Thanks to Karl Floersch for his sanity check on this idea.

TLDR:

Communication between layers can be achieved by having access to the other layer's state root and using a Merkle proof to verify some data. An implementation of this that is smart contract based however is not very efficient (Merkle proof data and verification) and isn't very convenient because data has to be made available explicitly before it can be used by a smart contract. Instead, we can expose a precompile contract that can directly call a smart contract on the destination chain. This precompile injects and executes the smart contract code of the other chain directly in the source chain. This makes sure that smart contracts always have access to the latest available state in an efficient and easily provable way.

[

so_simple

1080×539 8.28 KB

](https://ethresear.ch/uploads/default/original/2X/1/1421896f0674efcce86467d2d367ff752a8adeb9.png)

Intro

To make things easier let's only consider the L1/L2 case, but the same things hold for L2/L3 etc…

An elegant way to do cross layer communication is to make sure that each layer has access to the other layer's state root. For a smart contract based layer 2, the L1 will have access to the L2's blockhash. The L1 state root is injected somehow in the L2 so that it is available for any L2 smart contract to access. Any smart contract, most notably bridges, can now use that state root to verify any data necessary using a Merkle proof.

This system works great, and works well for any chain, but also has a couple of disadvantages:

- Data needs to brought in manually from the different chain because it depends on a Merkle proof data not known by the smart contract itself. It is impossible for a smart contract to simply point to some data on the other chain and read it from there directly.
- Depending on the type of state root check, this can get quite expensive in both data and smart contract logic.

We also know that an L2 node already needs to run an L1 node to get access to the L2 data published on the L1, and so the L2 node already has access to the L1 state.

Proposal

We propose exposing a precompile that calls an L1 smart contract. This precompile works in the same way as a normal call, but additionally switches out the normal L2 state root for the L1 state root injected into the current L2 block. This state root is active for the duration of the precompile, the L2 state root is restored after the precompile ends. Any state updates done in the precompile are discarded.

Because this requires executing the L1 EVM bytecode and reading from the L1 Merkle Patricia Trie, this only works out of the box for L2s that are EVM equivalent and use the same storage tree format.

This precompile could be implemented by having the L2 node do an RPC call to the L1 node to execute (but actually just simulate) the requested call.

Advantages

- Efficient

: We have replaced a smart contract emulated SLOAD by an actual SLOAD (+much more!). All necessary Merkle proof data remains internal and does not need to be made public in a transaction. Also no chain bloat by having bots/users bringing over this cross-chain data.This also is a reasonable argument to prefer an L2 with multiple L3s instead of just multiple L2s so that data can be shared more easily between L3s simply by making it available once on the L2.

- Convenient:

All latest L1 data is made available directly to L2 smart contracts, no extra steps required. For example, an L2 defi app can now simply use the precompile to read the latest oracle data on L1.

- Simple to prove:

For L2s that already support proving EVM bytecode execution and MPT it is as simple as just pointing to a different state root.

Disadvantages

- Only works for chains the L2 already depends on:

Supporting additional chains would increase the hardware requirements of the L2.

- L1 and L2 nodes have to run in sync, with low latency communication:

When executing L2 transactions it needs to be possible to efficiently execute L1 calls against the expected L1 state for that L2 block.

- Full L1 state needs to be available for anyone running an L2 node

: Prevents someone running an L2 from trying to minimize the L1 state.