Herodotus Turbo

Storage proofs are a cryptographic way to track and share blockchain information across chains. Similar to oracles, storage proofs provide proof that the information is true. However, unlike oracles, they do not require trust in a third party for this proof; rather, with storage proofs, the trust is built into the storage. The beauty of proofs lies in their ability to prove that a certain fact is true cryptographically. These proofs can be optimized with SNARKS/STARKS (Succinct Non-Interactive ARgument of Knowledge/Scalable Transparent ARgument of Knowledge).

In our recent [article](https://www.starknet.io/en/posts/developers/what-are-storage-proofs-and-how-can-they-improve-oracles#) on applications of storage proofs, we discussed multiple use cases for storage proofs. The most simplistic application for such proofs is proving a certain state of blockchain A on blockchain B without using an oracle as the communication medium. This can further be developed into trustless and transparent bridges, cross-chain voting systems, and asynchronous oracle updates.

[Herodotus](https://www.herodotus.dev) team is the current leader in the research and development of Storage Proofs on Starknet. They have been working on building Herodotus, a storage-proof-as-a-service application, for the past few years. Recently, Herodotus Turbo was [announced](https://twitter.com/HerodotusDev/status/1682400907969822722?s=20), which makes utilizing storage proofs to carry information between chains as easy as writing a single line of code! Let’s explore what Herodotus Turbo is and how developers can leverage this innovation in cross-chain interoperability.



<https://www.herodotus.dev>

**Traditional (API) approach**

Earlier, the experience of a developer who wished to include storage proofs in their application was cumbersome. Application developers used to include many steps in their code to harness the capabilities of storage proofs. The workflow for proving storage from the source chain on the destination chain (say, Ethereum mainnet is the source chain and Starknet - is the destination chain) used to be the following:

**Step 1:** Obtain an API Key  
The developer begins by registering on [Herodotus](https://www.herodotus.dev) website and creating an API key for interacting with the Herodotus API.

**Step 2:** Calculate the storage slot  
Determine the storage slot of the specific Ethereum contract you, the developer, want to generate proof of. Every smart contract on EVM chains has an associated storage space to maintain contract variables. This means that every mapping, array, or standalone variable element has a storage space (and an associated storage slot key). It is possible to generate proofs for all the slots of a contract. If the developer wishes to retrieve the user’s balance of the DAI token, they must determine the storage position (and the corresponding key) with the DAI smart contract’s `balances` mapping.

**Step 3:** Call eth\_getProof and send the proof to Herodotus API  
With the storage key and Herodotus API at hand, the developer needs to call the EVM JSON RPC method `eth\_getProof` and call the Herodotus API to retrieve a storage proof that can now be used on the destination chain.

**Step 4:** Utilize the FactsRegistry contract

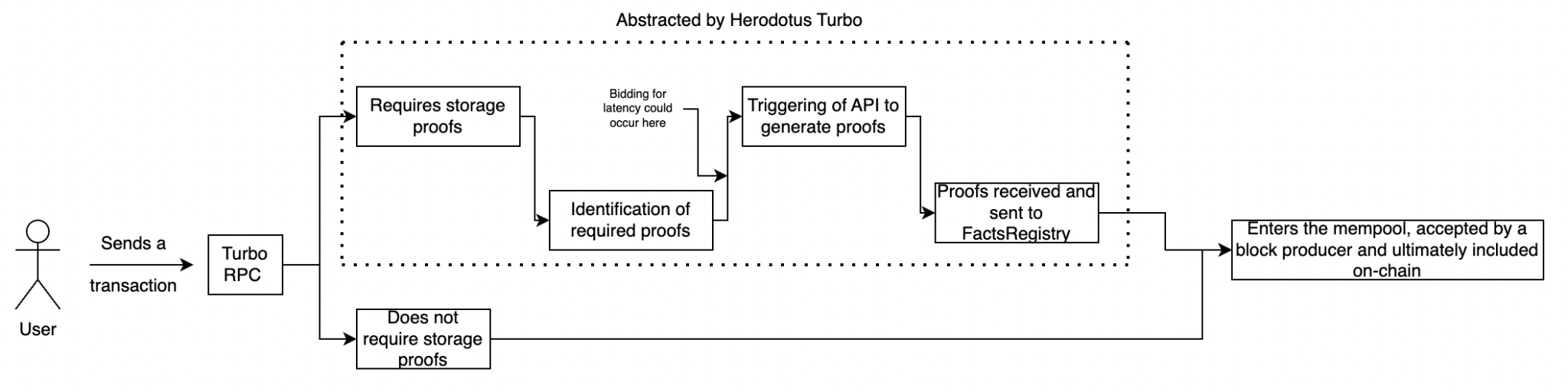
On the target chain, a FactsRegistry smart contract is essential for verifying the storage. These FactsRegistry smart contracts are already deployed on-chain, so the developer needs to send the generated storage proof along with the block height (at which the storage is proven) to the FactsRegistry contract. In response, the FactsRegistry contract examines the proof and, if validated, showcases the value of the storage slot that we proved on the source chain.

