Beacon Chain Light Client Networking

A beacon chain light client is classified by what APIs it could expose. See the conversation onclassification for more details. Different sets of APIs require the client to have access to a different sets of input data.

The source of the input data is a networking layer. The networking layer could be classified into two varieties. It could be in the form of a server-client, request-response model. The client knows about some servers who are willing to respond to its requests. This form of networking design is similar in spirit to the design of LES.

The second variety of networking is a peer to peer (p2p) network. There already exists p2p networks that could supply light client with the requisite input data. However, these networks are designed for full nodes. Most of these networks has data volumes that are too high to be appropriate data sources for light clients. The alternative to these beacon chain p2p channels are the networks that is part of the so-called portal network. Portal network is a work-in-progress effort to create p2p networks for resource constrained clients. Portal network provides a basic framework to design p2p networks for specific data types. These p2p networks do not exist yet.

For each set of input data from theclassification, I am going to look more closely into how different data sets could be implemented in different networking setups.

Level 1: Minimum

Server-Client Model

The server responds to the following requests

Latest snapshot

GET /les/beacon-chain/LightClientSnapshot

Latest update

GET /les/beacon-chain/LightClientUpdate

P2P Portal Network

We create light weight gossip channels on the portal network. These gossip channels have low traffic volume. Participants could further restrict the percentage of messages that they receive based on their resources availability. Below are the topics and wire format mapping.

- gossip network: bc-light-client-snapshot
- Data type: LightClientSnapshot
- A message is created every epoch
- A client do not have to keep this connection alive after the initial sync
- Data type: LightClientSnapshot
- A message is created every epoch
- A client do not have to keep this connection alive after the initial sync
- gossip network: bc-light-client-update
- Data type: LightClientUpdate
- A message is created every slot
- A client could skip some messages, and skip sync headers.
- Data type: LightClientUpdate
- A message is created every slot
- A client could skip some messages, and skip sync headers.

Beacon Chain P2P Network

- It might look a bit strange to consider using the gossip topics intended for full nodes as input sources for light clients. It is worth a consideration because these beacon-chain gossip topics do not have high volume data. If a client limits its number of peer connections, it is possible that the resource requirement is sufficiently low. That being said, this is listed out mostly for completeness sake. The other two data sources should give better results.
- beacon-chain topic: sync_committee_contribution_and_proof
- This topic is not high volume
- These messages contain sufficient data to update from one block header to the next
- This topic is not high volume
- These messages contain sufficient data to update from one block header to the next
- beacon-chain topic: beacon_block
- The number of messages is not high, about 1 per slot under optimal conditions.
- The messages are relatively large. Each message is a full block.
- The number of messages is not high, about 1 per slot under optimal conditions.
- The messages are relatively large. Each message is a full block.
- The beacon_block

topic let the light-client knows which header to update to, and the information in ``sync_committee_contribution_and_proofallows the light-client to extract sufficient up-to-date sync-committee signature to validates the update to the any header. It could also be argued that the light-client only needs to listen to

beacon_blockin order to advance the header because the full block also contains sync-aggregate messages. However, those sync-aggregate data only points to the parent root. The light-client would not be able to skip sync with only the information contained in the

beacon_block` topic.

Related links

All of the following links implicitly assume that the minimal light client implementation will use a server-client approach operating on the message types LightClientSnapshot and LightClientUpdate. It is worth noting that it is not the only way to get to a minimal light client.

- Minimal light client in theconsensus-spec
- prysm light-client prototype
- Lodestar's prototype
- Consensus Light Client Server Implementation Notes - HackMD
- https://github.com/ethereum/consensus-specs/issues/2429
- Consensus Light Client Server Implementation Notes - HackMD
- https://github.com/ethereum/consensus-specs/issues/2429
- Alex Stoke's writeup on how to bootstrap the light client ecosystem

Level 2: Access Beacon State

Server-Client

Latest snapshot

GET /les/beacon-chain/LightClientSnapshot

Latest update

GET /les/beacon-chain/LightClientUpdate

Request specific node in the beacon state

GET /les/beacon-chain/BeaconStateProof?root=&leaves_indices=

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

It is worth noting that it is very likely that for each beacon state request to the light client, the client would have to make a corresponding request to the server because the light client has a limited knowledge of the beacon state.

P2P Portal Network

- Type 1: Updating headers
- gossip network: bc-light-client-snapshot
- Data type: LightClientSnapshot
- Data type: LightClientSnapshot
- gossip network: bc-light-client-update
- Data type: LightClientUpdate
- Data type: LightClientUpdate
- gossip network: bc-light-client-snapshot
- Data type: LightClientSnapshot
- Data type: LightClientSnapshot
- gossip network: bc-light-client-update
- Data type: LightClientUpdate
- Data type: LightClientUpdate
- Type 2: Beacon State
- dht network: beacon-state
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- The node stores a small subtree of the state.
- If the participant could choose the node_id

, the participant could selectively store the subtree that the client is most interested in.

- Queries that fall outside of the subtree requires the client to make DHT requests to the p2p network
- dht network: beacon-state
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- content_id: sha256(root | leaves_general_indices)

- content: root, leaves_general_indices, leaves, proof
- The node stores a small subtree of the state.
- If the participant could choose the node_id

, the participant could selectively store the subtree that the client is most interested in.

- Queries that fall outside of the subtree requires the client to make DHT requests to the p2p network

Level 3: Support Attestation Client

Server-Client

Type 1: Updating headers

GET /les/beacon-chain/LightClientSnapshot GET /les/beacon-chain/LightClientUpdate

Type 2: Beacon State

This data is used to calculate committee assignments

GET /les/beacon-chain/BeaconStateProof?root=&leaves_indices=

Type 3: Full block

Only need to get the blocks that the attestor is assigned to

GET /les/beacon-chain/BeaconBlock?slot=

P2P Portal Network

- Type 1: Updating headers
- gossip network: bc-light-client-snapshot
- Data type: LightClientSnapshot
- Data type: LightClientSnapshot
- gossip network: bc-light-client-update
- Data type: LightClientUpdate
- Data type: LightClientUpdate
- gossip network: bc-light-client-snapshot
- Data type: LightClientSnapshot
- Data type: LightClientSnapshot
- gossip network: bc-light-client-update
- Data type: LightClientUpdate
- Data type: LightClientUpdate
- Type 2: Beacon State
- dht network: beacon-state
- content_id: sha256(root | leaves_general_indices)

- content: root, leaves_general_indices, leaves, proof
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- dht network: beacon-state
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- content_id: sha256(root | leaves_general_indices)
- content: root, leaves_general_indices, leaves, proof
- Type 3: Full blocks
- gossip network: bc-light-client-block-
- Data type: BeaconBlock
- Data type: BeaconBlock
- It has to be a gossip network because the attestors do not know the block root a priori.
- The gossip channel could be a per-slot subnet to limit resource usage.
- gossip network: bc-light-client-block-
- Data type: BeaconBlock
- Data type: BeaconBlock
- It has to be a gossip network because the attestors do not know the block root a priori.
- The gossip channel could be a per-slot subnet to limit resource usage.