Socket Protocol: First Chain-Abstraction Protocol

Version 1.0

Socket Labs

Contents

1 Introduction

The blockchain ecosystem has experienced unprecedented growth and diversification over the past few years. This expansion has led to the emergence of numerous chains and layers, each offering unique features and capabilities. While this proliferation has brought increased functionality and specialization, it has also resulted in a fragmented landscape where users, liquidity, and state are dispersed across multiple networks.

In response, the industry has focused on building bridges and cross-chain messaging systems. However, this approach has introduced new intermediaries to connect networks, resulting in a "connected" ecosystem that still doesn't function as a cohesive unit for applications and users.

Using blockchain based applications remains challenging. Paradoxically, as the number of chains increases, the distance between apps and their users seems to grow. A key insight into this growing fragmentation is the number of intermediaries that now lie between applications and their users.

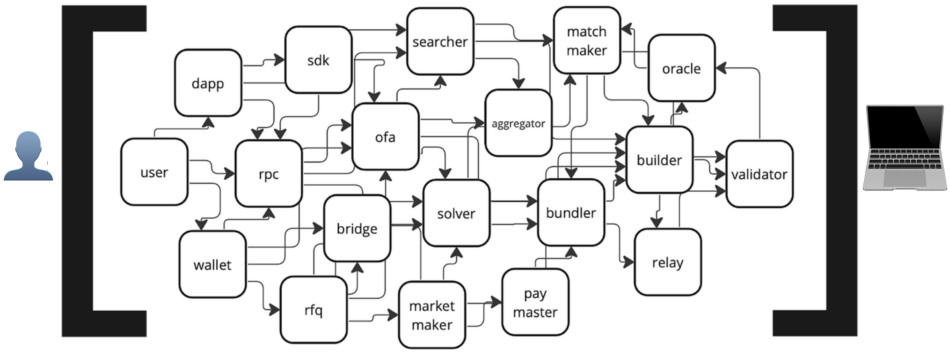

Figure 1: Illustration of blockchain fragmentation and intermediaries

Consider an app with its user on a different chain and the intermediaries user's transactions go through before they even land on the application:

- Bundlers & paymasters for accounts (on both chains)
- Relayers & liquidity providers for bridging
- Sequencers, validators etc. at the protocol level
- and more

What's worse is that each of these intermediaries are unaligned with application or user needs. As a result, users are forced to go through multiple layers of rent seeking intermediaries to be able to interact with application which simply act as hurdles user has to go through before they can use the application.

1.1 The Application-Centric Thesis

We expect the world to increasingly shift towards applications being the focal point for onchain user actions. Applications will become the first touch point for users such that users will interact directly with apps instead of going through intermediaries.

This means that applications will ingest the complexities arising from usage of these intermediaries while offering a chain abstracted experience for their users by controlling on-chain execution for their end users.

By doing this, applications become the gateway for user requests/transactions which manage how the actions actually get executed on-chain across networks. The execution of these transactions are managed by offchain agents that are "employed" by these app gateways, these intermediaries work for the application and users to help them perform actions they need to perform on-chain to complete the user's transaction across chains.

By controlling on-chain execution, applications can:

- Abstract away on-chain complexities for their users

- Ensure important elements are surfaced for their users
- Utilize intermediaries as smart agents within their apps, unlocking new possibilities

This approach not only simplifies the user experience but also opens up new avenues for application functionality and innovation in the blockchain space.

2 Application Centric Design

We think the ideal architecture going forward in this application-centric-world would enable a world where:

- Applications are the gateway
- Applications actively help users execute onchain actions across networks via aligned intermediaries
- Applications become the "glue" and onchain infra like chains, rollups are more like servers and execution-engines with various useful properties
- Applications no longer need to decide where to build
- Applications can be everywhere, all at once, maximising exposure, while providing a monolithic experience to end users

This is a general pattern across multiple industries as pointed out by Vitalik in his glue and co-processor blogpost.

