Walker's Chainlink Project

Summary

Define the components of a cross-chain token bridge. It should support ERC20 tokens and allow any token author to register their token contract with the token bridge.

Approach

Create a token bridging software stack that will augment existing Chainlink decentralize oracle infrastructure. Node operators will want to run the new software because it generates additional income. The network gains new functionality without having to build out a lot of new infrastructure.

Assumptions

- Validator network will continue to be a permissioned while the software matures, but will eventually become more open.
- Prioritize security over speed.
- Responsibilities delegated in a way that makes each individual responsibility as simple as possible and therefore easier to get right. Also allows for security checkpoints.
- Which chains are supported will be determined by the governance consortium (permissioned).
- Any interface-compatible token on a given supported chain should be able to register with the bridge in a permission-less manner.
- There will be hooks to pause a given bridge, in the case of an emergency, but not the ability to sensor transactions.

Strengths and Weaknesses

Strengths

- Brand earns reputation as a secure platform.
- Existing infrastructure gives the network an advantage over competition.
- Choosing to play the careful game will pay off more in the long term.
- Balances trade-offs between permission and permission-less.

Weaknesses

- Won't be as fast as other networks that have looser security assumptions.
- Ability to add chains will be bottle-necked by development capacity of one company.
- Projects building on unsupported chains will have to use other bridging options.
- Perception that it is a permissioned network.

Requirements

- Support for ERC20 token standard
- User calls one smart contract function on the source chain, specifying the end address, then the rest is automatic
- Each step in the process has an API to query the status

- Nodes are paid in LINK
- Governance consortium manages the members of each decentralized node network

Nice to have

- CLI
- Web application to visualize the progress of a transaction

Personas

- Developers of new tokens or chains
- End-users who want to swap tokens cross-chain
- Node operators
- Chainlink developers who create stack
- Stakeholders of Chainlink network (governance)

Components

- Router contract, one deployed on each supported chain.

The source-chain router is the entry point for the user. They call their transaction on the router, and then the router will pass the transaction along to the right sub-contract. The destination-chain router terminates the transaction by initiating the final token transfer on the destination chain.

- Intermediate contracts

Behind the router, there can be several layers of contracts. The idea is to simplify the role of each contract to make them as easily auditable as possible. Layers could include the following. Chain-specific coordinators, which have parameters suitable for their respective destination chain, or parameters for controlling limits and other QoS considerations. Token pools, which are used as the escrowing agent on the source side, whether through locking or burning, or the supplying agent on destination side, through unlocking and minting.

- Node operators

There are several different node roles. It is possible that one computer can perform multiples roles, but often operators find that they gain an advantage when specializing. The node roles may also be split up into separate networks, so that there is a wider delegation of oversight. Roles include the following:

1. Blockchain full-nodes that maintain the blockchain for their given chain. These are used to read and write transactions to the blockchains.
2. Transaction discovery nodes that subscribe to source chain events, then pool those events into a transaction for the destination chain.
3. Validator nodes that monitor both source and destination for irregularities. These nodes can pause operation on specific parts of the system if necessary.
4. Finality nodes that monitor the destination chain for transactions that are posted by the discovery nodes and approved by the validator nodes.

- Token smart contracts

Each token has its own contract on each source and destination chain, which are maintained by the token author, not Chainlink. These contracts keep track of the balances of their own users. These will support interface standards like ERC20, which will allow the router contract to move funds between accounts. These contracts will need to implement a minting function that is compatible with the authorizations that are provided by the finality nodes.

- SDK for developers
- Communications module for each blockchain that provides an common API to interact with full-node RPCs
- Governance contracts
- Module delivery system that nodes use to pull down the components that they need to run
- Bridging status interface
- Bridging front-end

Workflows

Deploying support for a new chain

Develop communications module that Chainlink infrastructure nodes use to interact with the RPCs of blockchain full-nodes and deploy on the module delivery system.

