Rolling Asynchronous Interchain Liquidity Settlement

Chris Whinfrey

April 2025

Abstract

This paper introduces Rails, a novel protocol designed to facilitate high-efficiency trustless token transfers across blockchain networks. Existing trustless bridge protocols rely on delayed message settlements for each transfer, leading to high capital requirements and pushing protocols toward less secure and more centralized messaging solutions. Rails enables direct peer-to-peer settlement, circumventing message times and significantly improving capital efficiency. An open network of bonders provides faster-than-finality execution on top of the peer-to-peer settlement mechanism.

Rails achieves the best-case capital efficiency profile currently offered only by trusted bridges. By optimizing liquidity settlement dynamics, Rails reduces costs and lowers barriers for cross-chain liquidity, fostering a more interconnected Ethereum ecosystem.

1 Introduction

While rollups have gained traction as Ethereum's predominant scaling solution, interoperability between rollups remains a core challenge. Interoperability can be divided into two categories: messaging — the ability for one chain to communicate with another; and asset maneuverability — the ability to move an asset from one chain to another. While advancements in cross-chain messaging, such as state proofs and on-chain light clients, are improving communication, liquidity fragmentation persists.

Asset maneuverability is straightforward for homogeneous rollups that share identical security properties and trustless messaging. Tokens can be transferred using a simple burn-and-mint bridge mechanism without compounding security risks. Bridging assets across heterogeneous rollups, however, presents a fundamental problem. Each heterogeneous rollup operates within a distinct security framework, making maneuverability of trustless assets via burn-and-mint mechanisms impossible without inheriting the cumulative security risks of each connected rollup.

Robust liquidity bridges are necessary infrastructure for enabling token maneuverability across rollups without compromising on security. Rails addresses

this need by optimizing capital efficiency and mitigating the security risks inherent to existing bridge models.

2 Limitations of Existing Bridges

Current bridge solutions are limited by their capital efficiency, risk isolation, and rebalancing mechanics.

Existing bridge solutions are message-settled — that is, funds are locked for the time it takes for a message to be passed from chain A to chain B. Optimistic rollups enforce trustless message times of approximately seven days, while zk-rollup message times depend on L1 checkpointing intervals ranging from 30 minutes to 12 hours. Messages can be sped up using a more aggressive optimistic window, but shorter windows compromise on security. To overcome these capital inefficiencies without sacrificing security, liquidity settlement must be decoupled from the underlying message settlement.

Some existing bridge solutions also lack risk isolation for external liquidity providers. Liquidity providers for these solutions are exposed to the risks of all chains in the network with risks increasing as the network grows. A maximally secure, scalable bridge solution must allow external liquidity providers to be exposed only to the chains they are on and keep them isolated from the other chains in the network.

Existing solutions take various suboptimal approaches to asset rebalancing. Some solutions externalize asset rebalancing to arbitrageurs but rely on capital-intensive liquidity pools. Other solutions require active liquidity providers to individually rebalance liquidity themselves leading to inefficiencies and centralization around the largest participants. An ideal bridge solution externalizes the rebalancing of assets without relying on large liquidity pools.

3 Rails

Rails (Rolling asynchronous interchain liquidity settlement) is a novel bridge protocol that allows users to settle directly with one another. This separates the settlement of cross-chain transfers from the underlying message settlement, dramatically improving capital efficiency.

In Rails, each transfer includes a **state attestation** containing information about prior transfers from the destination chain, allowing the protocol to use liquidity from the new transfer to free up prior transfers on a rolling basis. A **virtual automated market maker (AMM)** dynamically adjusts rates between chains to maintain balanced liquidity flows without requiring external liquidity provisioning. Transfers generate **claims** at the destination, which are released in first-in-first-out (FIFO) order. Additionally, an open network of bonders/solvers can facilitate faster-than-finality execution for immediate settlement needs. Each Rails **path** connects two chains, and multiple paths can be combined to route transfers efficiently to their destination. This construc-

tion enables a trustless and highly efficient method of exchanging assets across chains.

3.1 State Attestations

State attestations are the core innovation enabling Rails's capital efficiency. When transfers are initiated, liquidity is deposited at the source along with a state attestation referencing prior transfers from the destination. A valid transfer must include a valid state attestation or it cannot be withdrawn at the destination where the state attestation is validated. This allows Rails to leverage the deposits of transfers on the source chain to free up claims of transfers that came from the destination chain.

