NITRO by router whitepaper

Abstract

In recent years, the blockchain ecosystem has witnessed a proliferation of layer 1 and layer 2 networks designed to address Ethereum's scalability constraints. While these networks have succeeded in reducing gas costs and improving transaction throughput, they have inadvertently led to the dispersion of liquidity across various chains rather than concentrated on a single network like Ethereum. The fragmentation of DeFi liquidity across these disparate chains hinders the efficient utilization of assets. Towards this end, this paper introduces Router Nitro, a groundbreaking cross-chain bridge focusing on the specific use case of asset transfers. Built using the application-specific bridging construct enabled by the Router chain, Nitro strikes a perfect balance between cost, security, and latency. In addition to its capabilities as an asset-transfer bridge, Router Nitro also allows for arbitrary instruction transfers along with funds, thereby extending the composability of DeFi across various blockchains. This paper delves into the architecture, workflow, and features of Router Nitro, offering insights into its potential to resolve the challenge of fragmented liquidity and revolutionize cross-chain bridging in the evolving landscape of blockchain technology.

1. Introduction

1.1 Beyond Ethereum

The advent of Ethereum ushered in a revolutionary era for decentralized finance (DeFi). As the world's first Turing-complete trustless blockchain, Ethereum fundamentally transformed our perceptions of digital currency, global transactions, and decentralized applications [1]. However, the meteoric rise of the Ethereum ecosystem also unveiled underlying constraints with regard to scalability. Ethereum's elevated gas costs rendered it prohibitively expensive for the majority of market participants, restricting its accessibility and usability.

To surmount the constraints inherent to Ethereum and, consequently, bolster the DeFi sector at large, the market swiftly embraced diverse scaling solutions, including alternative layer 1 and layer 2 blockchains. While these solutions effectively addressed scalability issues, their introduction brought forth an equally critical concern - liquidity fragmentation.

1.2 Liquidity Fragmentation

In an ever-expanding landscape where numerous protocols contend for supremacy in the emerging Web 3.0 paradigm, none have unequivocally claimed the dominant position. This situation has left the DeFi sector grappling with the prospect of a fragmented market marked by widely dispersed liquidity pools and developer communities. To put things into perspective, the DeFi sector presently encapsulates over a trillion dollars in value, spread across an expansive crypto ecosystem encompassing over 8000 cryptocurrencies [2]. These assets operate within isolated infrastructures that lack interoperability, rendering them unable to harness the liquidity resources offered by one another. As liquidity fragments across these isolated pools, it becomes increasingly inefficient and cumbersome to utilize, thereby losing its utility. To address this scenario, there is a growing requirement for a mechanism that can port liquidity across blockchains and increase capital efficiency in the system. Moreover, as more and more institutional investors enter crypto, the need for better efficiency and flexibility grows almost exponentially.

1.3 Cross-chain Bridges to the Rescue

Without a robust mechanism that extends the composability of DeFi and promotes dynamic liquidity migration across various blockchains, there will be a cap on current as well as upcoming L1s/L2s achieving their full potential. The self-evident and almost natural solution to the problem of liquidity fragmentation is cross-chain bridges. Cross-chain bridges can enable liquidity to flow seamlessly between blockchains and increase capital efficiency in the system. Such bridges can operate between two blockchains, between a blockchain and a side chain, or even between two side chains. This interoperability allows the transfer of tokens, data, and even smart contract instructions between independent platforms.

2. Existing Solutions

2.1 Optimistic Bridges

Optimistic verification of cross-chain requests is another technique that has gained much traction. It is one of the more secure approaches to interoperability. Optimistic bridges are considered trustless because they require only one honest node to watch the network to ensure no malicious activity occurs. However, the foundation of its core competency also gives rise to its most significant drawback - high latency. Any cross-chain request sent via an optimistic bridge cannot be executed with instant finality, i.e., applications/users will have to wait for the challenge period to end before their request is marked as completed.

2.2 Light Client Nodes

A light client can be defined as a smart contract that parses source chain block headers on the destination chain. Since a light client node maintains a record of historical block headers, it can verify that a particular event has occurred on the source blockchain. In a light-client-based bridge, external actors named relayers forward events from the source chain to the destination chain, including block headers, state proofs, and other relevant data. Following this, the source chain's light client node on the destination chain cross-references its records to verify that a particular event was recorded on the source chain before executing a corresponding action on the destination chain.

