Walker's Chainlink Project

Notes on approach

Having used Axelar to bridge USDC across several chains, I understood what a token bridge should accomplish and how it should present a front-end. I read up on Wormhole and its exploit. I made a first draft of this project and then I read up on CCIP, which has just been opened to a general audience. CCIP is a solution to this project, which I found enlightening but also distracting. I understand this project has been used since before CCIP was a thing. I tried not to just copy the construction of CCIP, but it is very clear why some of the design choices are the right ones, like separating the Committing, Risk Management, and Executing nodes. I thought about areas where CCIP could be tweaked, like how permissioned the node sets currently are and how adding new blockchains and tokens could be made more self-serve. I found little documentation about these topics, and haven't yet dug through the code repos. I also wonder about the minting/locking side of adding a new token on a supported chain. How will the destination token pool validate that an incoming request from the bridge network is valid and authorized? This seems to be the cornerstone of the whole system, and yet it is never really clarified in the documentation. I have also not specified it to any extent in this project, besides to note its existence. 3-5 pages is quite a compact space to describe the requirements and rollout details for a complex project like this. The way I lay out the milestones is lighter on details than I would normally produce for a deployment plan, owing to the fact that I would want to specify the requirements in greater detail before outlining a detailed deployment plan, but I have summarized some of the deliverables.

Summary

Define a cross-chain token bridge system. It should allow any token author to register their token contract with the token bridge. Tokens must support the ERC20 interface as well as a specification for the mint/unlock function, so that the bridging contract can trigger token creation/release on the destination contract. Support for individual blockchain is added using a permissioned approach governed by a consortium.

Terms

Bridge Governance Consortium (BGC) -- the stakeholders of the bridging system who retain authority of several key components, including system parameters and membership in the various bridging network node teams.

Token author -- the owner/author of a given token contract. Responsible for deploying these contracts to the blockchains over which bridging is desired. Also responsible for calling the token register function on each respective router.

Node team -- a group of nodes who are doing performing a given task, like delivery, validation, and finality. The node's interactions regarding the bridge are validated by a consensus protocol.

Permissioned node team -- a teams whose membership is controlled by the BGC entity.

Permissionless node team -- a node team whose membership is governed by a smart contract or consensus mechanism, rather than an entity.

Token contracts -- owned by a third party, for example USDC, Wrapper Ether, or any other ERC20.

Bridge contracts -- maintained by the BGC, these are the contracts that handle the routing and intermediate functions like token pooling for the bridge.

Supported blockchain -- a blockchain that is supported by the bridging network. It will have bridge contracts deployed to it and a blockchain communication module implemented so the bridge-nodes can communicate with the blockchain's full-nodes or indexing service.

Source chain -- the blockchain from which a swap request originates.

Destination chain -- the blockchain that mints/unlocks the token at the receiving end of a swap transaction.

Minting/unlocking -- Two different mechanisms for releasing tokens on a destination chain. The token contract must provide an implementation for one of these methods that is compatible with the type of attestation that the bridge contract will provide. The bridge will instruct the token contract when to mint/unlock tokens in the case of a swap request.

Personas

Token authors

Token authors want their token to be transferrable across chains. They expect deployment on the bridge to be easy. They will have varying security requirements, but will generally value security as a top consideration.

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.

End-users

End-users want to swap tokens cross-chain. They want a reliable and trustworthy service at a competitive cost.

The end-user may need to manually submit the last leg of the transaction on the destination chain, in the case that the finality nodes were not able to successfully complete the transaction after a period of time. This could occur in situation of network congestion, where enough gas was not provided with the original transaction.

Node operators

Node operators join node teams and are paid in LINK by the bridging system for the services they perform. They must bond LINK to incentivize good behavior and secure their spot in the consensus hierarchy.

- Blockchain full-nodes

Blockchain full-nodes are not actively a part of the bridging consensus, but access to these is required by all of the node types in order to query and transaction on source and destination chains.

- Bridge-transaction discovery

Gathers a batch of bridge transactions from source chain, and pools them into a single attestation on the destination chain.

- Attestation validation

Validates transaction attestations on destination chain by monitoring both source and destination chains for anomalies. Can submit fraud notification that can pause the network or trigger slashing or other consequences to offending nods.

Can anyone who is properly staked be a validator or should it be a permissioned network?

- Transaction finality

Submits the actual bridge transaction on the destination chain, once the transaction attestation has been validated. Pays gas on destination chain. In times of congestion, will retry a number of times.

