Benchmarking Bridge Aggregators

Shankar Subramanian\(^1\), André Augusto \(^1\), and Rafael Belchior \(^1\)

\(^1\)Affiliation not available

January 10, 2024

**Abstract**

Blockchain aggregators play an instrumental role in the evolution of blockchain technology, serving as pivotal enablers of interoperability, efficiency, and user accessibility in an increasingly decentralized digital world. However, the literature on this emerging technology is scarce and not systematized, making it harder for practitioners and researchers to understand the field. In this paper, we systematize blockchain aggregators, with a specific emphasis on bridge aggregators. We present an exhaustive analysis of a diverse array of token and message aggregators, each distinguished by its unique architecture. Our investigation delves into critical aspects of these aggregators, encompassing their functionality, security measures, pricing models, and latency characteristics. The objective of this research is to furnish readers-encompassing both users and developers-with insightful and actionable information, thereby facilitating informed navigation through the complex landscape of blockchain aggregators.
Benchmarking Bridge Aggregators

Shankar Subramanian, André Augusto, Rafael Belchior

January 2024

Abstract

Blockchain aggregators play an instrumental role in the evolution of blockchain technology, serving as pivotal enablers of interoperability, efficiency, and user accessibility in an increasingly decentralized digital world. However, the literature on this emerging technology is scarce and not systematized, making it harder for practitioners and researchers to understand the field.

In this paper, we systematize blockchain aggregators, with a specific emphasis on bridge aggregators. We present an exhaustive analysis of a diverse array of token and message aggregators, each distinguished by its unique architecture. Our investigation delves into critical aspects of these aggregators, encompassing their functionality, security measures, pricing models, and latency characteristics. The objective of this research is to furnish readers—encompassing both users and developers—with insightful and actionable information, thereby facilitating informed navigation through the complex landscape of blockchain aggregators.

1 Introduction

Efforts to address the interoperability challenge have led to the development of various projects aimed at facilitating communication across different protocols and networks. These initiatives have introduced connectors such as cross-chain bridges, Decentralized Exchanges (DEXes), off-chain API-based protocols, and on-chain oracle services. Although these solutions have successfully addressed the issue of isolation, they fall short in providing a more integrated experience to users, lacking a significant layer of abstraction that could unify these disparate protocols and blockchains.

The early evolution of the internet offers a parallel example, where e-commerce was initially fragmented, requiring users to navigate through a multitude of isolated vendors. Not only was the process time-consuming but also impractical for many, driving them towards alternative, albeit more expensive, physical stores. Early internet aggregators, like Amazon, Hotel Booking, and Uber, emerged as pivotal entities by consolidating these scattered resources, thereby enhancing user experience and becoming indispensable in everyday life. Aggregation extends beyond the application level to the very protocols that underpin the internet. Protocols such as the Dynamic Host Configuration Protocol (DHCP) [1] and the Border Gateway Protocol (BGP) [2] have simplified internet accessibility. Users are relieved from the complexities of acquiring IP addresses,
configuring devices, or computing routing paths for data transmission. Users can effortlessly access websites without needing to reference directories for IP addresses, thanks to these aggregative mechanisms.

Recognizing the significance of such aggregators in streamlining and enhancing user and developer experiences, this paper proposes the introduction of similar aggregative structures within the blockchain ecosystem. Aggregators can substantially augment the functionality and accessibility of blockchain technologies for both seasoned and novice users. A vital attribute of an aggregator is its capacity to conduct routing analyses on behalf of the user, thereby optimizing multi-protocol or chain transactions based on various parameters such as gas costs, trade values, security, and transaction speed. This paper specifically focuses on bridge aggregators, exploring their potential to revolutionize the blockchain landscape [3].

1.1 Problem

The contemporary landscape of blockchain technology is marked by significant fragmentation and a lack of comprehensive abstraction layers. Users face the onerous task of manually managing keys, scrutinizing protocols for trading, and running local nodes to interact with blockchains. These requirements, while essential, add complexity and hinder user engagement with blockchain technology. A particularly challenging aspect is the issue of interoperability, where the integration of various protocols and blockchains is not streamlined. Users find themselves in a position where they must individually establish connections and networks, adding to the already extensive list of prerequisites for effective blockchain interaction, not to mention the necessity to abide by local laws and regulations [4].

The concept of aggregators presents a viable solution to the interoperability challenge. Aggregators aim to unify the fragmented protocols, thereby improving the user experience and facilitating smoother blockchain interactions. Aggregators also offer significant benefits in terms of maintaining operational continuity and resisting censorship. By amalgamating various sources, aggregators ensure that, even in the event of individual source failures, alternative routes through the network of protocols and chains remain accessible. This redundancy is key to sustaining the functionality of blockchain systems, as aggregators effectively manage communications across diverse and isolated protocols. Thus, aggregators streamline the user experience in the blockchain ecosystem and enhance the resilience and reliability of a system.

However, the practical analysis of cross-chain aggregators is a cumbersome task for users, given the multitude of protocols that require thorough and strategic examination. This complexity diverts user resources from other critical activities such as conducting security reviews, developing new features, or expediting product launches.

1.2 Contributions

The primary objective of this paper is to conduct an empirical analysis of the main bridge aggregators. As bridge aggregators are in their inception, there is a lack of existing literature on this subject. Our study dissects the various elements that form an aggregator and evaluates them. We examine the diverse architectures and design choices as-
associated with these aggregators, highlighting the different trade-offs they present. This research provides valuable insights to prospective users intending to utilize these protocols and future researchers and developers designing and implementing cross-chain solutions. We test different parameters such as system architecture, supported features, openness, decentralization, latency for completion, total cost, fallback model, and ease of use.

We also suggest use cases for each type of aggregator. In addition to theoretical analysis, we also rigorously benchmark token-aggregators as they provide the most data. As DeFi is currently the most competitive field, with token values fluctuating in seconds, and users trying to maximize their trade values, aggregators need to move very fast, or else they get cut by a competitor or are exploited by MEV [5]. Message-aggregators are also looked into in-depth in the theoretical department as they have different requirements where timing is not the most crucial, but instead, the data guarantees surrounding the cross-chain messages.

2 Overview of Aggregators

2.1 Aggregators

We define an aggregator as a service that allows users to interact with more than one service. There are three different kinds of aggregators:

1. Aggregators that allow users to interact with different blockchains by reading data and sending transactions.
2. Aggregators that allow smart contracts to interact with smart contracts on other blockchains
3. Aggregators that allow for multi protocol DeFi.

Figure 1: Aggregator Black Box
Protocols are stand alone projects that allow users to perform a particular transaction. Transactions can be DeFi swaps or changing the data stored in another smart contract. Aggregators abstract the complexities of writing custom handlers for each protocol by providing users with a black box model.

The user simply interacts with an aggregator API with certain transaction parameters and the aggregator performs the required computations on the input, which it parses to match the schema for each of the protocols the aggregator supports (listed as 0,1,2 in the figure). The aggregator then optimizes the generates routes for a certain parameter like speed, token value, or security and then selects a particular service (denoted by 1 in the figure and the ones that are not taken are 0) if possible and returns a transaction that is directly executable by the user on a blockchain by submitting a signed transaction.

Aggregators may operate on any kind of protocol as long as they provide a similar service. Figure 1 represents an orange rectangle as a blockchain with the squares as different protocols. Thus an aggregator can interact with different blockchains, different protocols, on and off chain APIs, etc.

Blockchain Aggregators can be broadly categorized into two classes – those that directly change on-chain state and others that do not. The aggregators that do are called Bridge Aggregators. This paper focuses on these aggregators, and we dive into their types and functionalities in the following section.

A bridge is an entity that connects multiple resources (for a formal definition, see [6]). These resources can be DeFi protocols, token pools, different blockchains, and oracles (blockchain and outside world). As such, bridge aggregators allow users to send transactions through multiple token pools that may or may not be associated with different protocols. These protocols may even exist on different blockchain than the one the transaction was issued on. It may also send a message from one blockchain to a smart contract in another blockchain.

The essence of a bridge aggregator can be distilled to a service that impacts user transactions by performing computation on their behalf through their smart contracts and off-chain APIs based on user input, and either executing the result themselves or allowing the user to select the result to be executed. Bridges can also be categorized by their trust model and generalizability as follows [7]:

- **Natively Verified**: Validators of the destination chain are responsible for verifying source transactions. This requires the least trust between the user and the nodes executing transactions but is the least generalizable and is not scalable across different kinds of VMs (EVM, Solana VM, etc). It is usually implemented by having a light client run on both networks or creating multi-purpose clients.
  
  Example protocols: IBC, LayerZero, Rollups

- **Local**: Parties interacting verify each other on demand. This runs on low trust as it follows the atomic swap model. However, stalemates can occur if either party is down and the recipient can be influenced to alter transactions, as parties usually settle transactions off-chain. On-chain settlers are not influenceable unless they have additional functionality that allows them to.
  
  Example protocols: Connext Legacy, Hop
• **Optimistic:** External validators that are part of the protocol bond slashable funds on the source chain while another set monitors the network trying to find fraudulent transactions. Only requires 1 of n honest validators.

  Example protocols: Nomad, Hyperlane (ISM)

• **External:** Validators that are not part of the source or destination chains validate transactions and execute them across networks. This requires the most trust out of the 3 models and is the most practical. There are methods such as staking, MPC, and slashing, to reduce trust.

  Example protocols: Wormhole, Axelar, Celer, Ronin, Hyperlane (ISM)

### 2.2 System Model and Actors

We categorize bridge aggregators based on their architectures: i) the user interaction model and ii) the open-source nature of the protocol. The two categories, while distinct, appear to be tightly coupled and protocols usually lie on a spectrum between open-sourced and user-run to proprietary and API-based.

Bridge aggregators can be generalized based on their primary functionality to be either Token or Message Aggregators. Token aggregators are services that take in as input the trade information such as from and to tokens, swap values, source, and destination chains and then compute different swap routes through different DeFi bridges such as Uniswap, Squid router, or 0x to name a few. These bridges can link different protocols on the same network, or different protocols on different networks with the aggregator being involved in the cross-chain swaps, or using another multi-chain connector that provides cross-chain liquidity. The functionality of these protocols is based around DeFi and because token values fluctuate a lot in a small time frame, aggregators try to be the fastest and provide the best routes and pricing to the users. As such, their routing algorithms are usually secretive as it is their only source of profit. This results in "closed off" architectures and they are not extensible by users as users cannot change the source code used by the routes but can only influence the routing by changing inputs.

