Mike Mendoza & Franco Charette
0xPASTE / ontokens.xyz
December 3, 2022

Layer 2 Light clients

Abstract. Current light clients like metamask and mobile apps can be a little unsecured since they are just requesting eth_getBalance from default RPC url's. The idea is to have RPC-based wallet completely trustless by first syncing to the latest header of the beacon chain and then use the eth_getProof endpoint to get the balance plus a proof that it is actually part of the root hash that we obtained. Using Merkle Inclusion proofs to the latest block header allows us to verify that the data is correct.

> Another approach is we can search the header for blooms that tells us that a particular block may contain a transaction that are logged that we are interested in. We can actually request the transaction receipt block, and search in that block for bloom filters.

What is a Layer 2?

According to ethereum.org: "Layer 2 (L2) is a collective term to describe a specific set of Ethereum scaling solutions. A layer 2 is a separate blockchain that extends Ethereum and inherits the security guarantees of Ethereum." On top of layer 1s, layer 2 refers to a collection of off-chain solutions (distributed blockchains) that alleviate scale and data bottlenecks. Consider a restaurant kitchen where just a few orders could be filled every hour if each order had to be completed by a single worker from start to finish before being confirmed and delivered. But layer 2s are more like prep stations, where each station can concentrate and do each operation much more quickly. For example, there is a station for washing and chopping food, a station for cooking, and a station for assembling the dishes. A last person can validate the order by comparing each assembled dish to the order when the time is appropriate.

Different types of Layer 2

Generalized layer 2:

boba - Boba is an Optimistic Rollup originally forked from Optimism which is a scaling solution that aims to reduce gas fees, improve transaction throughput, and extend the capabilities of smart contracts.
Note: State validation in development

Arbitrum - boosting the speed while adding privacy and addt'l features, it's the most EVM-compatibile layer 2 solution today. Arbitrum is an Optimistic Rollup that aims to feel exactly like interacting with Ethereum, but with transactions costing a fraction of what they do on L1.
Note: Fraud proofs only for whitelisted users, whitelist not open yet

Optimism - Optimism is a fast, simple, and secure EVM-equivalent optimistic rollup. It scales Ethereum's tech while also scaling its values through retroactive public goods funding. Note: Fault proofs in development

Application specific layer 2:

Zk-sync - zkSync is a user-centric zk rollup platform from Matter Labs. It is a scaling solution for Ethereum, already live on Ethereum mainnet. It supports payments, token swaps and NFT minting.

ZkSpace - The ZKSpace platform consists of three main parts: a Layer 2 AMM DEX utilizing ZK-Rollups technology called ZKSwap, a payment service called ZKSquare, and an NFT marketplace called ZKSea.

Loopring - Loopring's zkRollup L2 solution aims to offer the same security guarantees as Ethereum mainnet, with a big scalability boost: throughput increased by 1000x, and cost reduced to just 0.1% of L1.

Aztec - Aztec Network is the first private zk-rollup on Ethereum, enabling decentralized applications to access privacy and scale.

What is an optimism rollup?

Optimistic rollups control the communication between the L2 chain and the L1 blockchain using smart contracts installed on Ethereum (Ethereum). Users of the rollup must make a deposit into one of these smart contracts before an equivalent amount may be un-

locked. After receiving verification of the user's deposit in the rollup contract, a third party known as a sequencer credits the user with monies on the rollup. The user is thereafter unrestricted in their use of the rollup until their balance runs out.

