Concero Whitepaper

Concero v1.0

Andy Bohutsky, Oleg Kron

Table of Contents

Abstract

Concero is a fully decentralised abstracted cross-blockchain protocol able to facilitate secure value and message transfers in under a minute. Existing cross-chain solutions have to trade security for speed and vice-versa while Concero has been built to be secure and quick at the same time, while being fully decentralised.

Before designing Concero infrastructure, we have spent a lot of time building, iterating and collecting user feedback with different cross-chain aggregation MVPs. This has been done in order to understand user requirements and solve problems that the end users are facing. After thousands of survey results and usage data points we have boiled down user requirements for cross-chain transactions to three main attributes: Speed, Security and Ease of Use. After trying to look for infrastructure that is able to adhere to these attributes, we found that it does not exist, which laid the foundation behind Concero Protocol

We have also set ourselves a strict set of parameters when building this protocol. It had to be fully decentralised, economically feasible for all participants, and fully transparent. As the entire blockchain industry was built on the core principles of transparency, decentralisation and permissionless access, we believe that it is imperative for protocols to adhere to these principles and Concero does just that.

1 Introduction

Although there are many cross-chain interoperability solutions available on the market right now, a lot of them are cutting corners with centralisation. This is a very dangerous trend that is seen on the market as de facto control of cross-chain transactions being executed is concentrated amongst a small group of people. This may lead to a plethora of problems, starting from abuse of vulnerabilities in centralised structures with malicious intents to censorship and dictatorial control by insiders.

With something so fundamental as cross-chain interoperability, it is imperative that the fundamental principles of the industry are adhered to in order to allow developers the freedom of building with minimal to no counterparty risk.

1.1 Functionality

There are four distinct functions of Concero Protocol that have been built in order to provide developers with all of the necessary tooling to build cross chain apps:

1.1.1 Value Transfer

Migrating (‘bridging’) assets between blockchains using an intent-based system with optimistic execution. Liquidity providers incur finality risk and are compensated with a fee for bearing this risk. Intents are verified and transmitted using our messaging functionality.

1.1.2 Messaging

Transmitting messages between blockchains using our execution layer. Messages are verified on the source blockchain and a message is transmitted to the destination blockchain using the decentralised compute of the execution layer and a messenger wallet. The incoming unconfirmed message triggers a check on the destination chain with the purpose of verifying the source chain message before releasing it on the destination chain. The check is in place to ensure that even if private keys of messenger wallets get leaked, the attacker is still not going to be able to confirm illegitimate transactions.

1.1.3 Gas/Fee Abstraction

The abstraction of destination chain gas and fees allows users to pay them in native token of the source chain or in one of the 'bridgeable' assets (in V1 $USDC) at the moment of bridging. As $USDC is the only supported bridgeable token, the user doesn't need to have it prior to the transaction, as it will be seamlessly swapped before bridging. Gas abstraction also allows users the freedom to not have destination chain gas in order to complete a cross-chain transaction, which plays a crucial role in frictionless user experience.

1.1.4 Single-Chain Execution

Responsible for executing single-chain transactions through our orchestrator contracts on both the source and destination chains. Users will be able to execute single or multi-step transactions with a single signature and within the same transaction. A single source-chain function call is also relevant for cross-chain transactions as part of the efforts to simplify the user experience. We anticipate various projects developing their own routing algorithms to create call-data (routes) for users based on their intents and building more complicated yield strategies that can be executed through Concero Protocol.

2 Technical overview

High Level Design Choices

1. Existing networks and infrastructure was chosen for message relaying and settlement. For the launch of V1, we have chosen Chainlink CCIP as a settlement layer and Chainlink Functions as the execution layer, given Chainlink's decentralised off-chain compute is sufficiently decentralised and reliable.
2. We have kept the infrastructure modular in order for new settlement layers and execution layer networks to be added in an easy manner as the protocol is developed and expanded further.

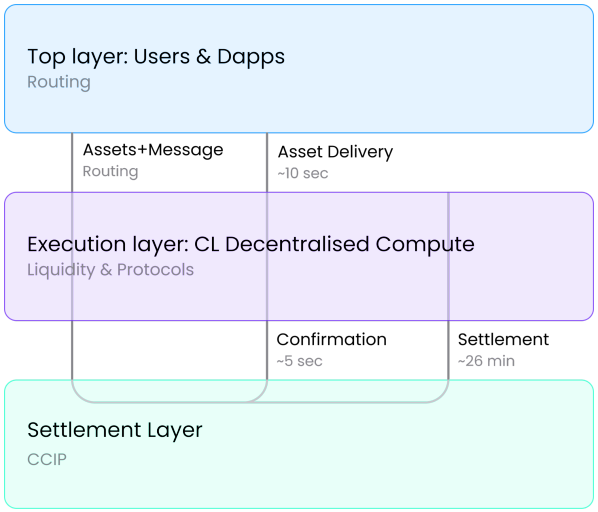
Reason for Design Choices

Building our own network from scratch would present a *chicken-and-egg problem*: the protocol needs volume and value to attract node operators, and it needs node operators to generate volume and value. Many other protocols combat this issue by setting up their own main nodes that account for a large proportion of the volume (this is inadequate due to centralisation risk) or by burning through capital to

incentivise node operators (very expensive and flawed approach). We see neither of these approaches as a viable option for Concero, hence we decided to use existing infrastructure to attract volume/value and expand the protocol further with the demand.