The message aggregators allow for arbitrary cross-chain transactions to be sent across chains and are usually more functional. They are usually open-sourced and have a more decentralized governance model. All protocols share a common subset of components:

1. **Transaction** $Tx(t, s, n)$
   - $t$ – Different kinds of transactions (cross-chain message, same chain DeFi, cross-chain DeFi)
   - $s$ – Sender of the transaction (user, relayer, protocol, aggregator)
   - $n$ – Number of times the transaction has changed state (nonce)

2. **Blockchain** $B(t, c)$
   - $t$ – Type of chain (Source, Target, Protocol)
   - $c$ – Type of contract
i. Router contract – The contract that receives, validates, and routes the transaction to the destination smart contract

ii. Governance contract – The contract that manages the protocol (user funds, fees, validator list)

3. Relayer

(a) Routing API – API that computes the path for the $T_x(t, s, n)$ based on the parameters provided by the user

(b) Transaction Parser – Extracts the data from the $T_x(t, s, n)$ to construct the destination transaction $T_x(t, s, n + 1)$

(c) Event Listener – Listens to the router contract for on-chain events to trigger the transaction parser. May also double as a relay

(d) Relay Executor – Builds and sends the destination transaction $T_x(t, s, n + 1)$

4. Validators – Monitors the network by comparing the $T_x(t, s, n)$ in the protocol mempool against the one in the router contract. They can be modified to not change state and only read the data from the network and hence can be open sourced and run by anyone.

Figure 2 contains components from the API Based Aggregators 2.4 and Open Sourced Model Aggregators 2.5. The current section provides an overview of an aggregator transaction and we provide an in-depth explanation of its components below.

User: This box contains represents the phases through which a user interacts with an aggregator. Users do so using an application or webpage that allows them to interact with the protocol. The UI parses user input into the aggregator-specific parameters and creates an API request to the routing api of the aggregator, which returns a list of quotes, routes, and transaction calldata that can be executed on a blockchain. The user builds the transaction by selecting the quote they wish to use and submitting it to the blockchain.

Routing API: This module receives requests from the user and breaks them down into the components that specify the action the user wants to perform. It then runs the inputs into an algorithm that can be proprietary to receive a list of routes and quotes, which is returned to the user.

Source Chain 1 & 2: The source blockchain contains the smart contracts that the user interacts with. This can be:

1. User interacting with Aggregator Contract

2. Smart Contracts interacting with Aggregator Contract

The Aggregator contract in turn can be a single contract or be governed by a DAO. The baseline is that the aggregator contract receives all the user-created transactions and is a way to notify the cross-chain protocol agents about a transaction.

Cross-chain Protocol Agents (Phase 1): Is the core of the aggregator, and consists of 2 phases. In Phase 1, it listens to the router contract for new transactions. We call this a Relayer in Phase 1. There can be several relayers run by the aggregator. In the
Figure 2: Generic architecture of a bridge aggregator
event of multiple relayers, they enter a consensus phase to decide on the validity of user-generated transactions.

Once the consensus phase is completed, the transaction is committed to a shared storage (Finalized Storage) among the nodes. From here, the Transaction Builders take over and create the destination transaction from the shared storage. Builders then create proofs for the transaction that showcase the validity of the destination transaction and submit the transaction and respective proof to the destination chain.

Phase 2 relayers that are linked to the Finalized Storage are informed about an incomplete source transaction that is yet to have a destination transaction.

**Destination/Target Chain:** The Destination Transaction reaches the router contract and gets routed to the target address. The target can be a user address, a smart contract, or a DeFi protocol. The destination transaction corresponding to the source transaction of the user executes and the events are logged on the network.

**Cross-chain Protocol Agents (Phase 2):** The Destination Transaction is executed on the Destination Network’s Router. Similar to Phase 1, Phase 2 relays listen for the events and monitor the destination network, and once they see an expected transaction, the global state is updated to reflect completion.

**User run Cross-chain Protocol:** In this type of aggregator, the routing software is open-sourced and runs on the user’s computer. Here the user acts as the validators and relayers as the user create transactions on both networks.

The routing algorithm relies more on the computation’s mathematical model and current network statistics because there is usually no coordinated source for processed recent data. Older data is used, and some aggregators use machine learning models to try to generate the best routes possible. This architecture does not have a router contract or consensus layers for a single client, however, they can be extended to work for n-client architectures.

### 2.3 Parts of an Aggregator Transaction

There are several parts that are involved in a transaction. We separate them into Components and Parameters. Components are the data generated by aggregators and users, such as routes and queries. Parameters are factors that influence generated data.

#### 2.3.1 Components:

Components are non-standard data generated by aggregators, users, and the benchmarking tool. In this section we explain what they are:

1. **User Quote Query:** Contains the user-generated transaction parameters such as source and destination tokens, swap amount, source and destination blockchains, transaction message, transaction deadline, and transaction mode.

2. **Aggregator Quote:** The quote object returned by an aggregator in response to the user’s quote query. It contains the Aggregator Route, Fee, and transaction calldata.
3. **Aggregator Route:** This part of the Aggregator Quote contains the hops of the transaction and information about the protocols being used by the aggregator to fulfill the user quote query.

4. **APIReport:** This is the standardized report that we generate for each aggregator. It contains information about the networks involved, the time of creation, the source, and destination tokens, the aggregator used, the protocol the aggregator uses, and the value of tokens traded.

### 2.3.2 Parameters:

Parameters are factors that influence the components. In this section we explain what they are:

- **Fees:** In addition to the fee involved in using a protocol such as the withholder, bridging, and protocol fees, aggregators charge an additional fee. Protocols may also use an intermediate token (usually a stablecoin because it provides a constant peg instead of having two variables in the trade) when bridging, which requires additional token swaps.

- **Latency:** There is the protocol latency and the aggregator API and routing latencies. The aggregator latencies are more noticeable in cross-chain transactions due to the additional security constraint of block finality time.

- **User Interface:** Some protocols have a user interface that they can use, and some do not. Some protocols such as Uniswap [8] charge an additional fee when using the UI and not through the SDK. Message aggregators usually do not have a User Interface, but have a Block Explorer.

- **Customizability:** There may be additional services provided by protocols that allow for more customizability of transactions.

### 2.4 Proprietary Backend Model - API Based Aggregators

The aggregator offers their services through APIs that users interact with through a UI (DEX, message passing) or an SDK. The main features that define an architecture of this type are:

1. Backend is not entirely open sourced
2. Users need to register with the providers
3. The functionality is limited and not extendable by the users
We explain the transaction flow and detail the specific architectural components.

**Phase I (Creating a transaction, color: Red):** A transaction can be created when a user either directly sends a transaction to the user contract or to the aggregator through the proprietary routing API to get the best route and the transaction calldata for that route. The user then executes that calldata at the user contract, which interacts with a router to notify the protocols involved that a transaction has been created.

1. **User Contract** – Contract that the user interacts with. Can be a custom contract or a DeFi protocol like Uniswap.
2. **Protocol UI** – Interface that allows users to interact with the project, such as a DEX UI where users input the tokens to swap.
3. **Proprietary Routing API** – Performs some computation involving the inputs (e.g., tokens, source chain, destination chain, message size, security involved, deadline time) and the available protocols to generate a list of possible routes that match the preference of the user.

**Phase II (The off-chain segment, color: Blue):** Once the transaction has been created and is alive, the protocol collects it through a Listener and sends it over to the relayers for consensus. Once the validity has been confirmed, the pending transactions may be bundled together to save gas and sent to the router contract on the target network, where they are processed and redirected. Not all components here are required.

1. **Listener** – Event listener from the Source Contracts that monitor different blockchains and pickup emitted events that contain information regarding the issued transaction.
2. Consensus – Several listeners share the events they pick up and try to arrive at a consensus to find the legitimate ones.

3. Relay – Relays transactions across Chains to notify the Target of a potential transaction. These may or may not be finalized transactions and can be simply stored in a memory pool waiting for confirmation before execution.

4. Tx Builder – These get their inputs after Consensus and trigger the execution of the transactions stored in the mempool.

**Phase III (Processing, color: Green):** The destination transaction gets processed to be executed at the destination chain and affects the contracts involved. The contracts can be a DEX that gives the target address some ERC-20 tokens or a message sent to a contract to trigger some function execution. The components here are no different from those from Phase I. Validators now monitor the network for transactions to make sure the network is functioning as intended.

1. Validator – Monitors the network for transactions in the mempool to make sure there is no misuse of the protocol, it is also a way for non-protocol members such as developers to check the status of a transaction.

### 2.5 Open Source Model

This architecture provides users with all components required to complete transaction on their own. As all the parts are open sourced they can be modified to create custom protocols or run them themselves. As a result, these aggregators tend to not have the most user-friendly interfaces.

The tradeoff is that users have the highest security and liveness guarantees (assuming the components used are audited). These aggregators may also complete transactions in fewer contract calls due to the lack of a router, on single user transactions. A router if present when there are several clients involved in a single transaction. Non user clients running these nodes can be networked together to function like an overlay blockchain. All the components are usually well-audited and battle-tested due to their open-source nature.

![Figure 4: Decentralizable Open Sourced Model Based Architecture](image-url)
Phase I (Create a transaction, color: Red, Purple): The user runs software on their computer that we call a client, which is usually a package of the relayer and validator along with an SDK and database. The User interacts with the SDK to create the transaction.

The SDK in turn generates the routing information and calldata for the desired routing by parsing the information and then building the transaction with the keys involved, such as the user’s blockchain keys. An update is also made to the local state for transaction creation, this is similar to the mempool. The user then sends over to the User Contract in the blockchain that in turn interacts with a router where it gets processed.

1. Client – Software that contains the modules that manage the transaction. It involves the Relayer, Validator, and Database.
2. Tx Handler – An SDK or UI through which the user inputs transaction parameters
3. Tx Parser – Parses the user input into Protocol Specific Format
4. Tx Builder – Constructs the finalized transaction by creating Routes and other steps involved in the transaction
5. Relayer – Client module that is involved with creating state changes such as sending a transaction to networks and listening to events emitted
6. Database – Local mempool and stores received transactions
7. User Contract – Same as in Proprietary model
8. Router – Same as Proprietary model except it may add some Fee or other parameters so that anyone running the client can execute the transaction

Phase II (The off-chain segment, color: Blue, Purple): All receivers connected with a router are notified of the transaction that has been emitted by the router. The relayers add the transaction to their mempool and the transaction is now in a global state. Some form of consensus or finalization of who executes the transaction is reached and a transaction is sent to the destination router.

Phase III (Processing, color: Green): The destination router executes the transaction and the validators involved check the status of the transaction.

1. Validator – Same as Proprietary model

3 Instantiations of Aggregators

In this section, we describe the main categories of bridge aggregators. We classified them based on their architectures, what they aggregate, and their functionality.