The main advantage of bridges with the light client approach is that they do not introduce any trust assumptions. Even though a third-party actor forwards an event from the source chain to the destination chain, light clients can independently validate the event's existence using the block headers it holds.

However, light-client-based approaches also suffer from a few issues:

1. Light client nodes can be incredibly expensive to operate: Due to the costs associated with executing gas-intensive validation logic, updating block headers is costly, especially for light clients running on Ethereum [3]. To combat this, some bridges batch these block headers before relaying them to Ethereum, which can be problematic for time-sensitive applications. For example, Rainbow bridge, one of the most popular implementations of a light client node, batches Near block headers and sends them to Ethereum after 12-16 hours. This can lead to long waiting times when bridging assets from Near to Ethereum.
2. Adding a new chain to the mix is resource-intensive: For every new chain, (a) a new light client has to be deployed on all the existing chains, and (b) light clients of all the existing chains need to be deployed on the new chain [4].

2.3 PoA Bridges

Proof of Authority (PoA) bridges operate through a limited group of external entities responsible for monitoring events on the source blockchain, validating them, and relaying them to the destination blockchain. In order to maintain the integrity of these actors, PoA bridges typically incorporate incentivization and slashing mechanisms. For the most part, PoA solutions work well:

- **Efficiency and Low Latency**: Due to the involvement of only a few validators, PoA bridges can reach consensus relatively quickly, resulting in lower latency for transferring funds and messages compared to the optimistic approach.
- **Ease of Chain Integration**: PoA bridges excel in integrating support for new chains. Only a smart contract deployment and a tweak in the validator configuration to subscribe to events from the newly supported chains are required.

Despite their effectiveness, PoA bridges have one notable limitation – they operate on a trust-based model rather than a trustless one. Users of PoA bridges are required to place their trust in a federation of third-party validators. The relatively small number of entities involved in a PoA system, compared to a Proof of Stake (PoS) system, introduces a risk of collusion. In cases where a majority of these authorities act dishonestly, they can manipulate the system to their advantage at the expense of end users, potentially compromising the security of the bridge.

3. Beyond Technical Limitations

Besides the technical shortcomings of the existing asset transfer bridges, they face a few other challenges that render them inefficient for mass-scale adoption:

3.1 Dependence on Unsustainable Liquidity Mining Campaigns

- **Issue**: Many existing bridges heavily rely on liquidity sourced from users, often through unsustainable liquidity mining campaigns.
- **Implication**: As the protocol bears the cost of these campaigns, there is a risk of depleting the project's resources and negatively impacting its long-term sustainability.

3.2 The Honey Pot Problem

- **Issue**: Most current bridges lack the maturity required to accommodate extremely large Total Value Locked (TVL).
- **Implication**: The inability to handle substantial TVL not only poses a direct risk to the bridge's security but also endangers the integrity of the entire liquidity base in case of security breaches in other bridges connecting the same blockchains [5].

3.3 Latency Issues

- **Issue**: Existing asset transfer bridges have yet to overcome the latency challenge. Even with Proof of Authority (PoA) bridges, a standard cross-chain transfer takes 2 to 5 minutes.
- **Implication**: This latency can hinder the development of low-latency applications, such as cross-chain liquidation engines and cross-chain perpetual aggregators. The delay in cross-chain transfers restricts the real-time execution of critical transactions, affecting the performance and responsiveness of DeFi applications.