BGC developers

These developers write the bridge software, including the node application and the bridge contracts. They are responsible to fix problems in the code. They want to develop a code base that balances modularity and simplicity, both so that it is easy for them to add support for new chains and new features, and so that their work can be more easily audited.

BGC stakeholders

These stakeholders are responsible for the decision making and delegation of components of the bridge. The group starts out as a company, foundation, or consortium, but likely becomes a DAO, using governance tokens to gate membership. They decide on membership in the various permissioned node teams. They direct the funding for development and marketing. They also determine various system parameters like fee rates.

Attackers

Attackers try to attack the network in various ways. They can operate malicious nodes in an attempt to block service or submit fraudulent transactions.

Attackers may also attack the smart contract surface area, by submitting transactions that exploit bugs or spoof/bypass safety checks.

Regulators

Regulators are governmental agencies who will most certainly approach the controlling authorities of the bridge in order to request information and potentially control or block parts of the service. The more decentralized the governance structure is, the more resilient the network will be.

Approach

Create a token bridging software stack that will augment or expand existing Chainlink decentralized oracle infrastructure. The network gains new functionality without having to build out a lot of new infrastructure.

- Prioritize security over speed.

- Some node teams will be permissioned while the software matures, but will eventually become permissionless.
- Delegate or layer responsibilities in a way that makes each responsibility as simple as possible and therefore easier to get right. This also allows for security checkpoints.
- Control which blockchains are supported. BGC determines which blockchains are supported.
- All any tokens on supported blockchains to be added. Any interface-compatible token on a given supported chain should be able to register with the bridge in a permissionless manner.
- Be flexible in ability to respond to emergencies, but don't compromise on user sovereignty. Provide hooks to rate-limit or pause a given bridge, but not the ability to sensor transactions.
- Support ecosystem development with documentation and SDKs for each of: blockchain owners, token owners, and end users.
- Inspire usage through a reference web interface for interacting with the bridge.

Assumptions

- There is demand for cross-chain bridging that users are willing to pay for both in terms of cost and time.
- End-users will prefer this bridge because they know and value the Chainlink brand and its commitment to security.
- Node operators will want to run the new software because it generates additional income.
- The majority of demand will come from only a handful of chains, so minor chains can be ignored for now.
- The given set of requirements addresses the largest potential pool of demand.
- Competitors will exist who are more decentralized and more user-deployable or new-chain-friendly, but they are chasing a smaller portion of the pie. Our roadmap will eventually intersect with theirs, and we will have greater momentum from our initial approach, which will allow us to continue to grow market share.

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 BGC
- Projects building on unsupported chains will have to use other bridging options.
- Some perception that it is a centralized network.

Requirements

- Support for tokens using the ERC20 standard.

- User calls one smart contract function on the source chain, then the rest is automatic.

- Each node in the system provides an API to query the status.
- Nodes are paid in LINK.
- Bridge supports combinations of lock/unlock and mint/burn token custody strategies.
- SDK for blockchain owner, token authors, and end-users

Extras

- Web application

Gives end-users and easy way to submit transactions.

- Web application

Gives end-users an easy way to visualize the progress of a transaction

- CLI

Gives technical end-users a tool for constructing transactions in a more programmatic and customizable way.

- Governance contracts

Eventually when BGC becomes a DAO, there will need to be governance contracts that direct its operation.

- Application/module delivery mechanism

Automatically installs bridge applications and blockchain modules that bridge-node operators need for their given configuration (which chains and node-roles they are supporting).

Components

Router contract

One deployed on each supported blockchain. 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 intermediate 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. Layers simplify the role of each individual contract, making them more easily auditable. Layers include at least these two.

1. Chain-specific coordinators, which have parameters suitable for their respective destination chain, or parameters for controlling limits and other QoS considerations.
2. Token custody, 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 or minting.

Bridge-node applications

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, and the network will benefit when there are a diversity of physical operators. Each node role is fulfilled by a team of nodes. Roles include the following:

1. Transaction discovery

The transaction discovery application takes in source chain events, pools those events into a single transaction attestation, and then submits that attestation onto the destination chain.

2. Attestation Validation

The validation application monitors both source and destination for irregularities. A node running validation software signs off on valid transaction attestations. Upon finding an invalid transaction attestation, the application can submit a fraud notification, which could pause a given component of the network, or trigger slashing or other consequences to nodes associated with the transaction.

