Why does Eth2 need light clients?

Ethereum consists of thousands of nodes running various clients. Clients differ both in terms of the language they are implemented in and their size. Running a full client on Eth2 means that you have to download the entire Eth2 blockchain. This is prohibitive for many environments and unnecessary for certain use cases. For example, a full node cannot be run on a mobile device hosting a trust minimized wallet. There are also many use cases that simply want to read some small piece of the chain, for example, fetching a recent validator balance or hosting a metered VM for a blockchain bridge instance such as [ChainBridge](https://github.com/ChainSafe/ChainBridge" \t "_blank). In these instances, a light client is sufficient and preferable to a full client.

In order to retrieve the latest state, Ethereum full nodes must download the genesis (or other) beacon state, and download and sequentially process every canonical block from start to finish. Using light clients, this bandwidth & processing load can be reduced by ~99%.

How does the Eth2 protocol support light clients?

The recent [Altair](https://github.com/ethereum/eth2.0-specs/releases) upgrade includes a key feature specifically designed to support light client syncing — the sync committee. The sync committee and its associated infrastructure lets a light client stay updated to the head of the chain very cheaply.

Broadly speaking, a light client has two main components: First, syncing, and then chain data retrieval.

Syncing is as it sounds: it facilitates the client to sync to a recent part of the chain. In order to stay synced to the chain, a light client periodically downloads just some of the chain’s data in the form of an update, which includes a block header, some merkle proofs, and signatures from the current sync committee. The proofs and signatures are verified, which allows for trust to be built from one update to another. Once synced, the light client is then able to retrieve recent data from the chain.

Diagram

Description automatically generated with medium confidence

Diagram

Description automatically generated

Chain data retrieval is provided by extensive use of Merkle proofs. Once the recent block header is verified and trusted, a light client server can provide chain data in the form of Merkle proofs to a light client. These proofs can be generated and verified against the block header’s state root.

Because the sync protocol plays such a central role for light clients, it is worth going into more detail on how it functions:

* Altair creates the sync committee, a special long-lived committee that rotates approximately every 1.1 days. Shorter periods would increase data load on light clients as they would need to sync more frequently, and longer periods would leave open too much opportunity to discover and corrupt committee members; 1.1 days was chosen as the happy medium that fares reasonably well on both dimensions ([ref](https://github.com/ethereum/annotated-spec/blob/master/altair/beacon-chain.md)).
* The sync committee is a subset chosen from existing Ethereum validators. Participating in sync committees will be an additional responsibility of validators once Altair is launched!
* At the time of writing, the sync committee would be a set of 512 validators out of the ~138,000 currently on mainnet. It’s a relatively large and conservative size (compared to attestation and later shard proposal committees) to ensure safety ([ref](https://github.com/ethereum/annotated-spec/blob/master/altair/beacon-chain.md)).
* The sync committee allows light clients to more easily sync and stay synced to the beacon chain.
* The beacon state tracks the current AND next sync committee participants. We will see why this is important below.

Sync committee participants \**attest*\* to the current state of the chain, specifically the previous block, and these attestations are aggregated into a single signature which is included in every new block.

* Now, every block contains a signature verifying its predecessor.
* This special sync committee attestation is named “SyncAggregate” (to differentiate it from “Attestations” put on chain via beacon committees)

Recap:

* (Relatively) small subset of validators attest to block N-1 in every block N
* (Relatively) small subset of validators only rotates every ~27 hrs
* (Relatively) small subset of validators (current and next) is included on chain

The light client sync protocol can now be crafted. It relies on the sync committee’s agreement on the state of the chain

* Instead of initializing the node with the entire beacon state (eg: the genesis state), the light client downloads a historical block header and the current and next sync committee at that block.
* Instead of downloading and tracking the entire validator set (200k validators and counting!), a light client can just download and track the current and next sync committee.
* Instead of progressing the chain by downloading and processing every historical block sequentially, a light client can just download the last SyncAggregate from a sync committee (ensuring that ⅔ of the committee signed) and a merkle proof to the next sync committee

In essence, the light client syncs by tracking each sync committee across time, syncing forward committee by committee, ensuring that each committee handoff has a ⅔ vote that verifies the next.