4. Intro to Nitro

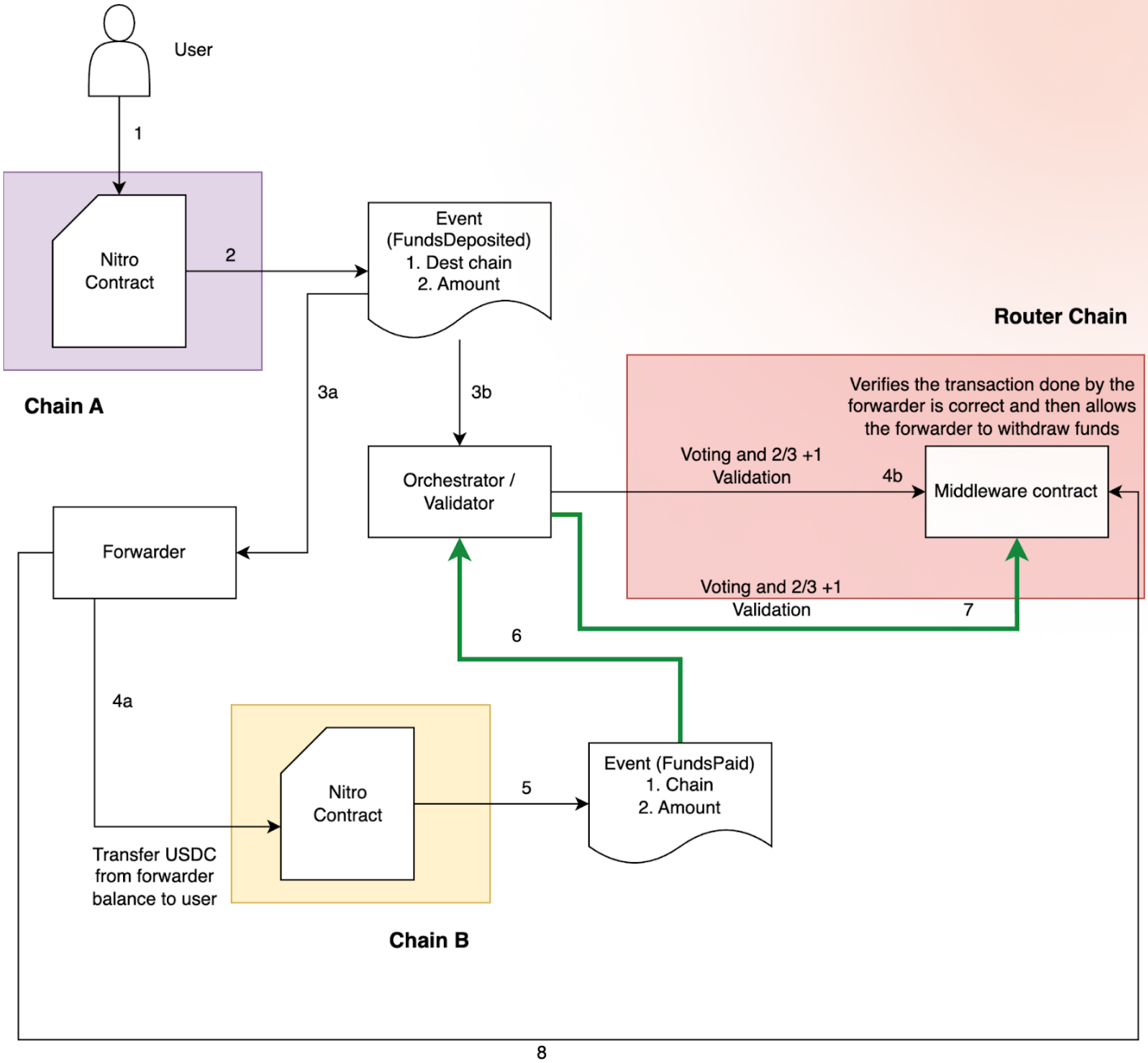
Until now, we have discussed a key problem plaguing the DeFi ecosystem and how certain technologies like optimistic bridges, light client nodes, and others are actively trying to solve this problem. Although these technologies have made significant strides toward building bridging infrastructures, they suffer from critical issues that impede their usability. To this end, in this paper, we introduce Router Nitro, an ultra-low latency trustless swapping engine that allows for cross-chain asset transfers as well as cross-chain sequencing of assets and arbitrary instructions. Nitro leverages the Router chain's middleware capabilities; it is mainly powered by an application-specific bridge contract deployed on the Router chain.

Router Nitro uses a trustless approach to handle cross-chain asset transfers. In this approach, an entity called the forwarder provides the users with their desired asset on the destination chain. Once the forwarder's settlement on the destination chain is verified, it can claim the funds deposited by the user on the source chain. This approach is secure and highly optimized in terms of latency and cost involved (for smart contract operations). More details about this approach are mentioned in the sections that follow.

5. High-level WorkFlow

5.1 Reverse Verification Flow

Step 1) User invokes the Nitro contract to transfer funds from Chain A (source) to Chain B (destination).



