title: Light clients description: Introduction to Ethereum light clients. lang: en

Running a full node is the most trustless, private, decentralized and censorship resistant way to interact with Ethereum. With a full node you keep your own copy of the blockchain that you can query instantly and you get direct access to Ethereum's peer-to-peer network. However, running a full node requires a significant amount of memory, storage and CPU. This means it is not feasible for everyone to run their own node. There are several solutions to this on the Ethereum roadmap, including statelessness, but they are several years away from being implemented. The answer in the near-term is to trade-off some of the benefits of running a full node for large performance improvements that allow nodes to run with very low hardware requirements. Nodes that make this trade-off are known as light nodes.

What is a light client {#what-is-a-light-client}

A light node is a node running light client software. Instead of keeping local copies of the blockchain data and independently verifying all the changes, they request the necessary data from some provider instead. The provider might be a direct connection to a full node or via some centralized RPC server. The data is then verified by the light node, allowing it to keep up with the head of the chain. The light node only processes block headers, only occasionally downloading the actual block contents. Nodes can vary in their lightness, depending upon whether the combinations of light and full client software they run. For example, the lightest configuration would be to run a light execution client and a light consensus client. It is also likely that many nodes will choose to run light consensus clients with full execution clients, or vice versa.

How do light clients work? {#how-do-light-clients-work}

When Ethereum started using a proof-of-stake based consensus mechanism, new infrastructure was introduced specifically to support light clients. The way it works is by randomly selecting a subset of 512 validators every 1.1 days to act as a **sync committee**. The sync committee signs the header of recent blocks. Each block header contains the aggregated signature of the validators in the sync committee and a "bitfield" that shows which validators signed and which did not. Each header also includes a list of validators expected to participate in signing the next block. This means a light client can quickly see that the sync committee has signed off on the data they receive, and they can also check that the sync committee is the genuine one by comparing the one they receive from the one they were told to expect in the previous block. In this way, the light client can keep updating its knowledge of the latest Ethereum block without actually downloading the block itself, just the header which contains summary information.

On the execution layer there is no single specification for a light execution client. The scope of a light execution client can vary from a "light mode" of a full execution client that has all the EVM and networking functionality of a full node but only verifies block headers, without downloading the associated data, or it can be a more stripped down client that relies heavily upon forwarding requests to an RPC provider to interact with Ethereum.

Why are light clients important? {#why-are-light-clients-important}

Light clients matter because they allow users to verify incoming data rather than blindly trusting that their data provider is correct and honest, while using just a tiny fraction of the computational resources of a full node. The data light clients receive can be checked against block headers that they know have been signed by at least 2/3 of a random set of 512 Ethereum validators. This is very strong evidence that the data is correct.

The light client only uses a tiny amount of computing power, memory and storage so it can be run on a mobile phone, embedded in an app or as part of a browser. Light clients are a way to make trust-minimized access to Ethereum just as frictionless as trusting a third-party provider.

Let's take a simple example. Imagine you want to check your account balance. To do this you have to make a request to an Ethereum node. That node will check its local copy of the Ethereum state for your balance and return it to you. If you don't have direct access to a node, there are centralized operators that provide this data as a service. You can send a request to them, they check their node, and send the result back to you. The problem with this is that you then have to trust the provider to be giving you the correct information. You can never really know the information is correct if you can't verify it for yourself.

A light client addresses this issue. You still request data from some external provider, but when you receive the data back it comes with a proof that your light node can check against the information it received in the block header. This means Ethereum is verifying the correctness of your data instead of some trusted operator.

What innovations do light clients enable? {#what-innovations-do-light-clients-enable}

The primary benefit of light clients is enabling more people to access Ethereum independently with negligible hardware requirements and minimal reliance on third parties. This is good for users because they can verify their own data and it is good for the network because it increases the number and diversity of nodes that are verifying the chain.

The ability to run Ethereum nodes on devices with very small storage, memory and processing power is one of the major areas of innovation unlocked by light clients. Whereas today Ethereum nodes require a lot of computing resources, light clients could be embedded into browsers, run on mobile phones and perhaps even smaller devices such as smart watches. This means Ethereum wallets with embedded clients could run on a mobile phone. This means mobile wallets could be much more decentralized as they wouldn't have to trust centralized data providers for their data.

An extension of this is enabling **internet of things (IoT)** devices. A light client could be used to quickly prove ownership of some token balance or NFT, with all the security guarantees provided by the sync committees, triggering some action on an IoT network. Imagine a bicycle rental service that uses an app with an embedded light client to quickly verify that you own the rental service's NFT and if so, unlocks a bicycle for you to ride away on!

Ethereum rollups would also benefit from light clients. One of the big problems for rollups has been hacks targeting the bridges that allow funds to transfer from Ethereum Mainnet to a rollup. One vulnerability is the oracles that rollups use to detect that a user has made a deposit into the bridge. If an oracle feeds bad data, they could trick the rollup into thinking there was a deposit to the bridge and to incorrectly release funds. A light client embedded in the rollup could be used to protect against corrupted oracles because the deposit into the bridge could come with a proof that can be verified by the rollup before releasing any tokens. The same concept could also be applied to other interchain bridges.

Light clients could also be used to upgrade Ethereum wallets. Instead of trusting data provided from an RPC provider, your wallet could directly verify the data being presented to you using an embedded light client. This would add security to your wallet. If your RPC provider was dishonest and provided you with incorrect data, the embedded light client could tell you!

What is the current state of light client development? {#current-state-of-development}

There are several light clients in development, including execution, consensus and combined execution/consensus light clients. These are the light client implementations we know of at the time of writing this page:

- Lodestar: consensus light client in TypeScript
- Helios: combined execution and consensus light client in Rust
- Geth: light mode for execution client (in development) in Go
- Nimbus: consensus light client in Nim

To our knowledge none of these are considered production-ready yet.

There is also a lot of work being done to improve the ways that light clients can access Ethereum data. Currently, light clients rely on RPC requests to full nodes using a client/server model, but in the future the data could be requested in a more decentralized way using a dedicated network such as the Portal Network that could serve the data to light clients using a peer-to-peer gossip protocol.

Other roadmap items such as Verkle trees and statelessness will eventually bring the security guarantees of light clients equal to those of full clients.

Further reading {#further-reading}

- Zsolt Felfodhi on Geth light clients
- Etan Kissling on light client networking
- Etan Kissling on light clients after The Merge
- Piper Merriam: The winding road to functional light clients