Based on the above we categorized them into – Centralized Token Bridge Aggregators, Decentralized Liquidity Aggregators, and On-chain Token Swap Aggregators.
The following sections are a succinct exposition into the Aggregators that fall into the above classes. The appendix has a more in-depth analysis where we look into the architectures, centralization, open sourcedness, and security.

3.1 Centralized Token Bridge and Pool Based Aggregators:

The following aggregators do not execute trades. They perform computations on the user input using other protocols and return to the user a series of quotes from which the user selects the transaction they wish to submit. An in-depth analysis of these can be seen in Appendix C

3.1.1 LiFi – Multichain DeFi Protocol Aggregator

Jumper aggregates cross-chain liquidity sources, such as cross-chain liquidity networks, bridges, DEXs, and lending protocols to offer swap and bridge operations [9]. Each bridge within Jumper gets a score, composed of qualitative factors such as trust assumptions and attack vectors, and quantitative factors prioritizing security, speed, and costs. Users can customize these priorities to align with their specific use cases. Notably, Jumper supports individual bridging and swapping operations as well as “xChain Contract Calls” allowing merged transactions for executing cross-chain contract calls. However, testing the protocol on testnets presented challenges, with limited support, heavy rate limiting, and instances of protocol failures, that we believe do not reflect across mainnet transactions. LiFi has an SDK [10] that only supports mainnet.

3.1.2 Socket – Multichain DeFi Protocol Aggregator

The Liquidity Layer facilitates swap and bridge operations by computing the best route based on user preferences for security, cost, and speed [11]. Socket also provides a service called “Refuel” that allows users to also get some native tokens on the destination network they are swapping to, so that they do not run out of gas tokens. Routing quotes are generated by proprietary software, redirecting swap requests to the most optimal DEX from their calculations [12]. This aggregator does not support testnets in any capacity, and we recommend users use the SDK when developing with this protocol.

3.1.3 XY Finance – Multichain Pool based aggregator

XY Finance is a DEX and bridge aggregator, focusing on optimizing routes and liquidity provisions that emphasize security, speed, and cost-effectiveness. It has a unique structure comprising XSwap (aggregator) and YPool (liquidity pool) [13]. XSwap is an atomic swap-based aggregator, and YPool allows for yield-based DeFi. It not only stores and manages liquidity but also performs automatic balancing to maintain liquidity thresholds within a certain range. All supported chains share the liquidity pool. XY Finance lacks an SDK, but API integration proved functional, albeit with implementation challenges due to the absence of comprehensive guides. XY Finance does not support testnets in any capacity. Their fee calculation is not clear from their documentation as it contradicts the results from the API and DEX application.
3.2 Decentralized Liquidity Aggregators:

These aggregators execute trades by allocating liquidity to user trade requests. They are stand-alone protocols that aggregate liquidity from various sources. Nodes running clients of these aggregators have surplus tokens they wish to trade or are willing to find liquidity that matches the user request for a fee. Users do not submit on-chain transactions when interacting with these aggregators apart from enabling token allowance. For an in-depth analysis refer Appendix D.

3.2.1 CoW Swap – Permissionless off-chain protocol that Aggregates Trades on single-chain

CoW Swap presents an interesting architecture that utilizes Batch Auctioning to establish the pricing of tokens across chains. The network support is limited and is not cross-chain. Transactions are submitted off-chain to “settlers” that try and complete the trade, settlers can complete partial or entire amounts of the trade [14]. Since the trading is off-chain, is also gasless and circumvents MEV. There is no SDK, but the documentation seems solid.

3.2.2 0x – Liquidity aggregator between Makers and Takers

0x tries to match liquidity between those that provide it (Makers) and those that consume it (Takers) [15]. Makers are those that provide liquidity such as off-chain DEXes and AMMs such as Uniswap, Curve, and Orderbooks. Takers are those that demand assets and thereby consume liquidity, such as centralized exchanges like Coinbase or Metamask. The trades start off-chain where a Maker and Taker arrive at a deal, and then submit the signed deal to the 0x smart contract that performs the trade between the on-chain wallets of the Maker and Taker.

3.3 On-chain Token and Pool Based Token Swap Aggregators:

Aggregators of this class are on-chain aggregators that can route transactions through different pools that offer token swaps. They are run on-chain and do not have any off-chain components. This results in them always completing a trade if the trade is possible and they have the most transparency as users can view the token amounts and price on-chain. They are also the quickest to complete as they only involve a single transaction and contain no external API calls or wait time. For an in-depth analysis refer Appendix E.

3.3.1 Uniswap – Decentralized AMM and off-chain trading protocol

Uniswap offers two services – AlphaRouter and UniversalRouter. They both are an open-sourced non API based DeFi protocols that only interacts with on-chain components. While a standard Uniswap transaction only swaps between the source and destination pools, the AlphaRouter allows for multiple hops between Uniswap pools to try to find the best route. The UniversalRouter goes a step beyond and aggregates trades across protocols, in addition to ETH and ERC20 tokens it also supports NFTs [16].
Uniswap is well-documented and has an SDK that works for mainnets and testnets. They also have a product called “Uniswap X” [17] that is similar to CoW but does not have enough documentation for us to test. While using Uniswap on-chain does not incur a fee, using the Uniswap dApp does charge a 0.15% fee [18]. Users can avoid the “UI fee” by interacting with the protocol through an SDK.

3.3.2 1inch – Decentralized AMM and P2P trading

1inch provides several services such as a DApp that functions as a standard DEX, an aggregation protocol that operates on several other protocols, a liquidity protocol that protects users from front-running, an on-chain limit order contract that uses off-chain signed transactions similar to CoW Swap, and P2P transactions where users can directly trade ERC-20 tokens [19]. The protocol is decentralized and run by a DAO and the core contributors. Users can take part in the trading by being resolvers or price aggregators. They have different services that offer user protection from front-run attacks and sandwich attacks.

3.4 Message Aggregators

These aggregators are about sending messages from one blockchain to another. They are of different flavors – some are block header verifiers, some are decentralized, and some are centralized for the benefit of speed. These can also execute generic blockchain transactions and are not restricted in functionality. For an in-depth analysis refer Appendix F

3.4.1 Hyperlane – Modular Interoperability Protocol

It is an open source aggregator model that functions as a cross-chain interoperability protocol designed to deliver messages and tokens across chains. It is developer-focused and comes with prebuilt modules and is customizable [20]. The customizability is on-chain and allows developers to create their own consensus modules, multi-sig requirements, and cross-chain transaction handling. Protocol fee is collected in native tokens and is collected after the destination transaction is sent. Cross-chain transactions are quick due to the centralized backend. Hyperlane v3 simplifies scripting of protocols but the documentation was not completed. Hyperlane v2 was complete and was used in this project. Hyperlane also supports a vast number of testnets and mainnets.

3.4.2 CCIP – Chainlink’s cross-chain interoperability protocol

This is Chainlink’s open source aggregator model that functions as a cross-chain interoperability protocol designed for message and token transfers across chains [21]. It is not customizable. Leveraging their expertise in Oracles, the backend is decentralized which contributes to slower transactions as the block finality of different chains is also taken into consideration. Users can pay the fee is Native or Link. The fee is paid when the source transaction is created. The developer documentation is well established, and they support a vast number of networks for users to test the protocol on, on both mainnets and testnets.
3.4.3 Hashi – Redundant Array of Independent Oracles

Inspired by RAID, this protocol collects block headers from different sources and validates them on-chain at each destination network [22]. Using this protocol is expensive and slow due to the amount of on-chain computation and works as a light client of sorts where developers are required to add additional constraints to their smart contracts to validate transactions within blocks. This protocol is under development and the documentation is not complete [23].

3.5 Aggregator Summary Framework

The framework that we came up with talks about the Aggregator components 2.3.1 and parameters 2.3.2 for each of the aggregators that we benchmarked 2.3.1.

- **Type**: The kind of aggregator that the protocol is, either token or messaging
- **Design Model**: Talks about how the user interaction model, and the aggregator backend architecture
- **No Setup**: Configuration required to use the aggregator, such as deploying aggregator-specific contracts, sending transactions to join avalidator list, requiring approvals from an Aggregator DAO
- **Block Explorer**: A block explorer to help users view transactions
- **Functionality**: The core component of the architecture that settles the transactions
- **Optimize Transaction On**: The parameters that the user can choose to optimize on
- **Customizability**: The changes to the base configuration such as implementing user-specific rules
- **Open Source**: How open sourced the aggregator is. We consider smart contracts, backend code, and documentation.
- **Target Audience**: Who should be using this.

4 Benchmarks

We present the experiments and analysis on the different bridge aggregators. For this, we designed a benchmarking tool that generates routes for all the aggregators, deploys contracts, executes routes, and executes cross-chain transactions.
The experiments were run on a computer with 8 cores@1.9 GHz base clock CPU and 32 GB RAM. The total storage used was 29 MB for 2250 aggregator specific quotes and 2250 APIReport quotes. The coingecko pricing for 360 quotes was 1.6 MB.

Computation and storage requirements are minimal with the generated reports being 2.1Kb, coingecko quote were 4Kb, aggregator quotes being 699 bytes to 18k. Cross-chain transactions had the largest file sizes and uniswap route objects were 24Kb. This is due to the way they represent data by denoting BigIntegers not as a string but as several fractions comprised of integers.

The networks we used were chosen based on the number of aggregators that supported them. For benchmarking token aggregators we used Ethereum Mainnet and Polygon Mainnet as we were simply querying the aggregator APIs and did not send transactions. However, the repository [24] was configured to also send transactions to the networks and execute them. In the case of token aggregators we chose Goerli as the source chain and Mumbai as destination chain. For message aggregators, we chose Sepolia and Polygon Mumbai. The RPC URLs we used for testing were provided by
Alchemy and the aggregator URLs were taken from the aggregator docs. We used the SDK or API to interact with each aggregators, and Coingecko\textsuperscript{1} pricing API to collect token price data.

4.2 Benchmark Process

We collected results from running the benchmarking tool over a two-day window, polling a new collection of routes every 20 minutes. In total, this resulted in 360 collected routes across all token-aggregators. The collection window was from 2023-12-08, 18:33 UTC to 2023-12-11, 14:02 UTC. A single batch ran for about 41 seconds.

The message aggregators were statically benchmarked, where we only logged the reports generated by the scripts on a single run. This was possible due to the metric generated by the message-aggregators, such as the deploy cost, message cost, and gas fee, being dependent on mathematical equations and hence do not have any variance apart from network gas price, which is a parameter of the blockchain and not the aggregator.

To benchmark message aggregators, we created a simple number storage script that could work with an aggregator. At the time of benchmarking, the following were the token prices: 1 DAI / 0.9985 USD; 1 ETH / 2,234.26 USD; 1 MATIC / 0.8556 USD.