2.1 Concero Stack

Concero 'Stack' consists of three distinct layers:



2.1.1 Top Layer

Top Layer is where all the intents are collected and where all the apps and users connect to through our SDK, API and Applications. A direct on-chain interaction is also possible through Concero Solidity Dev Kit (Concero Endpoints).

2.1.2 Execution Layer

Execution Layer is an on-chain layer where all the orchestrator contracts (with proxies to other protocols), liquidity pools, execution networks (in V1 Chainlink Functions) and ancillary Concero smart contracts reside.

2.1.3 Settlement Layer

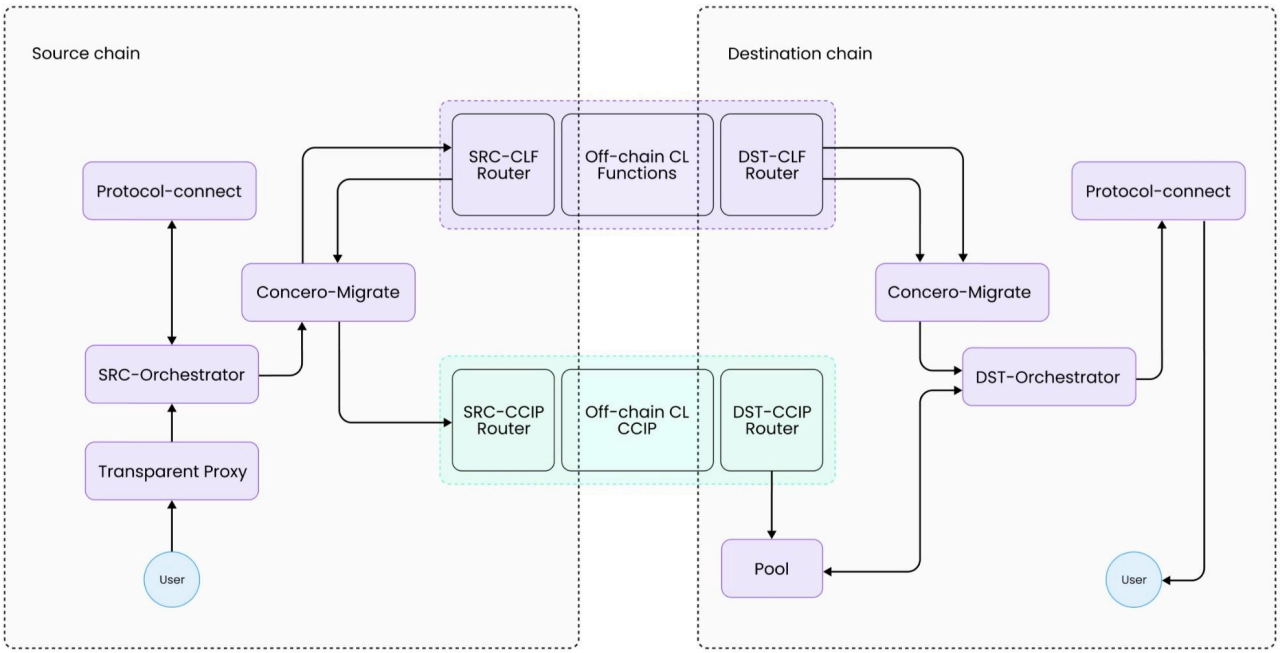
Settlement Layer is used to settle transactions that are executed by the execution layer and to rebalance liquidity and the entire system periodically (In V1 Chainlink CCIP).

2.2 Liquidity Provision

Liquidity Providers will be able to provide 'bridgeable' tokens (in V1 $USDC only) into Concero Liquidity Pools. This liquidity facilitates sub-minute cross chain transactions by optimistic loans that will be taken on the destination chain after confirming that the source chain funds are en route through the settlement layer. LPs will earn yield from user fees paid to use the Concero Protocol while assuming

chain reorg risk. Liquidity will be provided into a Parent Pool on the BASE blockchain and will be automatically distributed into Child Pools on all supported chains, ensuring that every liquidity provider is uniformly exposed to all of the fees collected by the protocol.

3 Infrastructure breakdown



Step-by-step breakdown:

1. User sends the assets to the contract on the source chain through a Transparent Proxy
2. Proxy passes the message to the orchestrator, which manages the flow of the transaction
3. The orchestrator performs a source-chain swap in order to obtain a bridgeable asset
4. The orchestrator calls the migration contract to migrate assets to the DST chain
5. The assets are sent to the settlement layer (CCIP), which triggers the execution layer
6. Source chain functions are triggered to put transaction in the queue on the destination chain
7. TX is put in the queue which triggers the check from the destination chain
8. Destination chain CL Functions ensures the transaction exists on the source chain
9. DST migration contract receives tx confirmation from the source chain via CL Functions
10. Orchestrator continues TX on the destination chain
11. Orchestrator obtains a loan from the Pool to proceed with TX
12. An optional swap is performed to obtain the destination asset
13. Asset is sent to user
14. Asset received from the settlement layer (CCIP) into the destination chain pool.

