate committee assignment.
- The client needs to access the beacon state’s random mix and validator information.

BeaconStateProof

- The client needs to calculate committee assignment.
- The client needs to access the beacon state’s random mix and validator information.
- Type 3: Get full blocks
- Need to get the full blocks
- The client does not need to know every blocks. It only needs to know the blocks associated with the attestation client’s assigned slots.

BeaconBlock

- Need to get the full blocks
- The client does not need to know every blocks. It only needs to know the blocks associated with the attestation client’s assigned slots.

Level 4: Limiting Case: Approaching Full Node

It would be ideal if a light weight beacon chain client could support all the API needs of a validator client. Furthermore, the same light weight client is also able to answer queries with regard to the current state of transient gossiped data. However, this view might not be realistic as some of these functionalities will fundamentally require the client to process large amount data.

These functionalities are not likely to be supported by light weight clients. However, it is still instructive to list out these classifications of functionalities to understand where should be the design boundary of light weight clients.

Supported APIs

1. Support validator client that are performing attestation aggregations, sync-committee duties, and block proposing.
2. Support a slasher.
3. Be able to answer queries about beacon-state as well as providing summarizing statistics about transient data. That is, the client would have to maintain up-to-date pools for all the gossip messages such as sync committee signatures, attestation, slashings, voluntary exists, etc.