Although the codebase is configured to execute trades and monitor the latency, we were unable to benchmark them as protocols did not support cross-chain trades on testnets. We believe that testing latencies benchmarks a DEX and does not truly isolate the aggregator, which is what we are trying to evaluate.

4.3 Benchmark Criteria

One can interact with a token aggregator using the UI or through an SDK/API. On the other hand, one can interact with message aggregators directly through smart contracts, using their APIs and SDKs, or creating a transaction through Etherscan. Etherscan \textsuperscript{25} is an EVM network block explorer that allows users to inspect blocks, transactions, addresses, and even connect their wallet and send transactions and read transaction data from a network. We analyzed the qualitative and quantitative aspects of these aggregators. To do this, we broke down the theoretical analysis and practical benchmarking experiments, which is explained in section 2.3.1.

4.4 Hypotheses

Before running the benchmarks, these were our hypotheses:

1. API-based aggregators have tighter bounds on the relation between aggregator quote value and the actual value of a token, quoted by coin gecko.

2. Fee involved to be a function of the gas price on the destination network.

3. Gas estimates to be a function of the network they are executing on.

4. Open Sourced Aggregator have a lesser tight bound on the token value. We also expected there to be more variance in the token pricing.

\textsuperscript{1}https://www.coingecko.com/api\textsuperscript{25}
5 Results

In this section, we present an overview of the benchmark results.

5.1 Fees and Gas Price:

We present the fees involved with constant inputs. The routes were all generated within a 30-second window that ensure the network behavior was constant throughout. We also capture the influence of the gas price on the quoted fee.

Figure 5 contains the net fee charged by a token aggregator (in blue), and the gas prices of the source and destination networks (in grey) as the y-axis. We also present Table 3 for the same data measuring the mean and variance to measure the fee.

As the message aggregators were statically benchmarked, we list the operations performed such as deploying a contract and sending a transaction, the gas price of the network, the gas used, and the amount in USD to create a standard comparison across aggregators. We did the above for both version of CCIP in Tables 4, 5, Hyperlane in Table 6, and Hashi in Table 7.
Takeaway 1: XY and Socket do not charge a fee (cf. Figure 5 and Table 3).

Takeaway 2: When not accounting for the gas fee when computing the net fee, only the destination chain gas price affects the net fee. (cf. Figure 5 - (d), (h)) show the blue lines (aggregator fee) tightly following the destination network gas price.

Takeaway 3: When accounting for the gas fee, the source chain also has an impact, although minor (cf. Figure 5 - (a), (c), (e)) show the aggregator fee move in a similar pattern to the source chain gas price.
<table>
<thead>
<tr>
<th>aggregator</th>
<th>source-chain</th>
<th>dest-chain</th>
<th>net-fee ($\sigma$)</th>
<th>net-fee ($\mu$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoW Swap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>8.88</td>
<td>11.30</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>9.85</td>
<td>28.80</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>15.69</td>
<td>40.90</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>7.03</td>
<td>12.61</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>6.08</td>
<td>11.32</td>
</tr>
<tr>
<td>Uniswap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>0.98</td>
<td>1.38</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>0.00</td>
<td>0.09</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>0.60</td>
<td>1.63</td>
</tr>
</tbody>
</table>

Table 3: Average net fees for each aggregator, source-chain, and dest-chain combination.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Network</th>
<th>Gas Price (gwei)</th>
<th>Gas Used</th>
<th>Amount (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Fee in Native (ETH)</td>
<td>Sepolia</td>
<td>133.685882142</td>
<td>29081</td>
<td>8.686</td>
</tr>
<tr>
<td>Message Fee in LINK</td>
<td>Sepolia</td>
<td>133.685882142</td>
<td>47732</td>
<td>14.257</td>
</tr>
<tr>
<td>Deploy Sender Contract</td>
<td>Sepolia</td>
<td>133.685882142</td>
<td>864548</td>
<td>258.230</td>
</tr>
<tr>
<td>Deploy Receiver Contract</td>
<td>Mumbai</td>
<td>3.000000032</td>
<td>498019</td>
<td>0.001</td>
</tr>
<tr>
<td>Send Transaction Native (ETH)</td>
<td>Sepolia</td>
<td>128.968232136</td>
<td>230984</td>
<td>66.557</td>
</tr>
<tr>
<td>Send Transaction LINK</td>
<td>Sepolia</td>
<td>128.968232136</td>
<td>237100</td>
<td>68.320</td>
</tr>
</tbody>
</table>

Table 4: CCIP v1.0.0 Cost to Use

**Takeaway 4:** Cross-chain transactions have a higher fee than same chain transactions, across aggregators that charge a fee (cf. Figure 5 (c and d), (e and f), (g and h) show the latter having higher fee) and Table 3 shows cross-chain trades costing more for LiFi and XY.

**Takeaway 5:** Open Source protocols (Uniswap and CoW Swap) have a lower variance in fee charged among protocols that charge fee. Table 3 shows that CoW Swap and Uniswap have the smallest variance-mean ratio.

**Takeaway 6:** Chainlink CCIP and Hyperlane use equations to compute protocol

<table>
<thead>
<tr>
<th>Operation</th>
<th>Network</th>
<th>Gas Price (gwei)</th>
<th>Gas Used</th>
<th>Amount (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Fee in Native (ETH)</td>
<td>Sepolia</td>
<td>126.512107387</td>
<td>29081</td>
<td>8.220</td>
</tr>
<tr>
<td>Message Fee in LINK</td>
<td>Sepolia</td>
<td>126.512107387</td>
<td>47732</td>
<td>13.491</td>
</tr>
<tr>
<td>Deploy Sender Contract</td>
<td>Sepolia</td>
<td>126.512107387</td>
<td>864548</td>
<td>244.373</td>
</tr>
<tr>
<td>Deploy Counter Contract</td>
<td>Mumbai</td>
<td>3.0000000032</td>
<td>498019</td>
<td>0.001</td>
</tr>
<tr>
<td>Send Transaction Native (ETH)</td>
<td>Sepolia</td>
<td>126.533233854</td>
<td>230984</td>
<td>65.301</td>
</tr>
<tr>
<td>Send Transaction LINK</td>
<td>Sepolia</td>
<td>126.533233854</td>
<td>237100</td>
<td>67.030</td>
</tr>
</tbody>
</table>

Table 5: CCIP v1.2.0 Cost to Use

21
<table>
<thead>
<tr>
<th>Operation</th>
<th>Network</th>
<th>Gas Price (gwei)</th>
<th>Gas Used</th>
<th>Amount (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy Counter Transaction</td>
<td>Mumbai</td>
<td>3.0000000032</td>
<td>222699</td>
<td>0.001</td>
</tr>
<tr>
<td>Send Transaction to Counter</td>
<td>Goerli</td>
<td>3.000000206</td>
<td>74685</td>
<td>0.501</td>
</tr>
<tr>
<td>Send Transaction to Counter</td>
<td>Sepolia</td>
<td>113.57621251</td>
<td>67081</td>
<td>17.022</td>
</tr>
<tr>
<td>Interchain Gas Paymaster Fee*</td>
<td>Sepolia</td>
<td>113.57621251</td>
<td>224289</td>
<td>56.913</td>
</tr>
</tbody>
</table>

Table 6: Hyperlane v2 Cost to Use. *We got Gas Used from an ETHEREUM -> POLYGON transaction as testnet transactions do not charge a fee.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Network</th>
<th>Gas Price (gwei)</th>
<th>Gas Used</th>
<th>Amount (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy Yaho</td>
<td>Goerli</td>
<td>3.000000804</td>
<td>1004278</td>
<td>6.731</td>
</tr>
<tr>
<td>Deploy Yaru</td>
<td>Gnosis</td>
<td>16.52392343</td>
<td>953260</td>
<td>0.016</td>
</tr>
<tr>
<td>Deploy AMBRelay</td>
<td>Goerli</td>
<td>3.00000081</td>
<td>480194</td>
<td>3.218</td>
</tr>
<tr>
<td>Deploy AMBAdapter</td>
<td>Gnosis</td>
<td>15.962067614</td>
<td>1082831</td>
<td>0.017</td>
</tr>
<tr>
<td>Deploy Counter</td>
<td>Gnosis</td>
<td>15.962067614</td>
<td>212871</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Table 7: Hashi Cost to Use

fees. The data however is updated by a select committee [26].

**Takeaway 7:** Hashi provides block headers from various independent oracles [27]

**Takeaway 8:** Chainlink CCIP v1.0.0 from Table 4 and v1.2.0 from Table 5 has no difference in gas cost. The only difference in the amount is from different gas prices. The gas used is the same.

**Takeaway 9:** Hyperlane does not charge a fee on testnets. We used a transaction from Ethereum to Polygon that we found on the Hyperlane v2 block explorer that used the same input that we did, and scaled that to match our benchmark results (cf. Table 6).

**Takeaway 10:** Contract deployment cost is a large part of the cost in CCIP and Hyperlane. CCIP also requires a sender contract that Hyperlane does not (cf. Table 5 and Table 6).

**Takeaway 11:** Sending CCIP transactions with native tokens is cheaper than with LINK (cf. Table 5).

**Takeaway 12:** CCIP is similar to Uniswap as all computation and fees are on-chain. Hyperlane does not estimate fee on-chain [26].

### 5.2 Aggregator Quote vs. Coingecko

We use Coingecko API as a common source to compare all the aggregator quotes, due to its reputation within the community. Coingecko is also integrated with more than 50 crypto exchanges and has had a historic 99.9% up-time of over 10 billion calls a month. Not needing the creation of accounts or API keys was a key decision point. Therefore, we compare the values provided by Coingecko against the quoted value of

---

2 [https://github.com/smartcontractkit/ccip/blob/6c9a0a2fc3b0d20184e602f6451dd39081008ce4/contracts/src/v0.8/ccip/onRamp/EVM2EVMOnRamp.sol](https://github.com/smartcontractkit/ccip/blob/6c9a0a2fc3b0d20184e602f6451dd39081008ce4/contracts/src/v0.8/ccip/onRamp/EVM2EVMOnRamp.sol)
aggregators. The aggregators were instructed to maximize trade value. Figure 6 shows the movement of the aggregator quote price (in blue) against the Coingecko price (in yellow). Figure 7 shows the difference in the quotes by subtracting the Coingecko price from the aggregator price. Similarly, in Table 8, we compare the deviation between the aggregators and Coingecko’s values.

Figure 6: Aggregator Quote vs Coingecko Quote
Figures 6 and 7 show that the quotes of CoW Swap, Socket, and XY are almost the same as that of Coingecko, with CoW Swap having more variance.