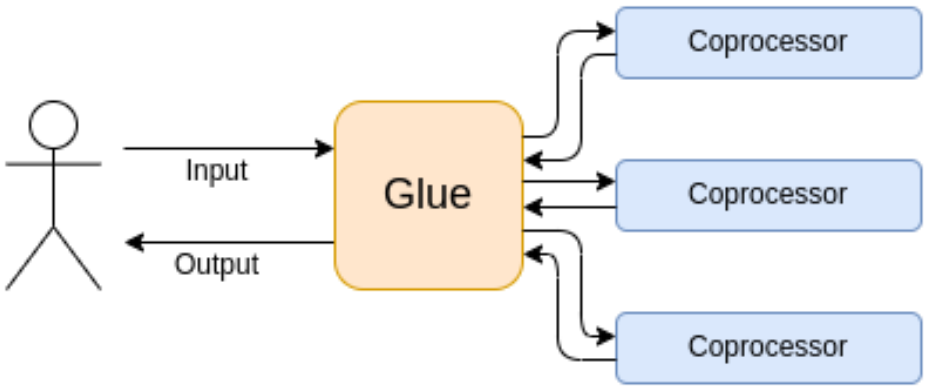

Figure 2: Illustration of glue and co-processor architecture for modern applications

3 High Level Overview

3.1 Chain Abstracted Packet

To enable the application-centric world, Socket Protocol centers around a structure called the "chain-abstracted packet," much like how general blockchains revolve around

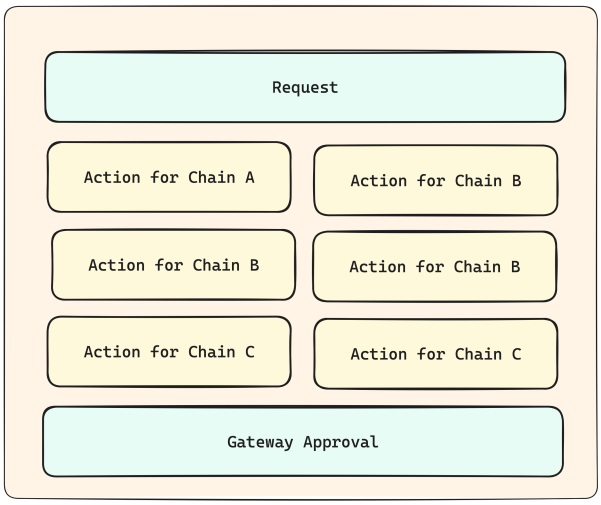

Figure 3: Chain Abstracted Packet

"blocks." This packet is simply an application-approved execution of requests and responses across networks.

To elaborate on the general architecture pattern for applications in a chain-abstracted and application-centric world:

request These are signed off-chain authorisations from the end user. Depending on what the application accepts, this could be userOps, intents, permit2 witness, or any custom type.

response These are signed responses by off-chain actors to help execute the pending requests on-chain. The combination of a request and response forms an on-chain actionable unit. The gateway is responsible for selecting the right response for the request in an application-specific manner.

gateway This is an application-specific lightweight business logic that lives off-chain but approves on-chain execution (packet) for the application contracts. It allows application developers to write arbitrary logic. The primary goal of the gateway is to create the packet by matching the right request and response, and to perform any pre-execution logic.

As mentioned earlier, Socket Protocol revolves around the chain-abstracted packet. Socket Protocol can be viewed as a framework that applications can leverage to create, execute, and finalize chain-abstracted packets (CAPs) across networks.

The supply chain can be roughly divided into three phases: packet creation, packet execution, and packet settlement.

3.1.1 Phase 1: Packet Creation

Applications are able to deploy a gateway with business logic where all user-requests and responses from off-chain actors are received as inputs. The app-gateway executes the logic to build a chain-abstracted-packet ready for onchain execution. As user-requests come in, smart off-chain agents compete to fulfil it as dictated by the app-gateway.

