TL;DR:

We propose zkThread, an application-level component allowing users to locally prove a batch of transactions and update the canonical state.

This protocol allows turning any smart contract or set of smart contracts into their own execution environment at will while leaving the system in a synchronized environment.

Notice this protocol can also be implemented at the OS level without requiring any change on the user journey.

This design pattern showcases Horizontal Scaling done right. zkThreads extends the base layer throughput and network capacity at Zero Overhead from the core protocol perspective.

We believe this model to be a new paradigm in the Fractal Scaling family alongside L3s and other designs known as zkSharding. You can find more details and contribute to its creation on here and a first implementation can be found under this repository under the Saya settlement service developed by Cartridge.

As a metaphor, imagine a chain where you can temporarily fork the chain locally to batch txs on your own. Others have hinted at such a design under the name Stateful Coprocessor or Execution Coprocessor.

Thanks a lot to Jay Yu from Stanford Blockchain and Tarrence from Cartridge for all the help in formalising and writing this post.

Background

In the context of a Validity-Rollup, the OS underpinning the Rollup is usually optimized for being provable, making proof a first-class primitive of the whole system. One important aspect is the ease of proving as one needs to run in a decentralized world with constrained machines.

Under the current system, Starknet and other chains only allow a smart contract to modify its storage slots.

[

image

1238×620 88.9 KB

](https://ethresear.ch/uploads/default/original/3X/a/d/ad676517d8e89bbc4d937ee07128835d0c72d402.png)

Proposal

Under zkThread, a new contract would be introduced to allow for proof verification of generic transactions against a specific OS version, a set of contracts, an input, and an output state list. Assuming the input state list matches the current canonical state values and assuming the proof verifies, the OS can self-apply the output to the canonical state.

We will call a zkThread

, the operation governing the runtime of the zkThread contracts updates.

Stated Goals

This protocol enables a whole new primitive into the network

- All compatible dApps become L3-ready
- Self Sharded Synchronous state
- Infinitely Adjustable TPS through permissionless self-update
- Future-proof backward compatibility

Setup

[

image

1748×980 95.4 KB

](https://ethresear.ch/uploads/default/original/3X/4/d/4d2ea6cbd7ef6a2fd9520727e728980356cbce58.png)

For a set of L2 Smart Contracts, one can define a zkThread by

- Batcher: an entity or set of entities that sequences the zkThread
- Prover: an entity or set of entities that proves the zkThread and forwards it to the L2
- A set of L2 contracts falling targeted by the zkThread

We call a Sub-block the output of the Batcher.

For a zkThread, the Batcher creates a sub-block and provides it to the Prover.

Once proven,sub-blocks are forwarded as a transaction to the L2.

Assuming all transactions in the sub-block are valid, a transaction is emitted from the zkThread to the L2 as follows

[

image

1738×968 93.2 KB

](https://ethresear.ch/uploads/default/original/3X/5/b/5bfafec8753e568daf47182021da94dd16225cde.png)

The zkThread transaction is then sequenced by the L2 and run against the canonical state

[

image

1698×966 108 KB

](https://ethresear.ch/uploads/default/original/3X/c/3/c3b9902412fe93266b9309cb5b5f4a10f327c263.png)

If the proof validates against the canonical state, the state updates are accepted and incorporated into the canonical state.

[

image

1742×980 124 KB

](https://ethresear.ch/uploads/default/original/3X/8/a/8a396a5cf7e88f798ab726157d5a3da6ca323c7f.png)

In the negative flow, the transaction just reverts and the Prover pays the transaction fees.

[

image

1746×978 121 KB

](https://ethresear.ch/uploads/default/original/3X/8/5/850d64651564478bfc37889a3bff85a7ac2a28e1.png)

Conclusion

This new scaling model is perfectly suited for critical DeFi applications, Gaming and all dApps which wish to use the base layer infrastructure but have strong constraints on Responsiveness, Fees and even Privacy such as Order Book. You can find more details and contribute to its creation here and a first implementation can be found under thisrepository under the Saya settlement service developed by Cartridge.