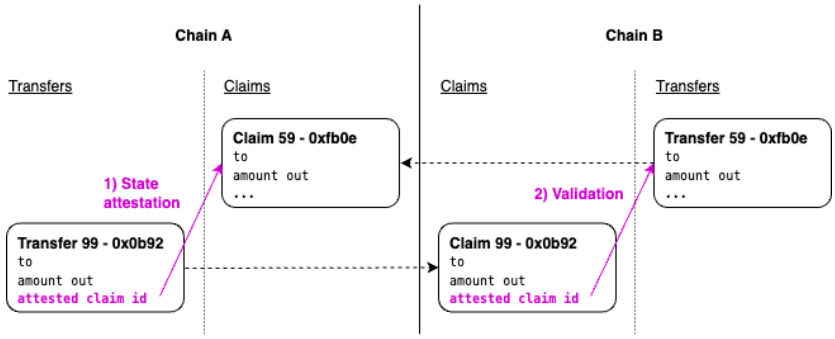

Figure 1: Each transfer attests to claims of transfers from the destination. State attestations are validated at the destination or the claim cannot be withdrawn.

Each state attestation references a single claim in the bridge's state but can be applied to free any previous claim. This is essential because claims are freed up in first-in-first-out order.

To achieve this, claims form a **state chain** where each claim's unique identifier (**claim id**) is the hash of the claim data (i.e. to, amount out, etc.) and the previous claim id. An attestation to any given claim id is also attesting to all previous claim ids, because any change in the state chain would result in a different claim id for the attested claim.

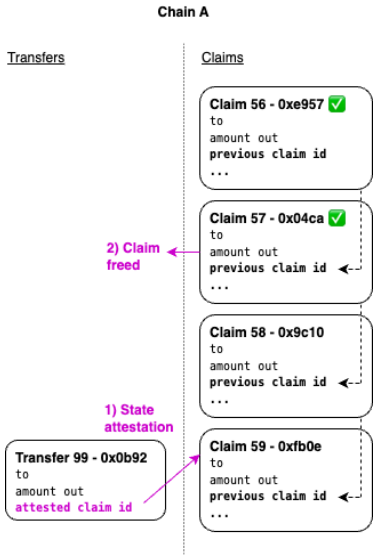

Figure 2: Transfer 99 attests to claim 59 which causes claim 57, the earliest unfreed claim, to be freed.

The claim chain is confirmed in two steps — first by state attestations which allow claims to be withdrawn and later by message settlement which finalizes the claim chain at a given claim.

3.2 Flow Balancing

Because cross-chain transfers in one direction are used to free up transfers coming from the other direction, it's important that these flows remain balanced over time. Rails sets a dynamic rate between chains using a virtual AMM model, requiring no external liquidity.

The virtual AMM curve is set upon initialization of a new path, with a tradeoff between liquidity efficiency and slippage. A shallower curve, representing a larger pool of virtual liquidity, will result in lower slippage for transfers but higher flow imbalances leading to longer queues. Conversely, a steeper curve will result in shorter queues but more slippage. A properly set AMM curve should take into account this tradeoff to strike a balance for users. Multiple paths, utilizing different curves, may be instantiated between two chains for the same asset to provide the appropriate tradeoffs for different sizes of transfers.

The virtual AMM does not require liquidity providers because the outputs of the AMM are token claims and not liquid tokens. Any curve may be set representing any amount of virtual liquidity. This removes the high barrier of bootstrapping liquidity required by current bridges. Additionally, no AMM fee is needed to pay liquidity providers further reducing costs.

3.3 Fast Execution

To address user demand for near-instant settlement, Rails incorporates an open network of bonders/solvers that enable fast execution on top of the peer-to-peer settlement mechanism. These participants provide liquidity for transfers that require immediate fulfillment in exchange for a small fee. Bonders can complete bridge transfers faster than source transactions reach finality using a novel mechanism called **contingent transactions**. (First implemented in Hop Protocol v1, August 2023.)

Contingent transactions allow bonders to specify an L1 blockhash, ensuring that the bonded transaction only executes if the referenced block is valid. By referencing the block of the source transaction, the bonder can guarantee that either the source transaction is confirmed or the bonder's transaction is reverted, preventing loss of funds. Contingent transactions can be chained together for transfers routed through multiple chains, guaranteeing that subsequent transactions are reverted if any previous transaction in the transaction chain is forked out.

3.4 Routing Cross-Chain Transfers

Rails connects chains through **paths**, where transfers in one direction free up claims from transfers in the other direction and vice versa. The more volume that is directed through a path, the faster the claims are freed up, driving path efficiency. While it is possible to establish direct paths between all chains, doing so would fragment cross-chain volume and lead to longer queue times.

Transfers can be routed through multiple paths on their way to their destinations, reducing the overall number of paths required to connect supported chains. This enables alternative graphs of connected chains with far fewer paths such as the hub-and-spoke model. For example, directly connecting 1,000 chains would necessitate 499,500 paths, whereas a hub-and-spoke model requires only 1,000 paths, greatly improving capital efficiency.