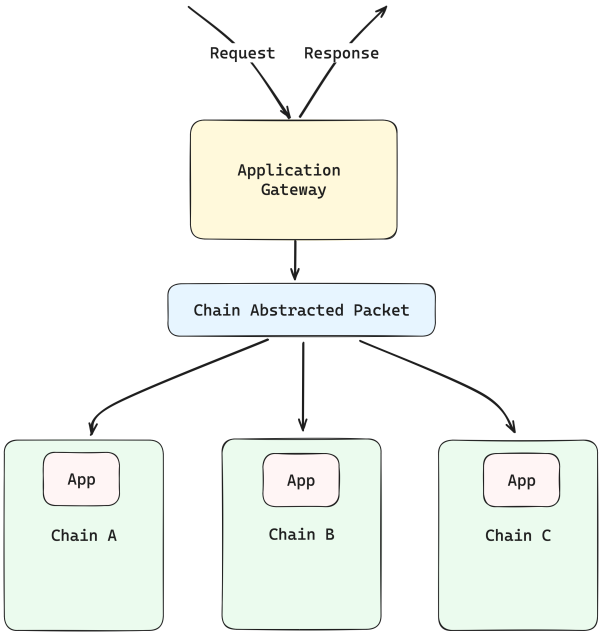

Figure 4: High Level Architecture of an Application in Chain Abstracted Future

3.1.2 Phase 2: Packet Execution

Once the Packet is created and authorised by the gateway, it's ready to be executed across networks. Off-chain actors will perform delivery to the networks needed and perform the execution of the Packet.

3.1.3 Phase 3: Packet Settlement/Finalisation

At the end, to finish processing, the application-gateway is informed of fulfilment.

4 Socket Protocol Design

4.1 Components and Agents

Socket Protocol is a framework with a set of immutable contracts and ability for application developers to leverage the framework with full customisability in essential areas.

Socket Protocol consists of 4 major components: Plugs, Switchboards, Gateway and of-course Socket which is the center-point and where everything is plugged into. The goal of the entire protocol is to create, execute and finalise chain-abstracted-packet.

Quick summary of what these components are:

Plugs Plugs are applications built on top of Socket Protocol aka smart-contracts that "plug" into Socket.

Gateway These again same as above are applications built on Socket with the difference being they are the entry/exit point of your onchain application plugs and are hosted by "watcher service operators" offchain.

Switchboard Switchboards are on-chain smart contracts that applications(plugs) can select to authenticate packets and their validity. Switchboards allow Plugs to elect

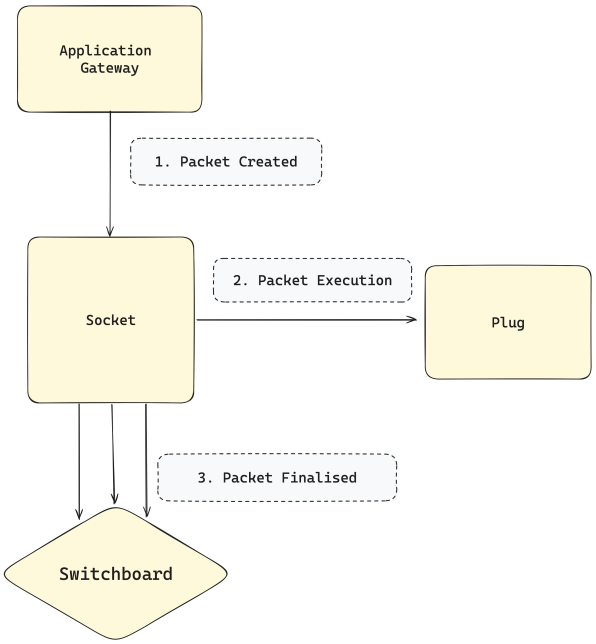

Figure 5: Socket Protocol Components

and choose multiple watcher-services operating in various settings, optimistic, msig based, zk based or rollup based.

Socket Socket is where everything gets plugged into, lightweight, immutable contract facilitates standardised interactions between the above components.

Socket Protocol operates with the help of 2 types of off-chain agents:

Watchers Watchers are service-operators that host application-gateways and "watch" chains. Watchers basically accept inputs from users and transmitters and run them via the gateway logic and finalise by signing off on the output called chain-abstracted-packet(CAP). To host a gateway you can deploy your own watcher-service or leverage existing ones. Watchers are employed by Plugs to host their Gateways.