3. Transaction Finality

The finality application monitors the destination chain for transaction attestations that are posted by discovery and approved by the validation. Once all approvals have been committed to the chain, the finality application submits a request to the router contract, which will trigger the token transfer from token custody to the destination address.

This application needs to pay for gas on the destination chain. It will therefore need to have a wallet with funds which have either been pre-funded or are funded as an automatic process of converting its LINK proceeds into native token through connection with an exchange.

Bridge-node Consensus

TBD. Build off of existing Chainlink consensus used in DONs.

Blockchain Communication Module

These modules provide a common API for accessing blockchain-specific RPCs provided by full-nodes or indexing services. There will need to be at least one module for each unique blockchain interface. Each bridge node operator will utilize the modules for the blockchains and services that they require to communicate with the blockchains that they are supporting.

Blockchain Configuration Module

A set of configuration parameters that describe properties like finality, address format, data limitations, native token details, etc. This could be used by various bridge contracts and nodes to validate incoming and properly form outgoing messages.

Token smart contract interface

Each token has its own contract on each source and destination chain, which are maintained by the token author, not BGC. These contracts will need to implement minting/unlocking functions that are compatible with the authorizations that are provided by the bridge nodes.

SDK for developers

Blockchain developers need an SDK that will help them build the blockchain communication and configuration modules, so that they chain is ready to be put in the support queue.

Token authors need an SDK that provides the lock/unlock/mint/burn base functionality that will be compatible with the requests made by finality nodes, which includes validating signatures from the various bridge nodes (discovery, validation, finality).

End-users need and SDK that provides an example contract that they can use and their own interface into the bridging network. This would serve to both send and receive tokens on behalf of the user. The user can integrate this contract into their other applications.

Workflows

Deploying support for a new chain

Develop communications and configuration modules that bridge nodes use to interact with the RPCs of blockchain full-nodes. Deploy on the module delivery system (if available).

Deploy router and intermediate contracts to new chain.

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

Update the website with new contract addresses, chain-id, etc.

Consideration: 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 includes support for minting/unlocking.

Deploy new token contract to two or more blockchains.

Register new token with the bridge router contracts on each chain.

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 bridge router address on the source chain.

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

Submit swap-token transfer approval to token contract on source chain for 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 router contract, specifying the source token address, transfer amount, destination chain-id, destination wallet address, and payment method. Native token payment, if used, accompanies this transaction. User receives a transaction ID.

User can manually query bridge nodes and bridge contracts with the transaction ID to get status from each stage.

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

Validator submits a fraud notification

TBD

Finality node fails to send a transaction within a given window

TBD

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 and back. Users are whitelisted. The following components are in place in a first-draft form:

- Router contracts
- Intermediate contracts
- Bridge applications running on nodes
- Bridge node consensus operational for each node team
- Blockchain modules appropriate for each supported blockchain
- LINK token contracts on both blockchains

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 1-2 additional chains. Add support for a stable coin like USDC.

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.

Add web interfaces for submitting transactions and viewing transaction status.

Increase size of whitelist user set.

- Modularize the end-points deemed critical
- Deploy everything from alpha phase to 2-3 other chains
- Publish draft SDK that is used when adding stable coin support
- Release blockchain module for the new chain

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. Make of list of required top-tier chains to support in this phase. Expand token whitelist, allowing token authors to deploy their own contracts.

- Add support for all top-tier chains
- Add support for additional 5-6 tokens
- Website is up to v1.0 level

v1.5 phase

Improve documentation and user experience. Plan for v2.0. Allow any tokens to register with the bridge.

v2.0 phase

The significant upgrade that possibly starts to decentralize some of the centralized components.

Details not covered

How does the consensus mechanism work within node teams? How are keys distributed?

How do contracts in a destination chain validate incoming bridge requests?

How/when are nodes paid?

How/when do nodes that need to submit transactions top-up their wallets for gas fees?

How do new tokens register with the router contract?

How do new chains register channels with existing chains?

Fraud recovery. How do you restore tokens that get stolen from a bridge?

Other Questions

What is the practical differences between locking vs burning on the source side? What are the pros/cons of each?

> Does one offer better accounting?

In the case of the lock/unlock scheme, how are tokens on the destination chain locked in a pool so that they can be unlocked in a swap? This seems similar to a liquidity pool.

Instructions from Chainlink for this project

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.