Deploy router and intermediate contracts to new chain.

Deploy a new intermediate contract to each existing chain. This will register itself as a new pathway with the router.

Update the website with new contract addresses.

There might need to be a testing mode, where only whitelisted addresses can use the new pathway, so that the new contracts can be validated properly. Upon completion of testing, the contracts are put into production mode, where anyone can use them.

Deploy support for a new token

Develop new token contract that optionally includes support for minting.

Deploy new token contract to two chains.

Register new token with the Chainlink router contracts on both chains.

Send a test transaction

End-user transferring cross-chain

Find the contract address for the token on the source chain.

Find the chain-id and wallet address on the destination chain.

Find the Chainlink router address on the source chain.

Submit a pricing request to router contract, to learn how much a given transaction should cost.

Submit bridging token transfer approval to token contract on source chain for Chainlink router to be a spender.

Submit payment token transfer approval, in case a non-native payment (like LINK) will be used.

Submit transfer request to Chainlink router, specifying the source token address, transfer amount, destination chain-id, destination wallet address, and payment method. Native token payment, if used, accompanies this transaction.

(A web front-end could combine several of those steps into one flow, simplifying the process for the user.)

Milestones

Alpha phase. Build a proof-of-concept that utilizes ethereum and one other chain. A user should be able to transfer LINK from one chain to the other.

Beta phase. Take the concepts from the proof-of-concept and build a starter SDK. It should be used by Chainlink to deploy its contracts to 4-5 additional chains. Make a list of the end-points that need to be modularized in order to allow for flexible insertion into a variety of chains, and sort the list into critical and nice-to-have groups. Build the critical items in this phase.

v1.0 phase. Refine the SDK based on experience thus-far. Start incorporating any of the nice-to-have features that represent important pain points. Possibly add a CLI.

v1.5 phase. Improve documentation and user experience. Plan for v2.0.

v2.0 phase. The significant upgrade that possibly starts to decentralize some fo the centralized components.

Questions

What entities have wallets? - Relayer needs to pay for transactions - Router contracts are payable

What is the practical differences between wrapped and mint-and-burn architectures?

```
- Both require a contract on each chain.

- Wrapped has a sort of double-entry bookkeeping feature.  The number of
wrapped should match the number of locked.

- Mint-and-burn would have a way to compute the total number of tokens
issued in the main chain, compared to the total active on the main chain
and then the quantity on each sub-chain.
```

What makes native assets harder than tokens?

How does Chainlink add support for new chains in the node software?

How do you restore tokens that were stolen?

How do you distribute keys?

How do you disincentivize bad behavior in nodes? Staking?

How are fees paid and distributed?

What tokens can be used to pay fees?

Do nodes need an interface with an exchange or bridge?

How is the single relayer chosen?

Is the minting algorithm provided in the SDK? Is it overridable?

Does the system support or rely on fraud proofs?

Does the system require staking?

Do signatures need to be stored on chain?

Instructions

Opportunity

1. Users send tokens to a smart contract for locking on the “home” chain (Ethereum mainnet), providing a destination address on the “remote” chain.
2. Through a mechanism, the tokens should be bridged and then sent to them on a remote chain.

The bridge should be built such that it can support any ERC20 token, and ideally, require as little work as possible to scale and repeat across blockchains. Minimizing trust and single points of failure is also a positive goal.

Objectives

1. Gather requirements and present a potential approach for solving the above problem. Please confirm any assumptions and discuss the strategy with the hiring manager. Present the strengths and weaknesses of your approach.

2. Produce a 3-5 page specification presenting the a. Product requirements, b. Components (services, smart contracts, tools etc) of the solution. i. Note: Define components at a high level, no requirements to produce any code. c. User personas and stakeholders d. Various workflows and application behavior for each user / stakeholder. This should also involve creating an operational process around setting up / releasing the components.

3. Produce a list of milestones for the project. For each milestone, please state the acceptance criteria. Please be as specific as possible.