Diagram

Description automatically generated

Diagram

Description automatically generated

How did we implement it?

During the [Scaling Ethereum Hackathon](https://scaling.ethglobal.co/), the Lodestar team decided to make a big push on our light client work to try and present a prototype and a demo by the end of the event. The light client prototype connected to a backend that advances the states using blocks created without validator consensus. This allowed us to test the consumer layer with a low resource cost.

After the hackathon, we migrated all functionality to our beacon node so the light client can consume real data from networks. The beacon node now prepares updates on initial proofs efficiently as the chain progresses and serves them through the new ‘lightclient’ API namespace. We’ve been able to connect to Altair devnets 0, 1 and 2 and consume data successfully.

For the purpose of this hackathon, and to simplify a demo, we created a REST API for the light client to fetch updates and request proofs. There are three endpoints involved in this process which:

* Get historical sync updates
* Get latest sync update
* Get multi-proofs for a beacon state

These endpoints expose underlying functionality which create sync update objects from beacon states, as well as multi-proofs from beacon states. We will seek to build consensus around this REST API to become the new standard for light clients in Eth2.

The light client was built to be initialized either from a trusted state root, on an initial startup of a light client, or from a trusted snapshot after the light client has been started. Once the light client is initialized, it requests updates from its current synced position up to the finalized state.

The light client also exposes functionality to request proofs, verified against currently synced state root.

Diagram

Description automatically generated

Diagram

Description automatically generated

What does our website show?

Our [website](https://light-client-demo.lodestar.casa/) shows bare-bones functionality of a light client running in the browser. The browser light client is connected to a beacon node running in a small multi-client Altair testnet. We show the network clock, the sync status, and a proof request/response interactive section. Check out our [demo](https://youtu.be/3O7nNHhdqlc) for more information!

The network clock shows the current slot / epoch of the testnet. This section is purely informational. It does not currently show any light client functionality.

The sync status section shows the current sync status of the light client. The light client is configured to update to the latest finalized state. On page load, you may notice that the sync will quickly progress to the current slot.

The proof request/response section allows users to execute proof requests and receive responses interactively. The proof request is described as a list of “paths” into the beacon state. The response is shown as both the deserialized result and the underlying proof.

A unique feature to note is that each request for multiple “paths” into the state returns a single multiproof, rather than one proof per path. This batching of results allows for additional bandwidth savings.

Next Steps

We are very excited about all the progress we have made on Eth2 light clients, but we are just getting started. One of our next steps will be to take what we learned while building our prototype and report it to the Eth2 spec repo. See are first issue [here](https://github.com/ethereum/eth2.0-specs/issues/2429).

Moving forward, we will be sure to stay up to date with spec updates. We’re currently on 1.1.0-beta-2. This will ensure that our Beacon node and light client stay up to date and on pace with the rest of the ecosystem. We will also continue to participate in Altair devnet / testnets and keep an eye on the Pyrmont testnet as it forks to Altair in the coming weeks.

Another goal for the Lodestar team is to create a reorg capable light client which can sync to the head of the chain. This is because the light client currently only syncs to the latest finalized checkpoint, ~6 minutes behind the tip of the chain in a healthy network. In order to support syncing to the head of the chain, we need to support “reorgs” that may naturally happen during network turbulence. Finally, we plan to do ongoing research and explore ways to extend consumable user data from consensus layer to execution layer, post [merge](https://github.com/ethereum/eth2.0-specs/tree/dev/specs/merge). The current light client only consumes data that’s in the beacon chain. This includes validator balances, randao values, and past beacon blocks. But critically, it does not include execution related data (previously called Eth1), such as ETH balances or Dapp state.

The merge ties the execution layer to the consensus layer (previously called Eth2). This forms the basis for using the light client to consume execution layer data, but lots of practical questions still remain.

We are excited to continue working on Lodestar’s Eth2 light client and look forward to providing future updates!