3.1 Single-chain Execution

The single-chain executor contract is responsible for interacting with protocols within the same chain. These include Decentralised Exchanges, Lending protocols, etc. For cross-chain transactions, in which the source token itself isn't bridgeable, the executor is responsible for transferring the source token into a bridgeable one. After the approval for token spending, the executor contract's entrypoint function takes an array of actions as a parameter and runs them on third-party protocols in order of their presence in the array. This allows to complete a complex multi-step transaction with a single function call.

3.2 Cross-chain asset migration

The Cross-Chain asset migration infrastructure of Concero consists of three key parts. The execution layer, responsible for fast-tracking the transaction en route, surpassing the chain finality. The settlement layer, a secure, finality-bound mechanism for sending assets across chains with additional security checks, provided by Chainlink CCIP. Finally, the liquidity pools, the liquidity of which is used by the execution layer for obtaining optimistic loans, after a minimum amount of block confirmations is obtained.

3.2.1 Cross-chain execution

Adds the transaction to the unconfirmed queue on the destination chain, which indicates that a cross-chain transaction has begun settlement on the source chain. Uses a threshold-encrypted messenger wallet private key hosted in Chainlink DON-Hosted secrets in order to add the transaction to the queue, passing through the list of allowed messenger addresses present in every contract. Despite the security & encryption of the DON-Hosted secrets, the incoming transaction requests are not trusted by default, and an additional check is performed. The destination contract, which receives the unconfirmed transaction from the authorised messenger, begins the confirmation process by triggering an additional set of Chainlink Functions from its own chain.

The purpose of Destination chain Chainlink Functions is to confirm whether the transaction has indeed been submitted to the source chain and if it has successfully entered CCIP. Using a randomised RPC provider inside the JavaScript code of Chainlink Functions, each of the 4 nodes performs an RPC call to read a specific event. This event confirms that the transaction has indeed entered the settlement layer (CCIP), thus indicating the beginning of the settlement process.

Additionally, the destination chain Chainlink Functions code holds responsibility for ensuring a specific number of block confirmations on a confirmed source-chain transaction before submitting a successful result to the destination contract. The minimum amount of block confirmations depends on the risk factor of each transaction that enters the bridging infrastructure.

Minimum number of confirmations for each chain can be found in our documentation at docs.concero.io

3.2.2 Specifics of Chainlink Functions as the execution layer

Chainlink Functions execution is initiated on chain, with the computation taking place off chain on a set of 4 nodes randomly picked from the Decentralised Oracle Network. Then, the consensus is made on each of the responses and submitted back to the initiating contract.

Runtime nonce clashing

The nature of a simultaneous code execution brings several unique problems, such as the Nonce clashing, which occurs when nodes are attempting to submit a transaction to the blockchain at the same time. In order to circumvent nonce clashing, the following measures are taken:

- **Randomised messenger wallets**: when signing a transaction, each node takes a randomly picked wallet from the messengers list, which significantly reduces the risk of nonce clashing with the same wallet.
- **Transaction retries:** in case the nonce clash occurs in the node, it picks another random wallet, fetches an up-to-date transactions count for that wallet, and attempts to post a transaction once again. This process is repeated until the execution time limit is reached.

Execution code storage

In order to store the execution JavaScript code securely and efficiently, while minimising inconsistencies with library resolutions that may occur during the runtime, a CDN is used, the link to which is present inside the contract, along with minimal initialisation logic. In order to ensure that the code is not tampered with during import, a SHA-256 hash sum is checked, before the code is executed. This keeps the contracts size to a minimum, reduces inconsistencies and ensures the authenticity of the code, the source of which can be reviewed by anyone online.

Limits

The 120-second execution time limit is sufficient for awaiting the minimum amount of confirmations required to pass the transaction onto the destination chain. Additionally, the limit leaves room for transaction retries, in case they are required.

The 20-HTTP-requests limit is also sufficient for running all of the necessary RPC calls in order to pass & verify the existence of source-chain transaction, with room for transaction retries.

3.2.3 Settlement

Chainlink CCIP is chosen as a reliable and secure settlement layer due to its multi-layered security architecture. As soon as funds enter CCIP on the SRC chain, a corresponding event is emitted and the execution layer begins the acceleration process. On the destination chain, the funds are first taken from the liquidity pool as a short-term loan. Depending on the settlement layer execution time, (~24 minutes for Chainlink CCIP) it repays the loan, thus settling the transaction.

3.3 Gas Abstraction

One of the key components in providing the easiest user experience is gas abstraction. Not having enough gas on the destination chain for a cross-chain transaction remains to be the biggest reason for users abandoning cross-chain transactions. The infrastructure of Concero allows a user to perform the entire multi-step cross-chain transaction by signing a single message on the source chain.