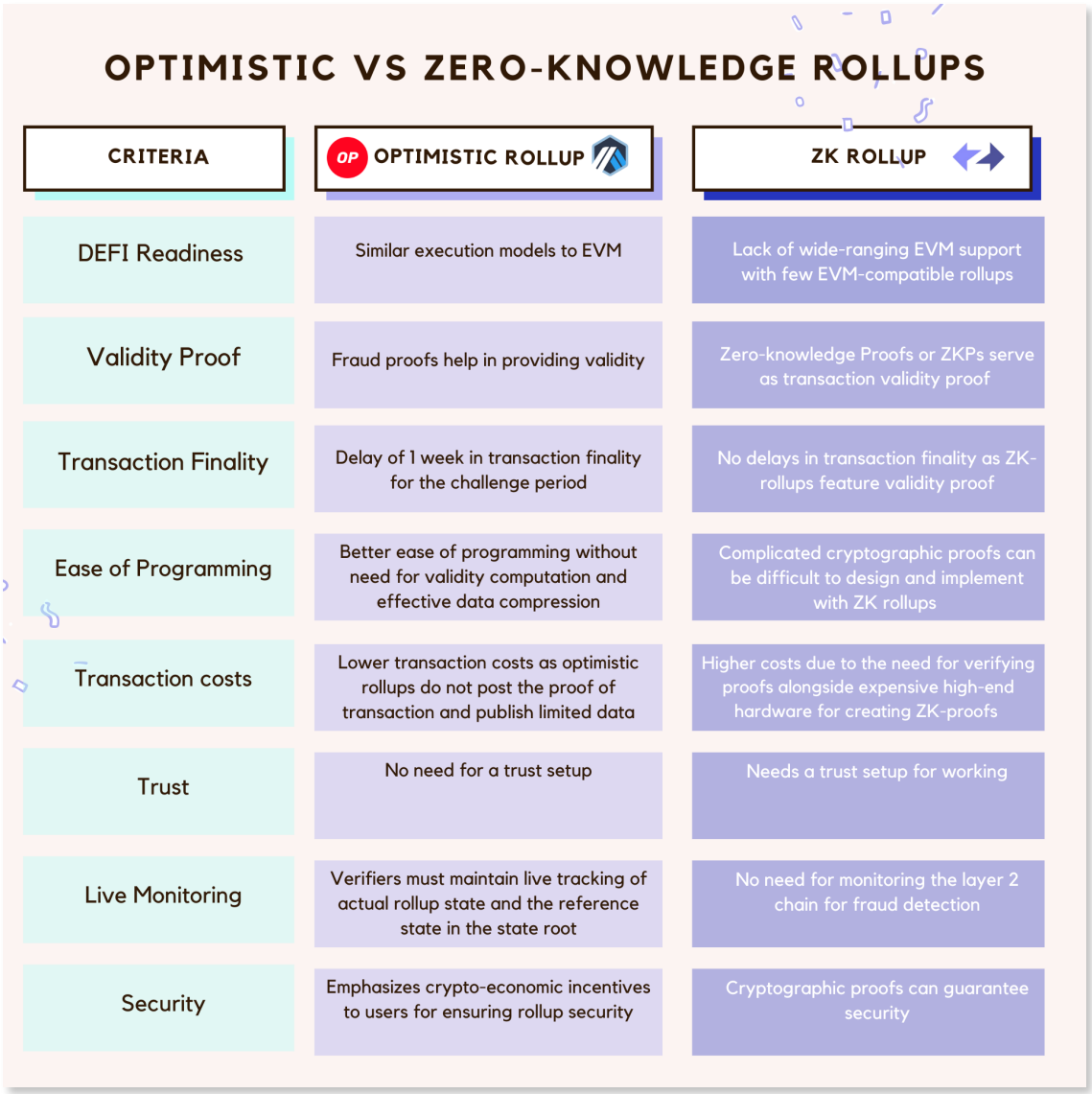
Users sign transactions on the rollup and send them to the sequencer, who is in charge of arranging and carrying out transactions. The sequencer compiles the batch of transactions into a block, verifies each transaction, and presents it to Ethereum as a single transaction. The rollup contract enters the picture in this situation as well. A "state root," or Merkle root of the rollup's state, is kept in the on-chain contract.

What are ZK-Rollups?

According to ethereum.org: “Zero-knowledge rollups (ZK-rollups) are layer 2 scaling solutions that increase throughput on Ethereum Mainnet by moving computation and state-storage off-chain. ZK-rollups can process thousands of transactions in a batch and then only post some minimal summary data to Mainnet. This summary data defines the changes that should be made to the Ethereum state and some cryptographic proof that those changes are correct. A ZK-rollup chain is an off-chain protocol that operates on top of the Ethereum blockchain and is managed by on-chain Ethereum smart contracts. ZK-rollups execute transactions outside of Mainnet, but periodically commit off-chain transaction batches to an on-chain rollup contract. This transaction record is immutable, much like the Ethereum blockchain, and forms the ZK-rollup chain.”

Comparisons with both

The ZK rollups might have a better appeal in ZK rollups vs optimistic rollups comparison for the element of cryptographic proof. However, they could not be the recommended option for scaling in all cases. A high-level overview of the comparison between ZK rollups and optimistic rollups shows some insights into differences between them.



In the case of optimistic rollups, transactions are assumed as valid, while zk-rollups rely on computation verification of state transitions through validity proofs. However, optimistic rollups are better production-ready with easier programmability. In addition, optimistic rollups also facilitate effective use of EIP-4844 for ensuring cost efficiency. Therefore, optimistic rollups turn out to be the better alternative in reality.

What are nodes & clients?