Uniswap from Figure 6 (b) has a minor drop which gets corrected naturally. LiFi’s pricing graph is the most interesting (Figure 6 (c)) because of the downtime that they had (the straight line in the beginning) and the tail end of the graph having LiFi in figure 7(c) quotes about $20 more in value until they correct in the ending of the tail.
Table 8: Price differences of Aggregator Quote (USD) vs Coingecko (USD)

<table>
<thead>
<tr>
<th>Name</th>
<th>Src Chain</th>
<th>Dst Chain</th>
<th>Over (µ)</th>
<th>Over (σ)</th>
<th>Under (µ)</th>
<th>Under (σ)</th>
<th>Diff (µ)</th>
<th>Diff (σ)</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoW Swap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>0.00</td>
<td>0.00</td>
<td>11.35</td>
<td>9.20</td>
<td>-11.35</td>
<td>9.20</td>
<td>360</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>13.53</td>
<td>11.93</td>
<td>35.20</td>
<td>43.28</td>
<td>-29.59</td>
<td>43.77</td>
<td>304</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>0.00</td>
<td>0.00</td>
<td>48.11</td>
<td>39.94</td>
<td>-48.11</td>
<td>39.94</td>
<td>305</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>5.63</td>
<td>4.20</td>
<td>7.13</td>
<td>11.05</td>
<td>-1.16</td>
<td>10.66</td>
<td>359</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>6.25</td>
<td>5.01</td>
<td>7.86</td>
<td>12.04</td>
<td>-0.81</td>
<td>11.61</td>
<td>358</td>
</tr>
<tr>
<td>Uniswap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>6.88</td>
<td>5.65</td>
<td>16.71</td>
<td>27.57</td>
<td>-9.45</td>
<td>25.58</td>
<td>341</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>5.97</td>
<td>4.23</td>
<td>6.34</td>
<td>8.61</td>
<td>1.36</td>
<td>8.63</td>
<td>360</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>7.02</td>
<td>4.85</td>
<td>7.31</td>
<td>12.21</td>
<td>2.49</td>
<td>10.35</td>
<td>359</td>
</tr>
</tbody>
</table>

Takeaway 13: Figure 7 and Table 8 show that API-based aggregator quotes tend to be closer to Coingecko Quotes.

Takeaway 14: Table 8 shows that API-based aggregator quotes tend to have higher variance than Open Sourced ones, notably CoW Swap.

Takeaway 15: There is more variance in the quotes when the aggregator quotes under Coingecko than when they quote over Coingecko pricing Table 8.

Takeaway 16: Table 8 shows that API aggregators may not have 100% availability and can go down, or not produce a route when there exists a route.

Takeaway 17: Figure 7 most notably shows that Lifi went down from approximately “2-09 00” to “12-09 12”, as indicated by the straight line in the graph.

Takeaway 18: Figure 7 and Figure 6 show that CoW Swap was unaffected by the “12-11 00” price drop across all the API aggregators and Uniswap.

Takeaway 19: Figure 7 shows the different aggregators can use different protocols to quote prices and still have the same price trends.

Takeaway 20: Figure 7 depicts price trends from different aggregators may not follow the same movement as a common 3rd party.

Takeaway 21: Figure 7 and Figure 6 quote value recovery from the “12-11 00” price drop was immediate by all aggregators but Lifi.

5.3 Variance in Aggregator Quote vs. Coingecko

We go in-depth into the quotes and find the variation between quotes and the Coingecko quote.

We decided to look more into the variance in quotes and thus split the quote difference in terms of % difference between the aggregator and Coingecko prices.

Figure 8 shows the difference as a histogram and they are also represented in Table 9 for aggregator quotes over Coingecko and Table 10 for quotes under.
<table>
<thead>
<tr>
<th>Aggregator</th>
<th>Source chain</th>
<th>Destination chain</th>
<th>&lt;1%</th>
<th>&lt;5%</th>
<th>&lt;10%</th>
<th>&lt;20%</th>
<th>&lt;30%</th>
<th>&gt;=30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoW Swap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>LiFi</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>14</td>
<td>74</td>
<td>53</td>
<td>26</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Socket</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>18</td>
<td>70</td>
<td>56</td>
<td>31</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Uniswap</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>12</td>
<td>38</td>
<td>33</td>
<td>17</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Ethereum</td>
<td>26</td>
<td>82</td>
<td>73</td>
<td>44</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>XY</td>
<td>Ethereum</td>
<td>Polygon</td>
<td>17</td>
<td>80</td>
<td>84</td>
<td>61</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9: Count of times Aggregators quote over Coingecko by percentage ranges

![Total Over/Under Quotes vs CoinGecko for Each Range Across Protocols](image)

**Figure 8: Quote vs Coingecko Pricing**

**Takeaway 22:** A particular token pair might give the trader more or less value when traded from one network to another than within the same network (cf. Tables 9, 10 and Figure 8).

**Takeaway 23:** Excluding outliers, most of the variance in API token-aggregators are within 10%, while non-API aggregators are around 10% (cf. Tables 9 and 10).

**Takeaway 24:** Most aggregators quote under the Coingecko value than over. The average quote difference combining the two seems to be in the range of 1-10% (cf. Figure 8).
5.4 Latency

We measure the latency of an aggregator quote, and plot them against the aggregator fee to find correlations between token price movement, fee, and the latency. We study the influence of the quote value (and hence price movement) on the estimation period from the aggregators.

**Takeaway 25:** API latency of cross-chain transactions is more than same chain, and can be anywhere from 2x to 10x longer. (cf. Table 11 – LiFi, Socket, and LiFi).

**Takeaway 26:** Variance in cross chain transactions is about 2x that of same chain. (cf. Table 11 - LiFi, Socket, and LiFi).

**Takeaway 27:** API aggregators tend to have lower variances in latency than the Open Source Model Aggregators. (cf. Table 11 – CoW Swap and Uniswap).

**Takeaway 28:** Ordering latencies: API based aggregators (same-chain) < CoW Swap < Uniswap < API based aggregators (cross-chain) (cf. Table 11).

**Takeaway 29:** CCIP waits for source chain block finality to complete a CC-Tx, while the others do not. This explains the higher latency and increased security against block reorgs and forks [28].

**Takeaway 30:** Hyperlane is broken into stages “Sent, Finalized, Validated, and Relayed”. Validated is always 5 seconds long and Finalized is between 15 and 100 seconds. Relayed is the total time between the creation of the source transaction and
the destination transaction.

6 Discussion

6.1 Token aggregators

Our hypotheses presented in Section 4.4 could more or less be reasoned by the centralization of API-based aggregators, as this would allow them to aggregate quotes from several DeFi protocols and return the best results, by also doing additional computation and other analysis on the source data before returning them to the user.

We used Coingecko as a common denominator to get the USD value of a token, which we used to compare the pricing of a token across aggregators, as there was no reason to believe that they all use the same backend to price tokens. This common peg also allows us to measure the movement of an aggregator’s quote price.

An argument can be made here that it is not the aggregator that sets the price of a token, but the protocol that they are routing the quote to. Our reasoning for comparison is that it is the aggregator that filters through a list of quotes from different protocols, and as such they are influential in determining the price of the token.

We set the routing type (if any) to give us the highest token value for the tests so as to get an estimate of the maximum values they each provide. The idea is that all aggregators choose the same protocol because there can only be one protocol that gives the highest value and, if there are multiple protocols, their quote values for a token must be the same.

For API-based Aggregators, as expected, not all aggregators used the same protocol and this caused variations in quotes. Across the board, we see that the pricing from an aggregator does generally move with Coingecko’s USD quote of ETH (Takeaway 17). In most cases, the quoted value from the protocol is lower to that of Coingecko (Takeaway 20), and there is difference in ETH:USD values across different networks with the same aggregator. When the DEX is also a constant, there is still some difference in the token value across chains (Takeaway 19).

When comparing cross-chain evaluations, we did not account for loss of token value from gas fee being paid to the protocol. When factoring in the gas fee that the trade value drops even further, which is expected in a cross-chain token transfer (Takeaway 2 and 3).

Cross-chain transaction used up more gas, as expected, and the gas price of the destination chain was more influential in the trade. We attribute that to the protocol/aggregator being the creator of the destination transaction and not the user. This offloads the gas payment of the transaction to the aggregator, who, in-turn, charges the user for it by increasing the trade fee. There are instances where an aggregator is not involved in the trade except for generating the quote, in these cases such as XY and Socket we see that the cross-chain fee is negligible, except for when they use a cross-chain liquidity protocol.

---

3https://explorer-v2.hyperlane.xyz/message/0x39e0b348e6b4a28a5a9bc9e11710121c5794778b87d4460cc301ec5b1101955
4https://explorer-v2.hyperlane.xyz/message/0x4db98e65e9ec8a63b9f24cc2814cdb978d26c9e4154c6f00d9203ef7e38909
In open source model aggregators, the movement of the token value pair is more erratic but within a short window. This is possibly due to the lower or lack of fee. As expected, Uniswap has the lowest fee as the protocol does not take fee when interacting from an SDK or smart contract and only does when interacting from their UI. CoW Swap’s fee includes the gas fee and when accounting for that the token value offered by it is superior to the same chain swaps from the API based aggregators.

Open source aggregators simply optimize the given inputs and try to provide the best value that matches the input requirements. Usually this is managed to be kept close to the actual value, but at times can slip, as seen in Figure 8.

Aggregators that have off-chain interactions have recovery strategies when several protocols quote under (as seen in Takeaway 18). It appears that many protocols were linked to the pricing quoted by Uniswap and when Uniswap slipped it impacted the pricing quoted by all aggregators except CoW Swap. We can see that all the aggregators except LiFi managed to recover before Uniswap recovered.

Takeaway 16 shows that the price movement across different protocols used by different aggregators seem to follow a similar pattern, this could be due to a good amount of overlap between protocols or simply several protocols using the same pricing oracle. This can be a problem if the source protocol has compromised oracles.

We also expected the Fee to be a function of the gas pricing or have some other model. The reasoning behind this stems from the centralized architectures of API based aggregators. The centralization provides them with more computer power and centralized data sources that allows them to run simulations and perform more complex computations than decentralized open sourced aggregators. This would explain the Fee structures seen in Takeaways 2 and 3.

Takeaway 4 can be explained by the fact that the transaction on the destination chain has to be created by someone that is not the user. It is either the Aggregator that is involved in the transaction or some other protocol. This would require the destination transaction sender to pay for gas on the destination chain on behalf of the user.

Takeaways 22 and 23 show that the average latency of a single-chain transaction is lower than multi-chain transactions. Open source model aggregator latencies are higher than the API based aggregators and are similar to API based aggregator cross-chain latencies. These follow the expected results. There also appears to be no correlation between token or gas prices and API latencies.

As aggregator quotes vary drastically with Coingecko quotes, we recommend having different thresholds for token values that they obtain from different sources. This is especially true when considering automated trading.