For cross-chain bridging, in order to facilitate gas-abstracted transactions, a portion of the bridgeable source-chain token ($USDC) is taken in accordance to the following formula:

$$totalGasFees\_USDC = \frac{(0.75 \times 10^{18} \times src\_gasPrice + 0.75 \times 10^{18} \times dst\_gasPrice)}{USDC\_to\_Native\_rate}$$

Additionally, the following variables are taken into account when calculating the fees:

- Execution layer fees in $LINK (ChainlinkFunctions)
- Settlement layer fees in $LINK (ChainlinkCCIP)
- Concero fees – 10 bps of source token
- LP fees – 10 bps of destination token

$$totalFees\_USDC = totalGasFees\_USDC + eFees + sFees + cFees + lpFees$$

3.5 Risk Mitigation

3.5.1 Cross-Chain Execution Layer risks

Transaction Failure risk

The messenger wallet, which calls a function on the destination chain in order to place the en route transaction into the unconfirmed queue may fail due to several reasons, the risks of which can be minimised.

- **Insufficient balance for the messenger wallet:** this may occur in case the wallet's balance is not kept track of, which may lead to insufficient native balance required for signing a transaction on the destination chain. At the moment, an off-chain watcher is used, however Concero plans to introduce a fully decentralised system which will keep track of and top up messenger wallets in a transparent and secure manner.
- **RPC call failures:** since the execution code uses centralised, publicly available RPC providers, such as Alchemy, Infura, etc., not only does trust has to be minimised in order to ensure the reliability of data (see Consensus risk), the RPC node may not always be online. In order not to be exposed to the node unavailability risk, the retry mechanism is put in place, which takes a random RPC endpoint from the endpoints list, and retries the request with a different provider. In order to provide reliable service, the uptime of RPC endpoints must be regularly reviewed with endpoint replacements in case they are required. List of RPC Endpoints used can be found in our documentation at docs.concero.io

For cross-chain transactions, which involve migrating assets to a different chain, the logical flow of Concero's infrastructure eliminates the risk of lost funds in case of a failure in the execution layer, as it's only triggered when a success has been received from the settlement layer. This indicates that the funds are already on the way to the destination chain and will be available there regardless of the execution layer. This ensures that in the worst case scenario, there is no value loss for the end user.

Logging risk

One of the security risks that Chainlink Functions bring is the possible secret leakage through the error or console output, which is encoded and returned on-chain at the end of the execution. Therefore, in order to mitigate the risk of such leakage, all errors must be handled strictly and no arbitrary errors should be returned from the code, in case they don't fit the handled cases.

Consensus risk

The fact that the consensus is made on the responses of only 4 nodes, introduces the risk of unreliable consensus. In efforts to mitigate such risk, additional measures are taken. As such, the RPC endpoints present in the executed code are randomised within each node, which drastically lowers the risk of tampering with the returned data, as all 4 of the nodes will have to come to consensus on the responses provided by different RPC endpoints. Additionally to the uptime of the RPC endpoint itself (see Transaction Failure risk), the endpoints must be regularly reviewed and replaced in case needed in order to minimise the probability of malicious or ingenuine query results.

DON-Hosted secrets leakage leak

Chainlink Functions, as the execution layer, utilises a threshold-encrypted secret storage, hosted on the Decentralised Node Network of Chainlink. The secrets are only decrypted via a multi-party decryption process Chainlink Functions Beta. In the event of a leak, the contents of the secrets are encrypted, which keeps the allowlisted messenger wallet private keys uncompromised.
However, trust is still placed on the integrity of each individual Chainlink node, which gets access to the private keys during runtime, and may capture them in case the node is malicious. This is the primary reason for considering the unconfirmed transactions queue untrusted by default, requiring a compulsory source-chain check.

3.5.2 Proxy deployer key leakage risk

One of the key points of control for the entire infrastructure is the proxy deployer, the private key of which, if not managed properly, can lead to catastrophic consequences for the system.
A set of measurements and policies is put in place include the following:

- Proxy deployer key is stored externally, away from the workstations
- The access to the proxy deployer key is reduced to a minimum and strictly controlled
- A multi-signature set of keys is used in order to remove a single point of authority

3.5.3 External contracts risk

It’s crucial to eliminate the potential attacks that can be taken by arbitrary token contracts the single-chain execution contract interacts with. These attacks include, but not limited to Reentrancy or Denial of Service attacks. In order to mitigate them, the following measures are put in place:

- Reentrancy guards
- Safe, Checked calculations
- Access control
- Gas limits for external calls
- FeeOnTransfer token handling

3.5.4 Chain reorganisation risk

Chain reorganisation occurs when a blockchain reorganises its blocks due to the arrival of a longer chain of blocks. This disrupts the order of transactions and leads to double-spending risks, which directly affects the infrastructure of Concero. Even though chain reorgs occur rarely and don’t pose a significant risk due to the efficiency of liquidity usage for Concero, the main measure put in place in order to minimise the likelihood of double-spending in case of a chain reorg is the minimum amount of confirmations that has to be awaited by the execution layer, depending on the per-transaction risk factor, as well as the universal minimum amount of confirmations required for any transaction passing through the infrastructure.

3.5.5 Settlement layer failure risk

Considering the security of Chainlink CCIP, the risk of transaction failures after the funds have been successfully sent into CCIP on the source chain are minimal. Regardless of this, it’s important to ensure the security of the funds that the end user is sending through the system. In case the execution layer has already issued the optimistic loan on the destination chain, but settlement has failed on the way, the net amount of funds in the infrastructure would not suffer from a loss, and the user would be able to retry the transaction on the settlement layer. If such failure occurs, there will be no destination chain operations taking place in order to ensure the funds are to arrive safely to the user’s wallet, therefore a bridgeable $USDC token will be sent directly to the user’s wallet.