**Step 5:** Further steps  
Once the FactsRegistry smart contract validates the proof, the application can continue its logic as needed.

**New (Turbo) approach**

With the launch of Herodotus Turbo, it becomes possible to complete all the mentioned steps in a much more concise code. The Herodotus Turbo service abstracts most of these steps and introduces an auction-bid system, allowing users to bid on the priority of proof generation based on the urgency of their requirements. Exciting? Let’s understand it better, then!

Herodotus Turbo relies completely on the same systems that the traditional storage proofs rely on, but it simplifies their application. With Herodotus Turbo, all on-chain transactions can be classified as belonging to either of two types: ones utilizing storage proofs and those not. The workflow remains the same for ordinary transactions that do not use any storage proofs: they are received by a node, fed into the mempool, selected by the block producer, and, as a result, included in the block. However, the process differs for transactions using Herodotus Turbo for storage proofs.



The user must send their transaction to a Turbo RPC to use the power of storage proofs with Turbo. Turbo handles the rest of the job, encapsulating the user in the process. However, a process is initiated on Herodotus' side of things that puts a chain of actions in motion.

1. Transactions are traced to identify what storage proofs are required. Each individual proof is determined and identified.

On the RPC node, the transaction steps are traced individually, and at each step, it is determined if it requires any storage proof to be calculated from a source chain. If none of the steps require it, the transaction is treated the same way any other RPC would treat it and sent to the network.

1. Auction-bid system put in place to determine the priority of generation of storage proofs.

The bidding system for Herodotus Turbo transaction is still under development. However, it will be a core piece of the entire infrastructure. Some use cases (like gaming) do not require transactions to be immediately accepted on the network and could be optimistically accepted on the user side until the transaction goes through. For such cases, a higher latency in generating the storage proof in exchange for a cheaper cost is a viable option. On the other hand, some users would want their transaction to be accepted on-chain as soon as possible and would be willing to pay a higher fee. To capture this extra value and to be able to provide cheaper fees, the Herodotus Turbo transaction will go through a bidding mechanism to determine the priority of what storage proofs are to be generated first (somewhat similar.

1. Generation of the required storage proofs

This step is similar to the traditional method used by Herodotus API to generate proofs.

1. Proofs are received, submitted to the FactsRegistry contract and the rest of the transaction goes through to the network.

After the waiting time (which depends on the fee the user is willing to pay), the storage proofs are generated and sent to the Fact Registry contract. If the proofs are valid, the rest of the transaction can now be completed, so it is sent to the network.

**Further Research and Development**

The incredible work done by the Herodotus team in simplifying storage proofs and integrating them into blockchain development workflow is commendable. However, as with any other team that achieves great results, Herodotus team is still working on optimizing future research and development:

**Implementing the Bidding system**

The above-mentioned auction system is an efficient way to balance latency and the cost of storage-proof processing.

On top of this, there are nonzero chances that there is at least some amount of MEV extractable, and this could be a potential research avenue for others who want to understand the system better.

**Caching requests for cost efficiency**

Caching previous requests can greatly reduce the computational overhead for Herodotus and hence reduce the fees for using storage proofs in user transactions.

**Example:** Consider a scenario where user A requests the Time-weighted Average Price for a common token pair on Uniswap over 100 blocks. If some other user B had also recently requested the TWAP for a subrange of those blocks, the system can use the cached results to save on computation and hence offer a lower fee to user A.

Multiple projects are actively building on top of Herdotous's stack, utilizing the power of storage proofs. [Snapshot X](https://snapshotx.xyz/) is trying to use storage proofs to allow on-chain voting on L2 networks without bridging the tokens from the safer Ethereum mainnet (L1). Similarly, [Pragma](https://www.pragmaoracle.com/) is working on the first provable oracle for a trustless way of using cross-chain data. The use of storage proofs is similarly vital for maximizing their efficiency.

There is still lots of research avenue in the field of storage proofs, and innovative teams like Herodotus are needed to unleash the power of this cryptographic "magic"!