According to ethereum.org: “A ‘node’ is any instance of Ethereum client software that is connected to other computers also running Ethereum software, forming a network. A client is

an implementation of Ethereum that verifies data against the protocol rules and keeps the network secure."

Post-Merge Ethereum consists of two parts: the execution layer and the consensus layer. Both layers are run by different client software. On this page, we'll refer to them as the execution client and consensus client. The execution client (also known as the Execution Engine, EL client or formerly the Eth1 client) listens to new transactions broadcasted in the network, executes them in EVM, and holds the latest state and database of all current Ethereum data. The consensus client (also known as the Beacon Node, CL client or formerly the Eth2 client) implements the proof-of-stake consensus algorithm, which enables the network to achieve agreement based on validated data from the execution client.

Different types of nodes

Full node

A full node keeps a complete copy of the blockchain data (although this is periodically pruned so a full node does not store all state data back to genesis), and contributes to the network by receiving transactions and blocks from other full nodes. It also participates in block validation, verifies all blocks and states. All states can be derived from a full node (although very old states are reconstructed from requests made to archive nodes). It also serves the network and provides data on request.

Light node

Instead of downloading every block, light nodes download block headers. These headers only contain summary information about the contents of the blocks. Any other information required by the light node gets requested from a full node. The light node can then independently verify the data they receive against the state roots in the block headers. Light nodes enable users to participate in the Ethereum network without the powerful hardware or high bandwidth required to run full nodes. Eventually, light nodes might run on mobile phones or embedded devices. The light nodes do not participate in consensus (i.e. they cannot be miners/validators), but they can access the Ethereum blockchain with the same functionality and security guarantees as a full node.

Light clients

Light clients are an area of active development for Ethereum and we expect to see new light clients for the consensus layer and execution layer soon. There are also potential routes to providing light client data over the gossip network. This is advantageous because the gossip network could support a network of light nodes without requiring full nodes to serve requests. Ethereum does not support a large population of light nodes yet, but light node support is an area expected to develop rapidly in the near future. In particular, clients like Nimbus, Helios, and LodeStar are currently heavily focused on light nodes.

How to build a light client?

Kevlar is a CLI tool to run a light client-based RPC Proxy for PoS Ethereum. Kevlar can be used to make your Metamask or any RPC-based wallet completely trustless! Kevlar first syncs to the latest header of the beacon chain and then starts an RPC server. This local RPC server can be added to MetaMask or any other RPC-based wallet. Every RPC call made by the wallet is now verified using Merkle Inclusion proofs to the latest block header. Currently Kevlar supports two kinds of sync methods: the **Light Sync** based on the light client sync protocol specified by the Ethereum Specification and the **Optimistic Sync** (which is 100x faster than Light Sync) based on construction from the research paper Proofs of Proof of Stake in Sublinear Complexity.

Start the RPC Proxy
npm i -g @lightclients/kevlar
kevlar

Light client challenges

New layer one blockchains forecast a decentralised future that moves at light speed. However, what we actually discover is usually a collection of intermittent, uncommitted users who connect to a single sponsored JSON RPC run by a foundation and connected to a sparse network of validator cartels.

Is this the future we want, or is it just a carbon copy of the existing power structures? It might be fine for the time being. The fact is that we can do better.

Several factors make decentralisation important. By doing so, a system becomes less vulnerable to single points of failure, hostile takeover attempts, and governmental censorship. Keep in mind that the beacon chain contains a number of validator pools with non-trivial shares. We require a wide range of validators because of this. We must be watchful and we require more evangelists for decentralization. More users can engage as full participants thanks to light clients, which check on-chain data without relying on a single, centralised JSON RPC endpoint.

What would happen if a bad full node attempted to utilise 1000ETH as the account balance in the merkle proof? How can a light client determine the validity of the Merkle Proof and reject the invalid account balance?

Since the light client generates all of the hashes for the trie nodes, it will generate a unique hash for the LeafNode. Additionally, the ExtensionNode's hash will change from the original hash4 because hash1 is intended to be a part of both the BranchNode and the ExtensionNode. As a result, the light client won't be able to locate the ExtensionNode when traversing the trie with the StateRoot hash4. The merkle proof is thus deemed invalid by the light client.