3.5.6 Low liquidity risk

Sufficient liquidity on the destination chain is essential for taking optimistic loans while the transaction settlement is in progress. It plays an essential role in Concero's fast-tracking in the execution layer. In order to minimise the likelihood of insufficient liquidity on the destination chain, the following measures are put in place:

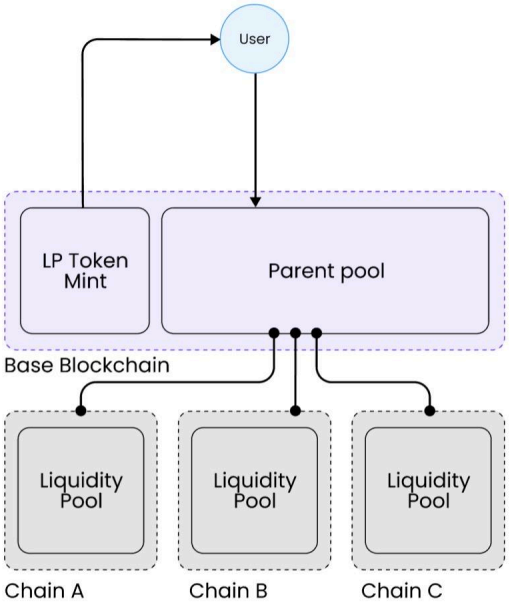
- Transaction simulation: A complex estimation of the entire flow of the transaction, which reduces the risk of failures. May not be available in contexts of solidity-level integrations, as it’s not enforced, but highly recommended
- Liquidity checks on the destination chain: A simple, highly available check which can be performed by integrators off-chain, before running a transaction.

The maximum liquidity cap will get adjusted in accordance to the utilisation volume of the liquidity leaving a liquidity buffer in order to minimise the transactions which risk to be on the fence of reverts due

to low liquidity. It's important to note, that in the worst case scenario there will be no value loss for the user, in case a transaction with insufficient destination chain liquidity is executed, bypassing the optional simulations and liquidity checks. As the execution layer is only triggered while the transaction is on the way through the settlement layer, the user will receive his funds in the form of a bridgeable asset on the destination chain when the settlement layer completes operation.

4 Liquidity Infrastructure

4.1 Architecture



The architecture of liquidity pools involves the Parent Pool, hosted on a single chain, and Child Pools, deployed to the rest of the supported chains. The Parent pool has the exclusive rights to control liquidity on each of the child pools.

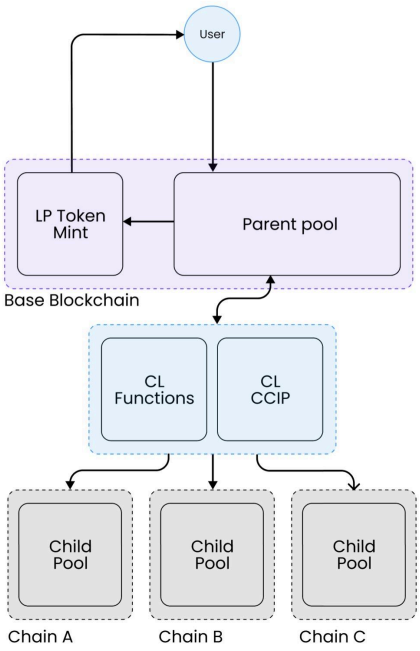
The advantage that this architecture brings is the ease of use for the liquidity providers, as the fee is earned universally throughout all the pools, regardless of the volume discrepancies between each pool. The liquidity provider is entitled to a share of earned fees, which is determined in proportion to the amount of LP tokens held by the liquidity provider and the sum of earned fees across every pool in the infrastructure. This approach removes the need of the liquidity provider to transfer the liquidity from low-volume pools, to the ones with greater volume. Additionally, the initial distribution of equal proportions of provided liquidity is taken care of by the Parent pool. In order to interact with the infrastructure as a liquidity provider, all actions are taken exclusively through the Parent Pool.

4.2 Liquidity Usage

The user-facing infrastructure of Concero takes optimistic loans from the pool on the destination chain in order to fast-track the transactions that are en route to settlement. In order for the pool to provide liquidity in an efficient and secure manner, several measures were implemented.
Every pool exposes a single interface for obtaining the loan which is only callable from within the infrastructure itself. Any other types of external calls are not allowed inside the pool, making it as isolated as possible in order to reduce the exposure to potential vulnerabilities.
The runtime efficiency of the user-facing liquidity infrastructure is achieved by minimising storage operations while obtaining and repaying the loans, with some operations moved to the execution level.

4.3 Deposits

The cross-chain liquidity deposit logic consists of 3 stages with all communication between the LP and the infrastructure taking place in the parent pool. The liquidity infrastructure uses the well-known concept of liquidity tokens, which represent a portion of the liquidity with fees the provider is entitled to.