Step 2) The Nitro contract will validate the request, deduct funds from the user's account, increment event nonce and emit a **CrosschainTransferRequest** event. The event includes the following details:

- **SrcChainId**
- **EventNonce**
- **DestChainId**
- **ReceiverAddress**
- **Token**
- **Amount**

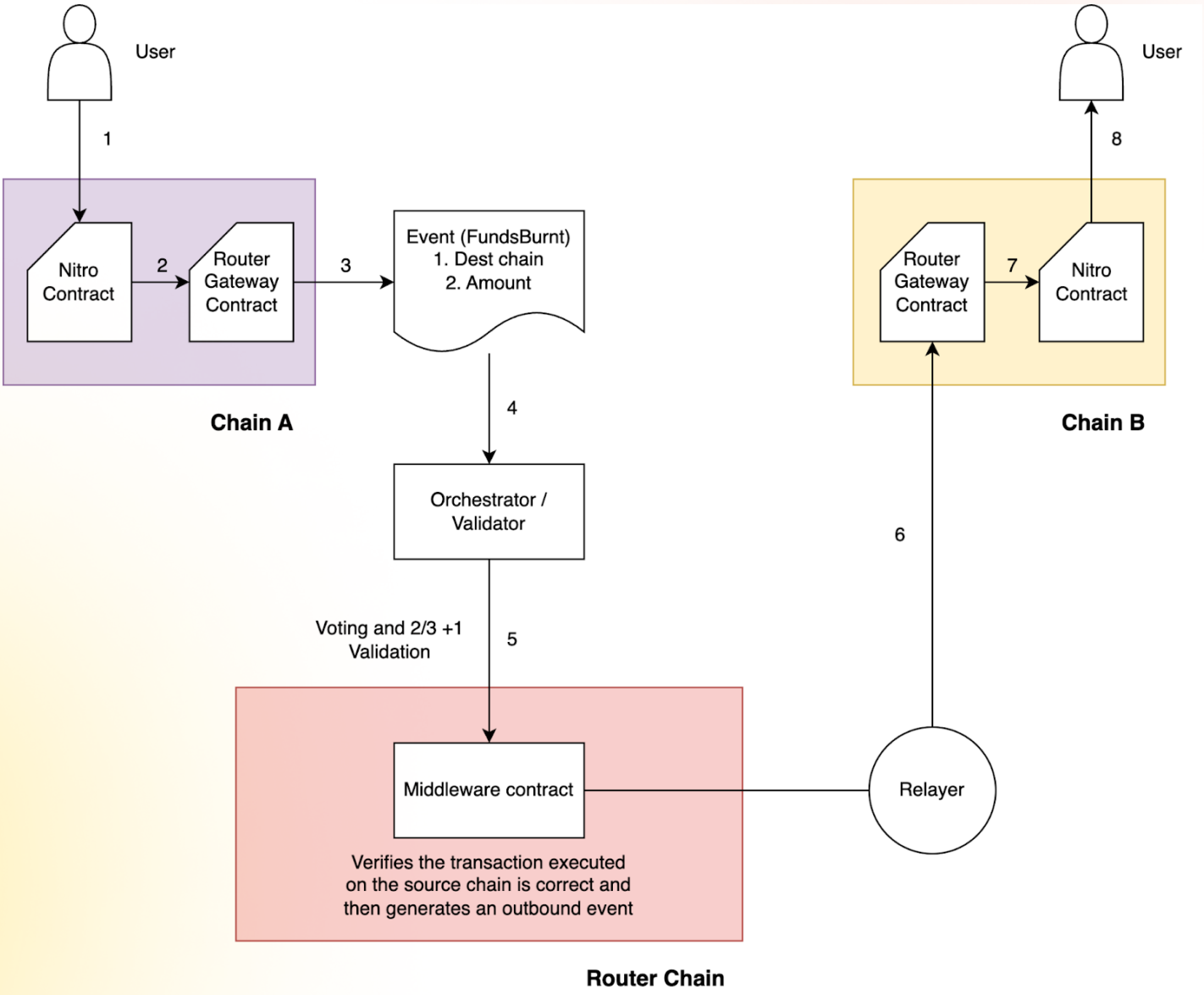
Step 3a) A forwarder will listen to the **CrosschainTransferRequest** event.
Step 3b) Orchestrators on the Router chain will also listen to the **CrosschainTransferRequest** event and submit it to the Router chain with their attestation.
Step 4) The forwarder will submit a transaction to the destination chain Nitro contract.
Step 4b) After 2/3+1 validation, the Router chain will invoke the middleware Nitro contract with the event info. Upon receiving the **CrosschainTransferRequest** event, the middleware contract will persist the request (mapping to the request will be maintained using the hash of the fields included in the source event).
Step 5) Upon receiving the tx, the Nitro contract on the destination chain will (a) transfer the defined amount from the forwarder address to the receiver address; (b) create a hash of the fields included in the request and persist the hash in the status map (to skip the replays); (c) emit a **CrosschainTransferExecuted** event confirming the execution. The event includes:

- **ChainId**
- **EventNonce**
- **hash**
- **ForwarderAddress**

Step 6) Orchestrators on the Router chain listen to the **CrosschainTransferExecuted** event from the destination chain Nitro contract and submit it to the Router chain with their attestation.
Step 7) Upon 2/3+1 validation, the Router chain will invoke the middleware Nitro contract with the event info. Upon receiving the **CrosschainTransferExecuted** event, the middleware contract will mark the request as **Completed** and persist the forwarder address and amount.
Step 8) Once the request is marked as completed, the forwarder can claim its funds (along with its incentive) by triggering an outbound request on the Router chain. The outbound request gets processed via the Router chain and existing Gateway contracts. The Gateway contract will invoke the source chain Nitro contract, which will settle the funds for the forwarder.
Note: To make the system more flexible for the forwarders, Nitro allows forwarders to claim their funds on any chain which has sufficient funds. For example, if a forwarder settled 500 USDC on Ethereum for a transaction originating from Polygon. They don't necessarily need to claim 500 USDC from Polygon, they can claim it from any other chain as well. While triggering an outbound request on the Router chain, they just need to specify the chain on which they want the funds.

5.2 Burn and Mint Flow

Step 1) User invokes the Nitro contract to transfer funds from Chain A (source) to Chain B (destination).
Step 2) The Nitro contract will validate the request, deduct funds from the user's account, burn them, and invoke the **iSend()** function on the Router Gateway contract.
Step 3) The Gateway contract on the source chain emits an event that is listened to by the orchestrators on the Router chain.
Step 4) Orchestrators on the Router chain will listen to the **CrosschainRequest** event and submit it to the Router chain with their attestation.
Step 5) After 2/3+1 validation, the Router chain will invoke the middleware Nitro contract with the event info. Upon receiving the **CrosschainRequest** event, the middleware contract will validate the request, generate an outgoing request from the Router chain to the destination chain, and pay the fees associated with the outgoing request.



Step 6) Once the outbound request initiated by the Nitro bridge contract is validated by the orchestrators, a Nitro relayer picks it up and forwards the event to the Router Gateway contract on the destination chain.
Step 7) Upon receiving the request from the relayer, the Gateway contract on the destination chain will invoke the Nitro contract.
Step 8) The Nitro contract will parse the message payload and mint the tokens to the user's address on the destination chain.
Note: This flow does not make use of the forwarders; therefore it is necessary to validate the request before it is executed on the destination.

6. Architectural Components

The Nitro cross-chain liquidity protocol comprises a set of integral architectural components that work in concert to facilitate the secure and seamless transfer of assets between different blockchain networks. Each component plays a distinct role within the Router Nitro ecosystem:

6.1 Nitro Application Contracts

These are smart contracts deployed on various blockchain networks, acting as intermediaries connecting forwarders and end users of the application. Users must invoke a Nitro application contract on the source chain to initiate a cross-chain request. Forwarders must invoke a Nitro contract on the destination chain to settle the transaction. On the source chain, a Nitro application contract deducts the user's deposit. On the destination chain, it facilitates the transfer of funds from the forwarder's wallet to the user's wallet.

In case of the burn and mint flow, the Nitro application contract on the source chain deducts the user's tokens, burns them and sends a cross-chain request to Router Gateway's contract. Once the request is received on the destination chain, the Nitro contract will mint the token and send them to the user's address.

6.2 Nitro Middleware Contract

The Nitro middleware contract is deployed on the Router chain and encompasses the necessary logic for verifying the requests and settling the funds for the forwarders. This contract persists cross-chain requests using the hash of the fields present in the event emitted from the source chain. Upon successful settlement of the request on the destination chain and validation of the corresponding **CrosschainTransferExecuted** event by orchestrators, the Nitro middleware contract marks the request as **Completed** and stores the forwarder's address along with the transferred amount. Once the request is marked as **Completed**, the forwarder can claim its funds and fee by invoking this contract.