Transmitters Transmitters are off-chain smart-actors that are responsible to execute user-requests onchain, anyone can become a transmitter and submit responses to user-requests. App-Gateway is able to select the right transmitters for the user-request ultimately.

4.2 How it works

Figure above illustrates the steps involved in creation, execution and finalisation of Chain-Abstracted-Packet across 2 networks, there can be arbitrary number of networks involved depending on application/user-request. This section walks through an example of an application on 2 networks composing via the application-defined-gateway leveraging Socket Protocol.

1. The user sends a request to the application-gateway with a signature over the required contents as per the application.

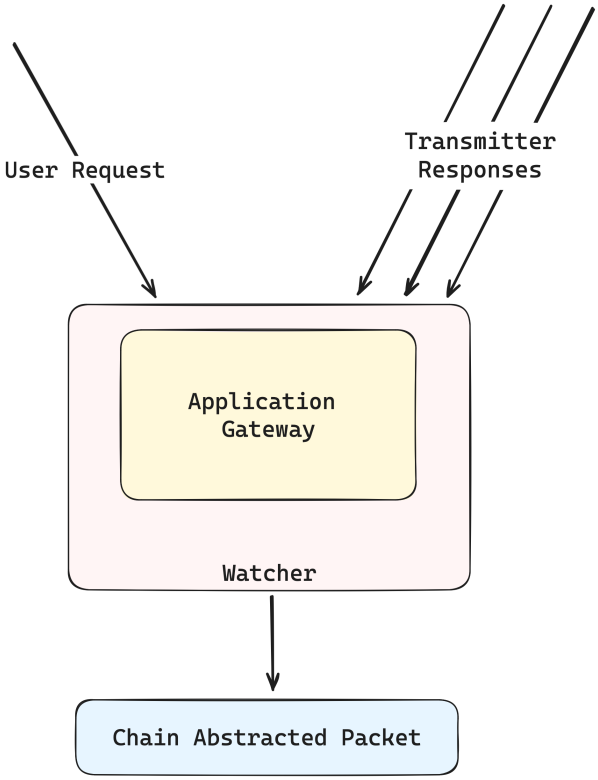

Figure 6: Socket Protocol Agents

2. Transmitters send responses to the gateway, resolving the request into multiple actions that can be performed on multiple networks onchain.

3. Gateway is triggered and application logic selects the winning transmitter that then delivers the chain-abstracted packet across chains. Packet contains the request, his response as well as watcher-signature that can be used to prove to switchboards that the watcher authorised execution.

4. As Socket contracts execute the plug onchain, authentication of the packet happens via the Plug selected switchboard, if the switchboard authorises the packet, the user-request and transmitter-actions meant for this network get executed.

5. Transmitter finalises execution on the source network by reporting to the gateway, which then authorises next leg with a fresh watcher-signature.

6. Same as 4, Packet is validated via switchboards and then transmitter-responses for this network get executed.

7. Finalisation of this leg happens as step 5 above.

8. Repeat of steps 5,6, 7 till the entire packet is executed.

4.3 Implications of Design

- **Incentivised Expansion:** Today, applications need to pay intermediaries aka service providers to expand to new networks, however due to the marketplace structure of Socket Protocol, transmitters compete with other transmitters to win the right

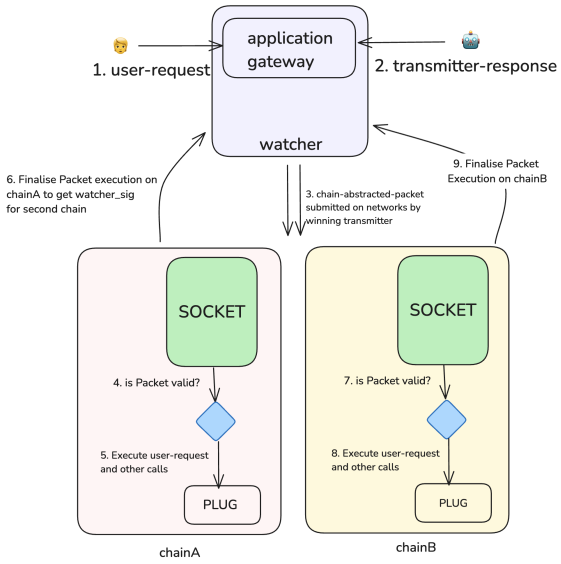

Figure 7: Step by Step Flow

of execution, transmitters will be open to serving new chain faster as it gives them edge over other transmitters and increases their likelihood of winning user-requests towards new networks. So while the Socket Protocol can already be permissionlessly deployed to new networks, the market dynamics enable "incentivised expansion" which flips the current meta.