1. LP deposits the funds into the parent pool
2. Parent pool uses CLF to fetch liquidity data on all chains in order to determine the LP amount to issue to the depositor
3. Parent pool receives liquidity data and issues LP tokens in proportion to the deposit, to the liquidity provider
4. Parent pool initialises the liquidity distribution using Chainlink CCIP, sending an equal proportion of the deposit to each of the pools, completing the liquidity provision

4.3.1 Liquidity calculation

After approving a certain amount of tokens, the liquidity provider, who is willing to add liquidity to the infrastructure, calls a deposit function in the Parent Pool. After checking the approval, the deposit function logic invokes Chainlink Functions to individually read each of the child pool's liquidity and return the sum of the liquidity as an integer back to the parent pool.

For each pool, the total pool liquidity is calculated according to the following formula:
$totalLiquidity = balanceOf(Pool) + loansInUse + withdrawalRequestsDue$

4.3.2 Liquidity token issuance

After the liquidity is obtained, the following formula is used to determine the amount of LP tokens to be minted in accordance to the user's deposit against the total liquidity present in the pools:
$lpAmount = (((Total\_USDC + USER\_USDC\_DEP) * totalLpToken) / (Total\_USDC)) - totalLpToken$

After obtaining the tokens provided and determining the amount of LP tokens to issue, the Parent pool calls the LP token contract to mint the required amount for the liquidity provider.

4.3.3 Cross-Chain Liquidity Distribution

The liquidity obtained from the depositor is then distributed in an equal proportion across all pools (including the parent pool) using Chainlink CCIP, which sends a transaction directly to each of the pools.

4.4 Withdrawals

As with any LP-related actions, the withdrawals are processed strictly through the parent pool, which in a similar fashion to deposits, interacts with the child pools in order to obtain total liquidity, burn the LP tokens and perform the withdrawal. As at the point of requesting the withdrawal, the liquidity is distributed across multiple chains. Therefore, the withdrawal process is split into three parts: the initiation of the withdrawal, the collection of cross-chain liquidity in the Parent pool, and the completion of the withdrawal.

4.4.1 Initiating withdrawal

In order to initiate the withdrawal, the liquidity provider must approve the spending (burning) of an arbitrary portion of LP Tokens that are issued during the deposit.
The parent pool uses the following formula to calculate the liquidity ($USDC in V1) the LP is eligible to withdraw based on the tokens provided against the total liquidity across all of the pools.

In a similar fashion to deposits, the pool invokes Chainlink Functions in order to obtain total liquidity for all of the pools according to the following formula:
$totalLiquidity = balanceOf(Pool) + loansInUse$

Then, the amount of USDC the depositor is eligible to withdraw is determined by the following formula:

$$usdcAmount = (totalLiquidity * USER\_LP\_RECEIVED) / LP\_Total\_Supply$$

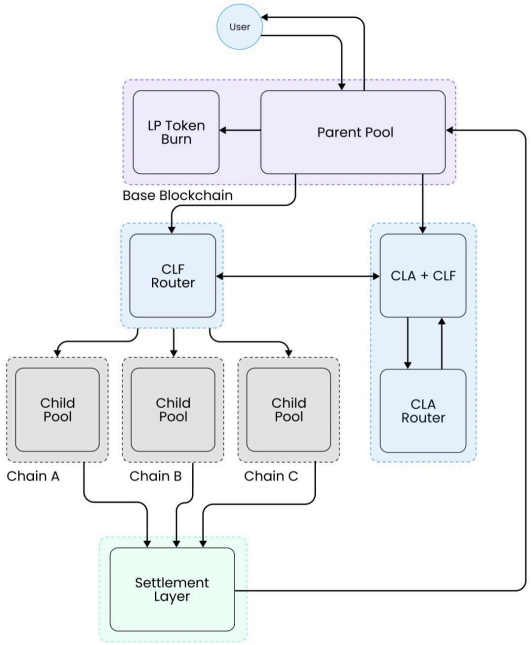
4.4.2 Collecting cross-chain liquidity

In order to obtain liquidity that is present on the child pools, the parent pool invokes the CLA+CLF contract, which is responsible for the delayed triggering of the liquidity withdrawal from child pools back to the parent pool. After a withdrawal request is placed into the contract, a deadline of +7 days is counted, after which a liquidity pull will be performed. For automating the task, Chainlink Automations is used, which check the condition of the contract with every block, and trigger Chainlink Functions for performing the liquidity pull using one of the authorised messenger wallets, the private key of which is located in DON-Hosted secrets, similarly to the execution layer of the user-facing infrastructure.

The parent pool is keeping track of the returned liquidity in a struct, originally created during the withdrawal initiation, and only allows the completion of the withdrawal when all of the pools have provided the requested liquidity. The struct updates take place when ccipReceive is triggered on the parent pool, which updates the corresponding withdrawal request, using the data present in the call.

4.4.3 Completing withdrawal

After the liquidity has been collected in the parent pool, the liquidity provider can call the *completeWithdrawal()* function in the parent pool, which will burn the LP tokens that were originally taken and send the liquidity back to the depositor, thus completing the withdrawal process.