### 6.2 Message Aggregators

Users interact with on-chain contracts and not APIs, which reduces the variance in fee and time taken for block confirmations. This is because you can not really have complex computation that changes the fee estimation, as that would require consistent off-chain probing that increases the fee. As such, the computation of fee is usually fixed or has a deterministic function in the case of CCIP that uses the length of the CCTx message.

Chainlink CCIP generates a fee mathematically and is dependent on the length of the message, the destination network, and the token used to pay the fee. All the data
and code used to compute the fee is embedded in the smart contract\textsuperscript{5}. (Takeaway 5)

As it uses an equation to compute the fee, and there are not enough variables to test programmatically, we decided to look into it from a theoretical view.

Hyperlane uses a set of equations to compute the fee, which is then settled by the relayer after they dispatch the destination chain transaction.

Takeaway 9 shows that deploying Contracts on the networks seems to influence the cost to use a protocol the most, with CCIP requiring a sender and receiver contract, while Hyperlane and Hashi only require 1. This is due to the difference in architectures where CCIP can be automated on-chain to send cross-chain messages and is more of a protocol than Hyperlane which is an application.

Takeaways 24 and 25 show that the latencies are constant with CCIP using block finality of the destination chain, and Hyperlane uses a consistent time-split through various phases of a transaction.

6.3 Summary

Aggregator quotes varied throughout, and in this section we try to look into some reasons that could explain that behavior.

While (Takeaway 12) shows that the API aggregator quotes were closely bound to the Coingecko Quotes, there was a good amount of price fluctuation. The price fluctuation of variance seems to stem from some further-down-the-line protocols that these aggregators in-turn rely on, Takeaways 16 and 17 show that.

Furthermore, API aggregators do not have 100% availability as seen from the number of reports generated in Takeaway 13. This is not due to a pool being unavailable but the Aggregator not working correctly, this is proof by other aggregators still creating routes in the same time-frame that another aggregator is down. We got lucky with an aggregator (LiFi) going down for a few hours while other aggregators kept functioning (Takeaway 14).

There was another instance where the Aggregators relied on a common low level protocol that influenced an accidental price drop across all aggregators (Takeaway 15). Notably, CoW Swap (an open sourced model aggregator) was not affected by this. Uniswap (open sourced model aggregator) seems to have a price slip and that in-turn affected the quotes of all other aggregators as it seems they all relied on Uniswap or on another protocol that relied on Uniswap for pricing (Takeaways 16 and 17).

However, these API based aggregators seem to have certain recovery strategies (Takeaway 18) where they rely on a lot of sources such that even if most of them go down or misquote it is only a few instances where the pricing is off. All the aggregators except LiFi recovered within the next route generation.

Looking into the quotes, we see that the quotes are not perfectly overlapping with Coingecko’s quotes and that there is a difference. The difference is usually within +/-10% (Takeaway 20 and 21). Most aggregators quote a value lower than Coingecko’s quote. There is also a notable point where a route may result in a user getting more tokens than the value of the token on the source network, (Takeaway 19). This could be due to different protocols on different networks using different sources for pricing.

\textsuperscript{5}https://github.com/smartcontractkit/ccip/blob/81c0aea2a6c3f744f8f19520278361d453ad481f/contracts/src/v0.8/ccip/PriceRegistry.sol#L47
and some may lag behind the other. There are instances when it is profitable to swap to a different chain than on the same chain, although the additional fee in the cross-chain transaction can offset the profits. Moreover, if a transaction gets reverted, most protocols just return users fund in a swap token like WETH or USDC.

6.4 Future Research Directions

Our study provided the first comprehensive benchmark on cross-chain bridge aggregators. We compile a list of promising future research directions:

- Support for more aggregators and parameters: work on augmenting this work should be done in order to provide even better insights for practitioners on choosing the best aggregator for their needs. Similar to [29], a framework for choosing an aggregator should be provided, supporting the main aggregators available in the industry and academia. For this, it is instrumental to test our hypothesis on production chains (mainnets).

- Security and privacy research: research on security and privacy in cross-chain technology is still scarce, but recent developments are promising [4]. Future work is on exploring privacy mechanisms for bridge aggregators, and performing a comprehensive security analysis of specific solutions using varied techniques such as cross chain models [6].

- Enterprise-grade integration: enterprises need connectivity to blockchains to satisfy their needs for modernizing their financial services. For this, enterprise-grade infrastructure providers play a pivotal role [30]. However, aggregators are still not ready to be adopted by companies needing to abide by different laws and regulations. Research to understand enterprise needs in terms of organizational and legal interoperability [7] is needed to create a well-defined list of functional and non-functional requirements for aggregators.

7 Conclusion

Aggregators are here to stay, as part of the ever-evolving mission to make blockchain interoperability more user-friendly. In this study, we performed the first benchmarking study of the bridge aggregators, showcasing a holistic analysis of the evolving landscape.

Most of the expectations coming into the project were consistent with the results. API aggregators usually had the lowest latencies but had more downtime as expected from centralized services. The decentralized and open-source models had higher latencies but the lowest downtime. Aggregators also tended to quote values under that of coinageco quotes. This is expected, as the protocols that they interact with have varying degrees of slippage on varied quote values. The open-source aggregators had a lesser variance in their price quotes, which was surprising. Message aggregators tend to be larger code chunks that take longer periods to execute while being more expensive in
terms of gas and fee. This is consistent with the expectation of a generalized messag-
ing service compared to that of a highly engineered single task service like a token
 aggregator.
The benchmarking results contribute to helping academics and practitioners alike to
reason systematically about aggregators. We provide future research directions, includ-
ing cross-chain privacy and studies on the feasibility of enterprise-grade aggregators.

References


blockchain interoperability: Past, present, and future trends,” New York, NY,


dark is the forest?” in 2022 IEEE Symposium on Security and Privacy (SP).
IEEE, 2022, pp. 198–214.

“Hephaestus: Modelling, analysis, and performance evaluation of cross-chain
transactions,” 09 2022.


[14] [Online]. Available: https://docs.cow.fi/

[15] [Online]. Available: https://0x.org/docs/introduction/introduction-to-0x

[16] [Online]. Available: https://docs.uniswap.org/contracts/universal-router/overview
A Appendixes

In this appendix, we list relevant aggregators that we did not benchmark.

B Node Aggregators

Node aggregators provide a way for Users to interact with blockchains. These interaction do not influence state changes on the chain apart from allowing users to send transactions and read contract data. They work as user experience aggregators by abstracting some of the harder parts of blockchain interaction and try to make it easier for users to get NFT data, historic on-chain data, send batch transactions, to name a few. This allows them to be more centralized as they execute signed transactions that can not be spoofed and let users read from chains.
B.1 Multi-chain APIs

These are Clients that are usually user run and have additional features baked into them, in addition to standard blockchain transactions. They perform key management, UI work, interaction with non blockchain programs. They are also scriptable and hence configurable.

B.1.1 Hyperledger Cacti

Blockchains were created to be decentralized state machines to minimize single point of failures and centralization, the cacti project decided to move that a step further and treat each blockchain as an independent system and tries to minimize single points of failures where each point is a blockchain.

Cacti is a framework that allows developers to abstract multi-chain operations, key management, certificates, plugins and other complicated functionalities across different DLTs, streamline them, and make it easier to develop applications and improve user experience by abstraction.

The primary objectives are:

1. Represent the internal state of a chain to other chains
2. Guarantee sound business logic operating on multiple chains

These are achieved through “plugins”. The different classes of plugins are:

1. Business for asset exchange
2. Connectors for clients
3. Tooling for the development of business logic

Plugins function similar to independent blockchains (called consortium in cacti vocabulary), each with their own validators and verifiers that handle transactions between users and the ledgers they are interacting with.

Plugins are modular and can be written for any tasks and to name a few - relaying transactions across blockchains, escrow contracts, cryptocurrency trade validation, and aggregated routing prices. This allows developers to make use of unique features on some DLTs.

This project is of type Open source Model 2.5 and supports multi-chain architectures so users and developers use pre-existing contracts without re-writing the codebase.

Each interaction is a custom configuration decided by the parties involved. It is an active protocol as opposed to passive ones where people build over.

The only “standardization” across consortium is the use of a Secp256k1 curve key. Other aspects such as the consensus, node size, validators, are configurable and decided by the members of the consortium. It supports value, data transfers and even data merges.

Cacti nodes can contain have API servers that each handle a different job - allowing a node can handle multiple protocols.
A project called “weaver” was introduced at GHyperledger and later merged with cactus to form cacti. This brings in relays and lets the cacti nodes be more distributed and not only use trusted members in the consortium.

B.1.2 Quant Overledger

Quant Overledger is an SDK with a proprietary backend that aims to create a unified set of operations and data structures for multiple DLTs (Ethereum, XRP, Substrate, Bitcoin).

The user creates an API key that they use to make use of the overledger api. The interactions between the use and the server is using OAuth2 AWS.

There are two kinds of transactions:

1. Prepare - Creation of a DLT specific transaction by the user or overledger
2. Execute - Execution of a DLT specific transaction
3. Prepare and Execute - Create and Execute a transaction

The user creates a ‘standard’ payload that we call an overledger-tx which is a user readable transaction with information about the destination DLT, how urgent a transaction is (gas), address to be interacted with, etc. and sends it to the overledger api. Overledger then generates a DLT specific transaction from the overledger-tx along with an overledger-requestId that is used to reference the transaction. A user can also create a DLT transaction and submit that to overledger instead of using the API.

In the execute phase the user signs the transaction and sends it to overledger which then executes the transaction on the destination DLT. The overledger-requestId can be also used to query overledger for the transaction and view the transaction information.

Overledger is queryable (manually or automatically called subscriptions) for all the incoming and outgoing transactions from an address along with the events emitted by a contract. It also lets users read data from a smart contract, get the address balance from smart contracts, get UTXO information from the bitcoin network, and retrieve block information.

Overledger supports ERC20 and ERC721 with their custom APIs to get token balances, sending tokens to another address, etc. They also have a QRC20 and QRC721 token that provides users with features like what you’d expect at a bank, with credit, debit, and shared accounts in addition to ERC features.

It lets users interact with bridges in v3.0.0 but there isn’t much documentation about that. There is also no information about batch transactions, executing conditional sequential transactions, and is not customizable by a user.

B.1.3 Blockdaemon - Universal API

The universal API from Blockdaemon is a multichain API that integrates 15 protocols for REST and streaming data. The supported chains are EVM based, Bitcoin based, Solana.
This protocol is a midway-point between Overledger and a traditional RPC url connection. It allows developers to get fee estimations, token balances, transaction constructions, event subscriptions from multiple chains with a single API call. It’s financial model is in Compute Units.