In case of the burn and mint flow, the Nitro middleware contract will validate whether the **msg.sender** of the request from the source chain is a valid Nitro contract. Post that, it will parse the incoming request and generate an outbound request to the destination chain with the necessary parameters.

6.3 Forwarders

Forwarders are permissionless entities responsible for listening to users' requests on the source chain and settling them on the destination chain. After successfully settling cross-chain requests on the destination chain, forwarders can claim the funds allocated for their services from the Nitro middleware contract. Nitro will maintain a set of forwarders operated by various third parties, effectively distributing the responsibility.

6.4 Router Gateway Contracts

As their name implies, Gateway contracts serve as the interface for the Nitro contracts to interact with Router's bridging infrastructure. Gateway contract functions include:

1. **iSend()** - The Nitro contract on the source chain can call its Nitro bridge contract on the Router chain by invoking **iSend()** on the Gateway contract with the relevant parameters.
2. **iReceive()** - The Nitro bridge contract on the Router chain can invoke the Nitro contract on the destination chain by submitting an outbound request with the relevant parameters. Relayers will eventually submit the outbound request to the destination chain by invoking the **iReceive()** function on the Gateway contract, which will subsequently pass the payload to the destination contract.

6.5 Router Chain Orchestrators

The Router orchestrators are entities that listen to incoming cross-chain requests from other chains, attest their validity, parse them into a unified format, and post them on the Router chain. In the lifecycle of a Nitro cross-chain request (via the reverse verification flow), orchestrators validate the **CrosschainTransferRequest** event emitted by the source Nitro contract, as well as the **CrosschainTransferExecuted** event emitted by the destination Nitro contract once the request is completed.

7. Features

Nitro offers a comprehensive set of features that collectively empower developers and users to seamlessly interact with multiple blockchains while ensuring the safety of assets and the efficiency of cross-chain transactions.

7.1 Trustless Security

Nitro ensures the safety of user funds by ensuring that the forwarder can claim them only once the transaction is successfully settled on the destination chain and the settlement transaction is verified on the Nitro middleware contract. This mechanism eliminates any incentive for forwarders to engage in malicious activities, providing users with robust security.

7.2 Additional Safeguards

In addition to the protocol-level security, we have two additional safeguards that fortify the robustness of the ecosystem:

7.2.1 Price Oracle For Depeg Monitoring

To safeguard against potential de-pegging of reserve assets on Router Nitro, we maintain a price oracle. This oracle actively monitors the prices of the reserve assets on the chain, promptly detecting any deviations or fluctuations that could indicate a de-pegging event. In the event of a potential de-pegging, Router Nitro will temporarily halt the operation of the affected chain. This proactive measure ensures that any anomalies in asset pegging can be addressed swiftly, mitigating risks and enhancing the overall stability of the protocol.

7.2.2 Community Pause Feature

Router Nitro introduces a community pause feature, enabling users to pause any chain on the bridge. This feature empowers the community to take swift action in response to potential security threats or anomalies. Users can invoke a chain pause function on the Nitro contract by staking a designated amount of tokens. If the pause action effectively prevents a security breach or hack, Nitro will reward the user with twice the staked tokens. This mechanism incentivizes Nitro's users to act in the best interests of the protocol's security and integrity.

7.3 Provision For Sequenced Transfers

Router Nitro facilitates composability by enabling the development of cross-chain applications that require both asset transfer and instruction transfer within a single cross-chain request. A good example of such an application is a cross-chain yield aggregator that can seamlessly transfer users' funds and issue instructions to stake them on a specific contract in a single transaction.

7.4 Provision For Cross-chain Swaps

In addition to facilitating seamless asset transfers between different blockchain networks, Router Nitro goes a step further by offering support for cross-chain swaps. Nitro leverages its sequenced transfer flow to enable cross-chain swaps. Suppose you wish to swap USDC on Polygon for USDT on Avalanche. When a swap request for USDC is generated on Polygon, a forwarder picks up the request and facilitates the transfer of USDC from Polygon to Avalanche. Simultaneously, Nitro prepares the swap calldata (for USDC to USDT conversion on Avalanche) and encodes it inside the message field of the request that is picked by the Router orchestrators. The Router orchestrators submit the request to the Nitro middleware contract, where it is verified. Post the validation of the request, the request containing the calldata is forwarded to the destination chain with the help of relayers. Once the request is received on the destination chain, the Nitro contract converts the USDC provided by the forwarder to USDT and sends it to the user's address.