- **Application leakage to intermediaries gets re-captured:** Applications leak all sorts of values due ttheir design to intermediaries like validators, sequencers, arb bots and relayers exposing users to worse execution and other negative externalities, Socket Protocol allows applications help users get better outcomes and protect them from these negative externalities.

- **Gasless and batched execution by default:** Due to the nature of the protocol, transmitters are incentivised to automatically batch user execution onchain as much as possible making it cheaper for users to interact onchain while also providing gasless execution, no failing transactions as a side effect.

- **Security Pre-Checks:** Applications while operating in 1000 chain world often want to maintain global invariants that should not break no matter what, for eg, token-supply for an erc20 is a great one, applications now have a natural place to write such checks and increase safety properties of their application for their end users.

- **Modular Settlement/Security:** Given applications/plugs can choose any switchboard AND registering new switchboards is permissionless, applications have full control on how they want to validate actions across networks as well as interactions between gateway and the onchain application.

5 Concepts

5.1 Application Gateway

Application Gateways are application-specific top level functions that can run pre-onchain execution of the application itself. Socket doesn't enforce a particular VM or language for these gateway contracts, but assuming EVM for simplicity, developers can leverage the gateway for various use cases.

Gateways are hosted offchain by watcher entities, which is a permissionless role. Applications can select and employ watchers via onchain contracts called switchboards. This gives them the flexibility to select multiple watchers and decide how they want to validate watcher execution of their gateway contracts.

There are various use cases for gateway contracts. Developers might use the gateway contract to:

- Run simulations of all interactions with their onchain applications to increase safety
- Run an auction to optimize better outcomes for their end users
- Enable global routing

5.2 MOFA

MOFA stands for Modular OrderFlow Auctions and is a unique concept Socket Protocol leverages to enable developers to create a market for the their orderflow where third-parties compete to fulfill the orderflow. It gives application developers the ability to convert an unaligned intermediary into an aligned friend that instead of being a hurdle instead then becomes the enabler of chain-abstraction for the application.

Applications can now optimise the properties they and their users care about, like price, latency or any other. Third-parties to fulfill these user-requests onchain as defined by applications enabling chain-abstraction while delivering value to users and applications.

5.3 Switchboards

Switchboards are essentially onchain verifier contracts that anyone can write and attach to Socket Protocol. Before executing the application, Socket Protocol checks with the application-selected switchboard, allowing applications to perform various checks before executing their onchain contracts.

You can think of switchboards as libraries that anyone can use. For example:

- A switchboard that allows for plug execution if only a single watcher authorizes execution
- A switchboard that employs 100 watchers and allows for execution if 2/3 authorize it
- Optimistic, ZK, or oracle-based switchboards that prove the watcher ran the application-defined gateway as intended

Applications will choose different switchboards according to their use case. Different switchboards will offer varying levels of cost, security, and latency for onchain execution. Of course, applications can switch Switchboards as needed.

5.4 Watchers

Watchers are entities that run the "watcher service," which read multiple chains and allows application developers to deploy gateway contracts on top of the VMs they run. Applications employ watchers via switchboards, making them an important part of the puzzle.

Key points about watchers:

- Depending on the switchboard application developers use, they would have varying degrees of trust in this entity
- Anyone can run a watcher-service and listen to as many or as few chains as they want to participate in the Socket protocol
- Participation is subject to applications opting to deploy their gateway contracts on the watcher's service

6 Conclusion

The application-centric approach presents a promising solution to the challenges posed by blockchain fragmentation. By empowering applications to become the primary gateway for users, we can create a more unified and user-friendly blockchain ecosystem, fostering greater adoption and innovation in the space.