Not all networks get the same features for instance only Bitcoin, Ethereum, Polkadot, Matic, and Solana get access to the transaction construction service. Blockdaemon has partnered with Bloxroute to faster execute Ethereum transactions. This also lets the users see the lifecycle through different stages of the transaction. It lets users create unsigned transactions in addition to submitting a signed transaction to execute on a network.

All the supported networks get access to websockets that subscribe the user to block identifiers and transactions. It supports all token assets across chains and lets users view all the transaction made in a time period along with the inputs and outputs of a series of transactions.

B.2 Multi-Chain API Providers

These are services run by companies that host blockchain nodes and create URL endpoints for users to interact with blockchains. They are not configurable as they are simply REST apis but are extremely easy to use.

The APIs mentioned in this section function very similar as shown in the table below. We proceed to analyze and compare only the unique features of each

<table>
<thead>
<tr>
<th>Network</th>
<th>Blockdaemon</th>
<th>Infura</th>
<th>Alchemy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks Supported</td>
<td>23</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>Dashboard Analytics</td>
<td>Good</td>
<td>Extensive</td>
<td>Extensive</td>
</tr>
<tr>
<td>Types</td>
<td>EVM, BTC, Solana, etc</td>
<td>EVM, IPFS, Filecoin</td>
<td>EVM, Solana</td>
</tr>
<tr>
<td>Tx, historic data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Monitor events and tx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Financial Model</td>
<td>Compute Units</td>
<td>Rate limits</td>
<td>Compute Units</td>
</tr>
<tr>
<td>Batch requests</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RPC functions supported</td>
<td>Full list</td>
<td>Full list</td>
<td>Full list</td>
</tr>
<tr>
<td>Non standard apis</td>
<td>Staking, Universal</td>
<td>None</td>
<td>Debug, Trace</td>
</tr>
<tr>
<td>NFT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

B.2.1 Blockdameon - Native + NFT + Staking API

Blockdaemon provides the highest number of API connections with a total of 463+ endpoints (main-net and test-nets).

Their Stake service is unique among the trio compared and it lets users interact with multiple networks to perform staking, signing stake transactions, and let users sign stake-transaction on any custodian. It lets users batch staked-transactions making the creation of validator pools for a protocol more gas efficient.

It also has direct API commands to interact with Solana staking for stake, de-active a stake, and withdraw it. On the supported networks it lets users setup a stake fee recipient
address, automatic reward accruing, withdrawing the accrued funds. It supports static and dynamic growth on validators.

The NFT API lets users access on and off-chain data for NFTs on multiple networks. It lets them obtain information such collection owners, current owner, creator of NFT, etc. It provides support not only for ERC-721 but also ERC-1155 NFTs.

B.2.2 Infura

Infura’s IPFS API is a limited IPFS service that lets users run dedicated gateways that are read-only. It doesn’t support pinning services on IPFS desktop but users can do that with the Infura API. Users have access to security lists, pin control, quota restrictions.

The NFT API lets users get NFTs owned by an account, the data of the NFT, lowest sale price in the last 7 days, and view the transactions on an NFT.

B.2.3 Alchemy

Alchemy provides users with the most features among the API aggregators that we analysed:

The NFT API on Ethereum, Polygon, Arbitrum, and Optimism for ERC-721 and 1155 allows users to read contract data for NFT metadata. They maintain a local IPFS gateway to provide better support for the hosted NFTs. They provide additional features such as token ownership, token gating, collections, sales, spam detection, and marketplace information. It also supports a subset of the projects created before the standardized contracts such as CryptoPunks and CryptoKitties.

Alchemy’s "Transfers" API that lets users fetch historical transactions quickly without the need to scan the entire chain. The Tokens API, like the NFT API lets provide support to tokens stored at an address. The "Debug" API and "Trace" API let users go through lifecycles of a transaction such as internal transactions.

The "Bundler" API allows for account abstraction (ERC-4337) that lets smart contract wallets interact with a Gas Manager, the ERC-4337 contract that lets entities sponsor gas fees for others.

C Centralized Token Bridge and Pool Based Aggregators

C.1 Jumper (Li.fi)

Jumper is a data mesh of cross-chain liquidity sources: cross-chain liquidity networks, bridges, DEXs, bridges, and lending protocols.

The Aggregator team manually grades the bridges in addition to an automated grading based on qualitative and quantitative factors. Qualitative factors are, for example, trust assumptions and attack vectors. They are also ranked with default prioritization of security, speed, and finally costs. However this is customizable by the user to better fit their use case.
Jumper supports bridging and swapping operations individually and “xChain Contract Calls” which is a merged transaction that executes a transaction after bridging tokens to execute cross-chain contract calls.

The back end is proprietary and not open sourced, this is the same for the liquidity provider and the cross chain message validators and indexers. However all the smart contracts used are public and audited.

The smart contracts on the liquidity use a EIP-2535 diamond architecture that makes upgrading the protocol easier, cheaper, and lets smart contracts scale so that the protocol can evolve over time. Adding bridges and DEXes to the list of supported ones is easy and possible.

There exist two kill switches:

1. API: Route calculation ignores a bridge
2. Smart contract: Ignore kill switch checks for the protocol

Looking at the cross-chain messaging repository we see that the gateway contracts called “Mailbox” are a 2/3 multisig and can route transactions to any contract’s functions. There is no custom ABI support available. It uses EIP-6170 for the messaging bridge.

While their protocol is centralized in the backend for speed and it being their competitive edge, they say that should a way to decentralize it be possible without much loss in value they may try to move over to that.

The SDK only works with mainnet and to interact with the testnet we had to send http requests to a different URL. Moreover there was heavy rate limiting and we couldn’t send more than 4 queries without having to wait for a few minutes. This made it hard to test the protocol, but they have a no rate limit version on their website which we used to test the protocol. Their protocol has limited support for testnets and after working for a WETH -> USDC swap it stopped working and the smart contract has no transactions on it since then.

C.2 Socket (Bungee)

Socket provides 2 kinds of cross chain interoperability services:

1. Liquidity Layer
2. Data Layer

The Liquidity Layer allows users to optimize their swap or bridge operations by computing the best route as requested by the user for security, cost, and speed. The architecture design is modular without upgradability and there is no information about the Data Layer.

There are some additional services offered that improve developer and user experience:

1. The liquidity layer has a centralized tool called "Refuel" which lets users native tokens on the network they’re swapping assets to in addition to the preferred swap token on the destination network.
<table>
<thead>
<tr>
<th>Name</th>
<th>Num Txs</th>
<th>Send Tx</th>
<th>Dest Tx</th>
<th>Swap+Bridge</th>
<th>Tokens Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>1</td>
<td>Swap+Bridge</td>
<td>0</td>
<td>Yes</td>
<td>Limited</td>
</tr>
<tr>
<td>Multi</td>
<td>&gt;=2</td>
<td>Swap+Bridge</td>
<td>Swapping</td>
<td>Yes</td>
<td>All</td>
</tr>
</tbody>
</table>

Table 12: Socket(Bungee) Transaction Types

2. Socket-plugins make it easier for developers to integrate Socket into their dApps.
3. A block explorer (Socket Scan).

Their routing quotes are generated by a proprietary route quote generation software that tries to optimize the assets by computing the best pairs to swap on the best computed bridge. In the event of low liquidity the user may get protocol wrapped tokens from the DEX used to swap, this is because Socket is an aggregator and does not facilitate the trade and they simply redirect your swap requests to the most optimal DEX from their calculations.

SocketDL enables smart contracts to read state and perform arbitrary communication with other contracts across different chains should they be implemented to support the SocketDL framework. The information we obtained is directly from their github repository as there is no available documentation for it.

The data layer has open sourced relays that act individually and not networked together. They also seem to be customizable by dapp developers. Message interaction is similar to Chainlink oracles and uses predefined function names and does not allow arbitrary calls as the SocketDL ExecutionManager can only interact with functions named “inbound” and receive transactions from a contract through the function “outbound”. This limits programmers to being able to use only 1 inbound function but this can be worked around with decoding the bytes data payload and then routing the transaction. There is no limit to the number of outbounds. There is also no ABI decoding or encoding and all data is transacted with in bytes.

The protocol is very closed source and does not have any source code for the backend, token prices aggregation, gas estimation, and routing. The only open sourced components are the smart contracts, which have been audited by different parties.

While their docs is not up to date with the SDK it is possible to implement the swaps and routes with their SDK from their github page which is what we did. Using the SDK makes the swapping much easier to implement but fetching a route is the same with both methods. Users require a dedicated API key to fetch routes as the public one listed on the website gets you routeless queries as it is only intended for testing purposes.

C.3 XY Finance

This protocol is a DEX and bridge aggregator that tries to optimize for best routes and liquidity providers. It does not do messaging. It claims to be secure, fast, and cheap to use because of the unique construction of a XSwap (aggregator) and a YPool (liquidity pool). Like other protocols the backend is proprietary.
XSwap is an atomic swap based aggregator that usually completes in 1 swap. The swapper tries to swap the users token to a bridge token first (USDC/USDT) and then tries to swap it to the user’s desired token. If the second part fails then the refund is issued in the bridge tokens. The routing parameters can be configured by the user. There is a Withholding Fee and a Bridge Fee.

YPool allows users to perform yield based defi in addition to swaps. This is where the liquidity is stored and maintained, and should the liquidity break the threshold then XSwap is automatically called and it balances it. There is a common liquidity for all the chains, this allows you to create a new token and make it multichain.

There is also a settlement chain that is run by the protocol which is a local chain for quicker logging of events and finalizing swaps. There is only one executor that is run by the protocol that interacts with the mainnets. There are multiple validators for onchain and settlement consensus and it constantly validate the data of the requests from the executors where necessary. Validators can be run by anyone.

There is only one audit from over a year ago. Smart contracts are proxyable and simple, the smart contracts are few and simple to read as most of the work is done offchain by the settlement chain and only the swaps occur on-chain. This should help reduce the total gas cost for a user.

No SDK but the API worked. The API was not the easiest to implement as there were no guides but the API endpoints were clear on the developer docs. They only support mainnets.

D Decentralized Liquidity Aggregators

D.1 CoW

CoW Swap is a permissionless protocol that uses Batch Auctions to find prices through what it calls "Coincidence of Wants" (CoW).

CoW is a 2 phase process split as Phase 1 - "batch auction phase" which is run by the protocol and Phase 2 being the "settlement phase" which is run by the solvers.

The batch auction allows the protocol to aggregate liquidity from multiple on-chain sources while the settlement phase allows solvers to compete with each other to find the optimal settlement for a given batch auction. The off-chain transactions that are signed by the traders can be bundled into a single transaction that not only saves gas but also allows for more constant trade prices.