Step-by-step breakdown:

1. LP initiates the withdrawal request and approves LP tokens to spend
2. Parent pool obtains cross-chain liquidity of child pools using CLF
3. Parent pool submits a pending withdrawal request to CLA+CLF contract

4. 7 days later, CLA+CLF sends a withdrawal request to child pools, moving the liquidity back to the parent pool
5. After the liquidity is moved to the parent pool, the LP calls the withdrawal completion, which burns the LP tokens and returns the liquidity + collected fees

4.4.4 Withdrawal cooldown period

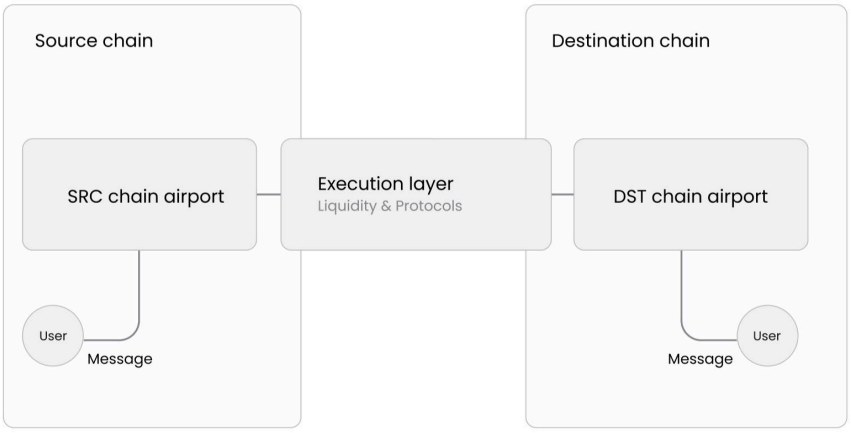
In order to discourage frequent liquidity withdrawals and re-provisions, and in order to provide Concero with a more stable and predictable liquidity balance across all of the supported chains, a 7-day cooldown period is introduced before the withdrawal is processed. This cooldown period freezes the amount to be withdrawn at the time of the initial withdrawal call, thus freezing any potential fee earnings that would have taken place during the cooldown period.

In order to mitigate potential liquidity use during the withdrawal which is to take place 7 days after the request, there is a 30-minute buffer, within which the liquidity of the destination chain pool is checked including the withdrawal request amount, therefore allowing any last loans, in case they were taken, to be repaid no later than 30 minutes of the withdrawal request, and a 2-hour period, within which the liquidity is being sent to the parent pool.

5 Messaging

Concero messaging is a cross-chain messaging protocol intended for transfers which don't incur any value, thus there is no settlement taking place, as opposed to the cross-chain interoperability infrastructure of Concero.
There are two key functions which will be available in any airport contract:

- sendMessage (sends a message to the destination chain with arbitrary calldata)
- checkStatus (returns a status for a message by a given message_id: Unverified, Verified, Executed)



5.1 Airport contracts

Airport contracts are responsible for taking and releasing messages from/to wallets or integrators. After the function sendMessage() is called on the source chain by any of these entities, the source-chain airport contract begins the cross-chain message transfer utilising the execution layer, for which Chainlink Functions is used.

The Airport contract takes the following parameters:

- destination address (which may be a contract in case a call needs to be performed)
- callData (which in case of the contract needs to be passed in a function call)
- Function signature (to indicate which function is to be called on the destination chain)

Airport contract accepts messages from any entity, which could be integrators or users. The fee is taken for the entire cross-chain message transfer within the initial sendMessage call as per the formula:
$totalFee = srcClfFee + dstClfFee$

5.2 Execution layer

The service of choice used as the execution layer is Chainlink Functions. Its responsibility is to send messages to the destination chains after being triggered from the source chain. Analogous to the execution layer in Concero cross-chain interoperability platform, the execution layer uses a messenger wallet with its private key which is threshold-encrypted in DON-Hosted secrets storage of Chainlink. The architecture implies that one of the 4 nodes which are executing the code during the call will succeed in posting a message to the destination chain and a successful fulfilment will be returned to the source to indicate a status of the message sending operation.

5.3 Risks

Given the messaging infrastructure is not designed to pass value-bearing transactions, the risk profile is lower. Yet there are still several risks which are similar to the execution layer of the cross-chain interoperability infrastructure.

5.3.1 Arbitrary contract interaction

The permissionless design behind the messaging protocol implies that it has to interact with any wallets or contracts from both the source chain and the destination chain. This means that the contracts are at high risk of common attacks such as the reentrancy attack, the DOS attack or others.

Even though there is no attack that can lead to any type of value loss, in order to ensure stable operation of the messaging protocol, the following measures are implemented:

- Reentrancy Guards
- Gas limits for function calls
- Allowlisted management/messenger functions
- Upfront fee payment

5.3.2 Execution layer

The execution layer poses the same risks as with the cross-chain interoperability infrastructure, which have been covered above. These include: Transaction Failure risk, logging risk, Consensus risk, DON-Hosted secrets leakage risk.