Getting proof and implementation

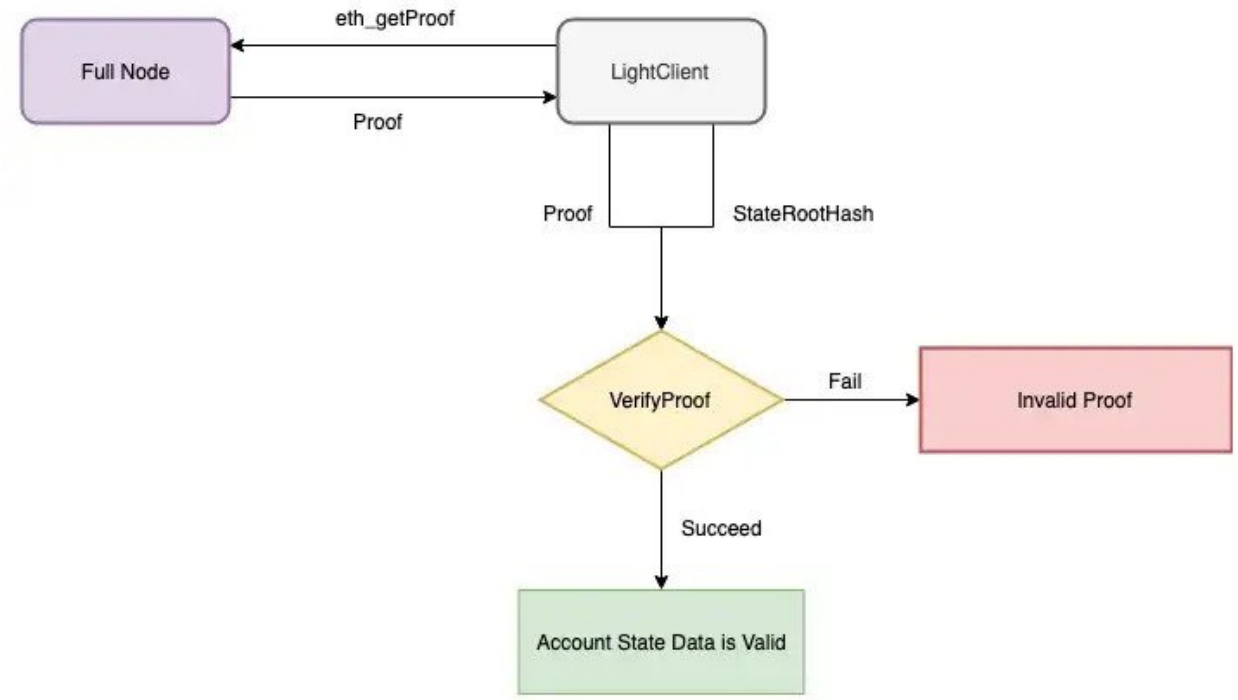
Use-cases for getting account proof

1) You can ask for Merkle Proof for your Ethereum account balance and verify the correctness yourself.
2) You can ask for Merkle Proof for your ERC20 token account balance. For instance, USDC account balance.
3) You can ask for Merkle Proof for your ERC721 token ownership. For instance, to prove you are the owner of a BAYC NFT.

Since you can readily check the proof, you can query these Merkle Proofs from untrusted full nodes. If the Merkle Proof is invalid, the storage state or result of your Ethereum account that was transmitted with the proof will also be invalid.

EIP-1186

EIP-1186 defines a method eth_getProof to query for Merkle proof: The eth_getProof method takes an address to query account state for and a block number. It returns an account object, which contains the account state data including balance, nonce, storageHash and codeHash. More importantly, it also return the accountProof for verifying the account state data.



In order to verify the Merkle proof with the state root hash, we need to use the Merkle Patricia Trie data structure. Here, we will continue using the simplified implementation of the Merkle trie to verify the proof that we queried from eth_getProof call.

```
// create a proof trie, and add each node from the account proof
 proofTrie := NewProofDB()
 for _, node := range result.AccountProof {
  proofTrie.Put(crypto.Keccak256(node), node)
 }
```

The proof is essentially the encoded trie nodes on the path from the root node of the trie to the leaf node that stores the account state data. Now the trie is constructed, we call the VerifyProof method of the trie and pass in the account address and the state root hash.

```
// get the state root hash from etherscan:
https://etherscan.io/block/14900001
 stateRootHash :=
common.HexToHash("0x024c056bc5db60d71c7908c5fad6050646bd70fd772ff22270
2d577e2af2e56b")

// verify the proof against the stateRootHash
 validAccountState, err := VerifyProof(
  stateRootHash.Bytes(), crypto.Keccak256(account.Bytes()), proofTrie)
 require.NoError(t, err)
```

The state root hash is the start point from which we traverse through the trie, and the account address is the actual path to traverse through the trie. If we are able to reach a leaf node in the end, then the proof is valid. If we didn't reach a LeafNode, then the proof is invalid. In the example, when we run the test case, it passes. So it means the account state is valid.