As it supports off-chain orders the trades are gasless. User are instead charged a protocol fee that is taken from the sell token in the transaction, it even allows for partial transactions to be executed and the fee is pro-rated. This protocol fee makes up for the gas fee. The off-chain segment allows for more user privacy and lets users bypass MEV and front-running. The off-chain settlement mode allows for better abstraction and scaling across different chains should the pricing of tokens not be manipulated by attacks.

The architecture apart from being permissionless and open has 3 smart contracts:

1. Settlement - maintains orders and interacts with liquidity
2. Authentication - solver addresses (proxy contract)

3. Vault Relayer - funds are handled here

Users are guaranteed to not get a worse price than the one they requested. The settlement contract allows solvers to execute arbitrary calls on on-chain contracts, so it is an extensible contract that can interact with any on-chain contract if the protocol is extended to support it, this can include messages, off-chain transfers, etc.

There is no SDK but the API worked without issues. There are a lot of custom types involved in this protocol and we had to write a lot of code to convert them to the types we were using. The interaction looks messy but it works well for testnets and mainnet. They have a block explorer that also helps viewing the transactions and it updates as different settlers work to complete the swap. It only supports single chain swaps.

D.2 1inch

1inch is a DeFi protocol that provides users with a dApp, an aggregation protocol, a liquidity protocol, limit orders, and p2p transactions. All the trades can be completed in partial. The off-chain trades also support cancellation and validation.

The dApp connects users to the smart contract and perform on chain swaps. It allows for connections with several DEXes and also has the ability to complete a swap across multiple liquidity sources. They also have an AMM that aims to allow users to trade, while protecting users from being front-run.

The aggregation protocol called PathFinder finds swap routes for a token pair and can split the trade across different protocols in a short period of time.

They have an off-chain swap called Fusion Swap that is similar to Cow and is executed off-chain by resolvers through a dutch auction. They also have RabbitHole, which aims to solve sandwich attacks for wallets that do not support Flashbots.

A unique feature of 1inch is that they allow P2P trading where users can simply trade ERC-20 tokens with each other.

All the code is open sources and there is a foundation that governs the project. It is also governed by 1inch “core contributers” and a DAO.

E On-chain Token and Pool Based Token Swap Aggregators

E.1 Uniswap

The Uniswap cross-chain swapper is an Open Sourced Model where the users handle the cross-chain transactions. Each user makes use of the Universal Router SDK to interact with the Unowned and Unupgradable Universal Router Smart Contract. There are many smart contracts involved with this protocol and the ones on the docs used for examples are not audited and users need to take precaution around frontrunning and other attack vectors. The protocol is community governed using UNI tokens.

The SDK performs gas estimation by
1. simulation the transactions based on historical data and current network conditions.

2. The current transaction data and function being called are also simulated.

3. L1 - L2 security is also taken into account

4. As the protocol uses swaps and ticks the base swap cost, the cost per hop, the cost per initialized tick, and the cost per uninitialized tick are used for estimation.

5. The tokens being swapped as the native swap token used is WETH, which uniswap converts into the highest liquidity pool before swapping

UniswapX is a new protocol from UNISWAP that’s based off CoW and offers non-custodial dutch auction based trading that aggregates on and offchain liquidity which internalizes MEV and offers features such as gas-free swaps and even cross-chain trading (not yet built at this point in time) on an open network that finalized using an optimistic model. The off-chain transaction handling allows for transactions to be merged, added, subtracted, split, etc that makes swaps quicker, cheaper, and more secure.

Using the SDK is simple with the extensive documentation with a small caveat that the ethers version needs to be v5, we expect that to be changed in the future with v6 out for a while now. As it is a same chain swap the swaps work at expected and we didn’t have any issues.

E.2 1inch

1inch is a DeFi protocol that provides users with a dApp, an aggregation protocol, a liquidity protocol, limit orders, and p2p transactions. All the trades can be completed in partial. The off-chain trades also support cancellation and validation.

The dApp connects users to the smart contract and perform on chain swaps. It allows for connections with several DEXes and also has the ability to complete a swap across multiple liquidity sources. They also have an AMM that aims to allow users to trade, while protecting users from being front-run.

The aggregation protocol called PathFinder finds swap routes for a token pair and can split the trade across different protocols in a short period of time.

They have an off-chain swap called Fusion Swap that is similar to Cow and is executed off-chain by resolvers through a dutch auction. They also have RabbitHole, which aims to solve sandwich attacks for wallets that do not support Flashbots.

A unique feature of 1inch is that they allow P2P trading where users can simply trade ERC-20 tokens with each other.

All the code is open sources and there is a foundation that governs the project. It is also governed by 1inch “core contributers” and a DAO.
F Message Aggregators

F.1 Hyperlane

This is an interoperability protocol that allows developers to build cross-chain dApps with either off-the-shelf modules or custom modules designed by the dApp’s developers. Hyperlane performs cross-chain interaction using Validator-Relayer networks, with Watchtowers being a Stake/Slash governor (Watchtowers and Stake/Slash are under construction) to reduce the likelihood of an operator misbehaving. Validators pick up transactions from the Mailbox (an on-chain smart contract that acts as a gateway) and sign them on the Mailbox Merkle roots, which are then picked up by relayers that deliver these messages to the recipient’s mailbox. Validators validate transactions individually and do not send transactions to a blockchain. Sending transactions is the job of the relayers, and there is a trust assumption where the developer trusts that the relayer would forward their transactions and not take the fee and refuse to send.

The protocol is modular, allowing developers to customize the Interchain Security Modules (ISM), validators, relayers, etc. There are pre-configured ISMs that make development easier. It also comes with certain helper tools that make interaction with the protocol easier for developers and users.

The protocol also comes with certain helper tools:

1. **GasMaster** - lets the user pay for gas, the dApp pay, or manual payment.
2. **Explorer** - Lets you view executed transactions.
3. **ABI coder** - Lets you deal with data-type parameters and not bytes on function calls.
4. **Prebuilds (APIs)** - for message passing, token swapping, liquidity.

And some pre-built APIs that make integration of Hyperlane easier:

1. **Messaging** - Send and receive interchain messages.
2. **Accounts** - Call smart contracts on remote chains.
3. **Queries** - Query state on remote chains.
4. **Warp** - Move tokens between chains.
5. **Liquidity Layer** - Attach value with Hyperlane messages using the Liquidity Layer API.
6. **Interchain Paymaster** - Pay for message delivery on the origin chain.

Despite being a developer-focused interoperability protocol, it has some issues, such as the requirement for a smart contract to be created with the IMessageRecipient interface to interact with Hyperlane. The documentation of code is not the best, with most of it not being commented on or documented on their GitHub. Gas fees on transactions are fixed with no customizability, but refunds are available. Validators do not communicate...
with each other, and as Stake/Slash and Watchtowers are still being built, this leads to
easier network attacks should a group of validators collude with relayers. The GitHub
repository states that it does not currently support Sovereign Security Models. Addi-
tionally, many aspects of the protocol, such as the Liquidity Layer API, Stake/Slash,
Watchtowers, and ISMs (Aggregation, Optimistic, Wormhole), are still under develop-
ment.

Their documentation was good and we didn’t have any major issues with it. Their
block explorer really helped us debug and test the protocol. There is a v3 coming out
which should have simpler and more unified and elegant smart contract scripting than
the current version.

F.2 CCIP

From Chainlink, well known for their oracles and off-chain compute is CCIP, their on-
chain smart contract triggerable cross-chain messaging service that also handles token
transfers.

The architecture model is similar to their oracles and is decentralized throughout
and this increases their fee and latency, which is a lot on the testnets. The product is
also new and isn’t mature so we expect that the latencies should drop significantly. One
of the features they’ve added that isn’t present in their other products is the ability to
pay fees in Link or Native tokens. This helps automation a ton as it allows developers to
simply write code and forget. Also this project doesn’t use user APIs and works 100%
on chain similar to layerzero and others.

Works as intended but they had a package upgrade in the middle and some fea-
tures broke, but their docs had the updated code and we didn’t have any issues. Their
examples are well written and easy to follow.

F.3 Gnosis Hashi

The Hashi protocol is a suite of modules that when combined work like a “Redundant
Array of Independent Hash Oracles” (RAIHO) similar to a RAID for data storage.

It essentially functions as a Multisig-Bridge where signatures are from oracles that
fetch block headers of the chains that Hashi interfaces through with its bridges. It acts
as an abstraction layer and only validates block headers but not their contents. The ideas
behind this protocol are

1. To create additive security

2. Standardize the transfer and validation of blocks on one chain on another chain

The reasons being:

1. By increasing the security of the CCTX instead of replacing existing ones with
   others as the security gain from additive is superior to replacement

2. Developers shouldn’t have to design and build frameworks that support block
   transfers but should instead worry about building on top of that, this is why the
protocol ends at validation of blocks and anything on top of that such as block transaction handling is the responsibility of the dapp developers.

This allows for different configurations such as number of oracles, consensus, block challenges, etc. Message validation tools such as merkle proofs or zk proofs can be built on top by dapp devs. Governance only comes into picture when the local hashes (hashes stored by oracles) aren’t the same and so an agreed hash by the oracles needs to be sought after (a global hash).

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Owner</th>
<th>Anyone</th>
</tr>
</thead>
</table>
| Hashi                   | -            | 1. Build oracle adapters  
2. Query local and global block hashes |
| ShoyuBashi              | 1. Define target Hashi  
2. Define Hashi config  
3. Alter Hashi config | Query global block hashes |
| GiriGiriBashi           | 1. Initialize oracles  
2. Set challenges | 1. Query global block hashes  
2. Challenge oracles  
3. Resolve challenges |
| Yaho                   | -            | 1. Emit hash as event  
2. Store the message  
3. Relay a stored message  
4. Dispatch and relay messages |
| Yaru                   | Execute Yaho message | - |
| Hashi Zodiac Module    | -            | 1. Control address on another chain  
2. Define a Yaho to send messages |

The protocol’s redundancy however comes with a problem and that it is slow and not cheap to use. Because of this, dapps that rely on fast transactions and high throughputs may want to look at other options for a cross-chain messaging framework but dapps that desire increased security and uncompromised security guarantees such as a DAO on a chain that controls the operations of a protocol on other chains would feel comfortable using Hashi. This protocol is also under development and has had a past audit which is currently outdated as there has been development since.

This ones a curveball and their website only has the contract addresses for Gnosis and not their testnet. So we had to poll the mainnet faucet a bunch to have enough xDai to test the protocol. The documentation was good and we didn’t have any major issues with it. The code was well written and they used hardhat and not foundry so adding the repository to our project took a few extra steps such as installing another project called Solidity-RLP. But that wasn’t a big issue.