7.5 Ultra-low Latency

Router Nitro stands out as one of the fastest asset transfer bridges, offering sub-minute transaction finality. This ultra-low latency empowers developers to create applications necessitating near-instant cross-chain interactions.

7.6 Low Cost

Router Nitro employs highly optimized smart contracts that result in minimal gas costs. The efficiency of these contracts ensures that users and developers can interact with various blockchains without incurring exorbitant fees. In addition to this, Nitro's fee structure is designed to be cost-effective. Compared to other bridges with high operational costs due to liquidity mining emissions, Nitro has a considerably low operational cost, allowing it to charge lower fees. This affordability makes Nitro a compelling choice for users and developers seeking cost-efficient cross-chain solutions.

7.7 Cancellable Requests

Incorporating a fail-safe mechanism, Router Nitro affords a time-based cancellation feature to safeguard user funds. If, for any reason, a user's request remains unattended by any forwarder, the user retains the option to cancel the request. When such a cancellation occurs, the user's funds are promptly returned to them on the source chain. This ensures that user funds are never immobilized, enhancing the peace of mind and flexibility of Nitro users.

7.8 Provision For Incremental Fee

Router Nitro recognizes the dynamic nature of cross-chain interactions and offers an incremental fee feature to empower users further. Even after initiating a request with a specific fee, users maintain the flexibility to increase the fee associated with their request. This feature is particularly valuable in scenarios where no forwarder has picked up the user's request. By allowing fee adjustments, Nitro ensures that users can enhance the incentive for forwarders to attend to their requests promptly, ultimately leading to more efficient cross-chain transactions.

8. Future Work

As Nitro continues to evolve, we are committed to expanding its capabilities and improving the user experience. Our future work will focus on the following key areas:

8.1 Connection to Non-EVM Chains

In addition to our current support for EVM-based chains such as Avalanche, Polygon, and Arbitrum, Nitro is poised to extend its reach to blockchains that do not operate on the EVM. This expansion will include prominent chains like Solana, Bitcoin, and Near, broadening Nitro's interoperability across a broader spectrum of blockchain ecosystems. By connecting to non-EVM chains, Nitro will unlock new possibilities for cross-chain applications and asset transfers. Users and developers will have the flexibility to interact seamlessly with a more extensive array of blockchain networks.

8.2 Destination Chain Refueling

To streamline the cross-chain experience for our users, Router Nitro will introduce a feature that allows users to choose whether they want to receive native gas tokens on the destination chain along with the assets they are transferring. This feature eliminates the friction in regards to sourcing the gas token and ensures that users can seamlessly transition their activities on the destination chain, further simplifying the user experience.

9. Concluding Remarks

In the ever-evolving landscape of decentralized finance and blockchain technology, Router Nitro emerges as a beacon of innovation, pushing the boundaries of what is possible with cross-chain asset transfers. Our journey has been marked by a relentless commitment to addressing the challenges that have hindered seamless interoperability for years.

With trustless security at its core, Router Nitro stands as a symbol of confidence for users and developers alike. The protocol's unique design ensures the safety of user funds while ensuring sustainable incentives for forwarders to operate with integrity. The provision for sequenced transfers extends the horizons of cross-chain applications, empowering developers to create more complex and feature-rich decentralized solutions. Nitro's ultra-low latency challenges the status quo, enabling near-instant transaction finality and real-time cross-chain interactions. In addition to its technical prowess, Nitro remains committed to cost-efficiency. With optimized contracts and a lean operational model, Nitro ensures that users and developers can experience cross-chain interactions without the burden of exorbitant fees.

At Nitro, we are committed to delivering a robust, user-centric cross-chain bridge. We look forward to continually enhancing Nitro's capabilities and ensuring that it remains at the forefront of blockchain interoperability, providing users and developers with the tools they need to navigate the increasingly interconnected world of blockchain technology.

References

[1] https://ethereum.org/en/whitepaper/

[2] https://coinmarketcap.com/?page=90

[3] https://www.okx.com/learn/blockchain-bridges-explained-how-crosschain-messaging-protocols-work#Decentralized-or-trust-minimized-bridges

[4] https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

[5] https://beincrypto.com/bridge-liquidity-paradox-water-under-the-bridge/


NITRO