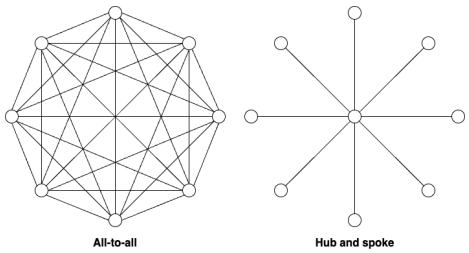

Figure 3: Routing through a hub chain dramatically reduces the number of paths required.

4 Interacting With Rails

For end users, interacting with Rails is no different than interacting with other bridge protocols whether it is a traditional bridge, intent, or chain abstraction experience. Wallets, frontends, or backend infrastructure can construct Rails transfers with state attestations in their current transaction creation processes with two simple RPC calls. First, the latest claim is fetched from the Rails contract. This claim is then verified with a second call to the destination contract. After verification, it can be safely included in the user's transaction as a state attestation.

Once support for a chain is added to the Rails network, paths for any token can be added permissionlessly. Since there is no need to bootstrap external liquidity, new paths can be used immediately after creation.

Intent frameworks can use Rails for efficient settlement. Current solver networks settling cross-chain intents require solvers to fragment their liquidity across many supported chains and receive payment on the opposite chain from where the intent was executed [1]. This means that all solvers must rebalance their own liquidity on top of solver duties. Rails allows solvers to settle in place, enabling them to focus on individual chains instead of rebalancing assets. After executing an intent with Rails, solvers take immediate ownership of a claim on the same chain. When the claim is freed, the liquidity can be used again without any rebalancing required.

Chain abstraction experiences can also be built on top of Rails. Just like any other bridge, Rails can be used in the background to move assets to the destination where the primary transaction is being executed (e.g. bridge and swap, bridge and deposit.)

5 Comparing Efficiency and Risk

Maximum throughput and attack surface area are two useful dimensions to compare the capital efficiency and security of Rails to classic bridge designs such as liquidity pool bridges, atomic swap bridges, and trusted bridges.

Both liquidity pool bridges (Hop Protocol v1, Connext, Across v1) and other message-settled bridges (including ERC-7683 [2] or "Intent Bridge" implementations) require, at a minimum, all bridge volume to be locked for the duration of the message settlement. Liquidity pool bridges require passive liquidity providers, but allow active liquidity providers to settle on the chain where the funds were provided. Atomic swap bridges do not require passive liquidity providers but active liquidity providers settle on the opposite chain from where funds were provided and must manage the rebalancing of their liquidity individually. Both of these bridge designs are limited in throughput by the message time, during which liquidity is locked. Their maximum volume processed within a given duration can be calculated as a function of the amount of liquidity and the message time. Duration is assumed to be a multiple of message time.

$$MaxVolume = Duration * Liquidity/MessageTime$$

Trusted bridges have no such limitation since message times are near instant. However, they must introduce a trusted intermediary greatly increasing the attack surface area. Over two billion dollars have been stolen from trusted bridges [3]. The reason security incidents have been so frequent and severe for trusted bridges is simple; it's much harder to secure active off-chain keys than it is to secure an immutable onchain protocol. Despite this existential risk, trusted bridges can process a virtually infinite amount of cross-chain volume in a given duration (limited only by block times and sizes) because they lack the capital lockup period of the message settlement required by current trustless bridge designs.

Rails is fully trustless but achieves the best-case capital efficiency profile currently only offered by trusted bridges. Each transfer sent frees up an equal amount of locked liquidity in the system making each transfer's net impact to locked liquidity zero. An infinite amount of volume can be processed in a given duration (limited only by block times and sizes) without the additional attack surface area that trusted bridges expose.

6 Conclusion

Rails introduces a novel approach to cross-chain liquidity settlement that decouples bridge settlement from the underlying messaging protocol, significantly improving capital efficiency without compromising on security. By leveraging state attestations and a virtual AMM for flow balancing, Rails reduces costs and lowers barriers for cross-chain liquidity. Furthermore, by incorporating an open network of bonders, Rails achieves faster-than-finality execution for a best-in-class user experience. As rollups continue to proliferate, Rails offers a scalable, trust-minimized, and efficient solution for cross-chain liquidity settlement, creating a more user-friendly, unified Ethereum.

References

[1] Uniswap Labs. Uniswap labs and across propose standard for cross-chain intents. `https://blog.uniswap.org/uniswap-labs-and-across-propose-standard-for-cross-chain-intents`, 2024.

[2] Nick Pai Mark Toda, Matt Rice. Erc-7683: Cross chain intents. `https://eips.ethereum.org/EIPS/eip-7683`, 2024.

[3] Chainalysis Team. Vulnerabilities in cross-chain bridge protocols emerge as top security risk. `https://www.chainalysis.com/blog/cross-chain-bridge-hacks-2022`, 2022.