6 Fees & Economic Viability

6.1 Fee Structure

There are four main types of fees that are incurred when using Concero Protocol:

1. Executor fee - fee charged by the executor network (in V1 Chainlink Functions)
2. Settlement fee - fee charged by the settlement layer (in V1 Chainlink CCIP)
3. Gas fee - network gas fees
4. Protocol fee - fee charged by Concero Protocol (and split with LPs)

For a further explanation of the fee structure with examples, please refer to the relevant section in our official documentation at docs.concero.io

6.2 Capital Efficiency

Due to the nature of our infrastructure, Concero requires a relatively small amount of liquidity to facilitate a large transaction volume. All of the transactions in V1 are settled through Chainlink CCIP which on average takes around 24 minutes to settle. This means that the level of liquidity required is equated to roughly 30 minutes worth of volume that is flowing through Concero cross-chain infrastructure. This approach ensures that Concero can scale very effectively with an increase in demand.

There are, however, downsides to this approach especially when it comes to larger transactions as there might not be enough liquidity to facilitate a transaction. In this case, and in any other case of failure of the execution layer, Concero falls back onto the settlement layer (in V1 Chainlink CCIP) and the user has to incur longer transaction times. At no point are user funds in any danger. Within our SDK, we will include transaction simulation so users are made aware that their transaction will take longer in case the fall back on settlement layer is required before the transaction is initiated. For those developers who are interacting with the protocol directly through an end-point, we recommend integrating transaction simulation functionality for better user experience.

6.3 Liquidity Provision

Users can become liquidity providers of the Concero to earn transaction fees of 10bps (0.1%) on every cross chain value transfer that uses liquidity. The fee will be collected in the asset the pool holds (in V1 we will only have $USDC pools so all of the LP fees are collected in $USDC). In order to ensure that

liquidity providers are exposed to all transactions across all chains, we have used the Parent-Child pool design where liquidity is provided into a Parent Pool and distributed equally across Child pools.

Our main goal is to ensure that we have a very steady amount of liquidity as large liquidity fluctuations will lead to a worse user experience for the end user. There are two mechanics that we have implemented to address this:

1. **Parent-Child Design**: by exposing every LP to all of the fees that are being collected, LPs do not have to stake and unstake regularly depending on chain activity to maximise returns. In Concero V2 we will introduce active liquidity rebalancing where liquidity will be rebalanced depending on chain activity to further maximise the exposure to fees and to further optimise liquidity requirements.
2. **7-Day Unstake Period:** When a liquidity provider decides to withdraw liquidity they will have to wait 7 days after the withdrawal request until their assets are claimable. During that 7 day period the liquidity provider is not exposed to additional fees being collected. This process ensures that liquidity providers are incentivised to keep their liquidity in. Additionally, it gives other users who want to provide liquidity a 7 day window to do so. Ideally, if there is enough demand and volume the level of liquidity will not diverge from the liquidity cap.

6.4 Liquidity Cap

Capping liquidity increases returns for Concero liquidity providers (by maximising the % of in-use liquidity) and improves the overall security of the protocol (by reducing the downside risk). At the launch of the protocol, liquidity capping is done manually by the team behind Concero but this process will be automated and decentralised with future updates. The process is manual at the start to ensure that as the protocol is deployed and in the early days we can experiment and collect data on LP and user behaviour so we can determine the best system to manage the cap. You can expect a fully decentralised and automated system to be integrated within the first 6 months after mainnet launch.

7 What's next?

The team behind Concero is committed to improving and growing Concero Ecosystem and Concero Protocol. We are actively developing the next iteration of the protocol and expanding the ecosystem through various integrations. Follow us on social media to stay updated on our progress. Additionally, join our regular community calls to ask questions and engage with the team.

7.1 Concero v2

Concero V2 is going to be a major update where we are focusing on expansion of the execution layer and settlement layers in order to grow the coverage of chains and protocols supported. In V2, users will be able to perform transactions from any chain to any other chain with a consistent user experience and high speed. We are also working on transaction batching mechanisms that will reduce end-user transaction costs. As mentioned previously, we are building an active liquidity re-balancing system that will be able

to pre-emptively migrate liquidity to the chains where higher usage is anticipated in order to optimise for capital efficiency further.

7.2 Concero Token

We wanted to address the question of a Concero Token within our Whitepaper in order to remain transparent about the future of the protocol. We strongly believe that the primary attribute of a token should be its utility. Therefore, we have decided not to launch a token until there is a clear and absolute need for it. Although it is likely that such a need will eventually arise, the timing of the Concero Token launch will be determined solely by its utility and necessity, without influence from market conditions, sentiment, time, or other variables.

8 Acknowledgements

The journey to Concero V1 has been challenging, and it wouldn't have been possible without our dedicated core team:

1. Nikita Gruzdev - member of the core team (help with building the core infrastructure)
2. Kitu Kuznetsov - member of the core team (product development and user insights)
3. Nithin Shylendra - member of the core team (marketing and communications)
4. Patric ‘Barba’ Carniero - member of the core team (help with building the pool infrastructure)

Chainlink Team has played a crucial role in Concero development by helping with everything from protocol architecture to partnerships and marketing support. Most notably the following individuals:

1. Ade Fola-Alade - Chainlink Systems Architect (help with Concero architecture)
2. Sam Friedman - Chainlink BUILD CTO (help with Concero architecture)
3. Sergi Rey- Chainlink BUILD (help with integrations and partnerships)
4. Oliver Birch - head of Chainlink BUILD (help with partnerships and fundraising)
5. Nicole Jackson - member relations (help with introductions, partnerships and integrations)