Enso: The intent engine

Connor Howe, Peter Phillips, Milos Costantini

July 2024

Version 0.1

Abstract. We introduce Enso, a decentralized shared network for generating executable bytecode for smart contracts across many blockchains, rollups, and appchains. Enso is a network that lives as a tendermint layer-1 based blockchain, mapping all smart contract interactions allowing developers to combine and interact with any smart contract on any chain through one source. The Enso network addresses the key issue of usability within the blockchain ecosystem by enabling developers to state their desired outcome, and network participants coordinate to obtain a solution to fulfil the developer's request. By using Enso as a coordination layer developers can now build truly composable, and unstoppable applications with minimal friction.

1 The Problem: Usability

The world of blockchain is continuously changing. What was once a simple shared network state known as Bitcoin, has evolved into a complex ecosystem with thousands of blockchains and hundreds of thousands of smart contracts. As the space continues to mature these numbers will continue to grow, only adding to the complexity and fragmentation of the industry. This complexity and fragmentation poses challenges for developers looking to integrate smart contracts. These challenges create friction in building innovative applications and new primitives, reducing the ability to scale for mass adoption.

At Enso we view the largest problem in the current blockchain ecosystem to be **usability,** created by fragmentation of these blockchains and smart contracts. We believe composability is one of the most important value propositions of blockchain technology. Therefore, allowing developers to interact and compose smart contracts together in a simple, and convenient manner is imperative to grow the blockchain ecosystem.

The Enso network introduced within this paper addresses the key problem of **usability.** Developers before using Enso were required to manually write smart contracts to interact with other smart contracts, understand each smart contract's nuances and build customized infrastructure to maintain their integration. Now with Enso, developers can interact with one source that abstracts all complexities of interacting with smart contracts. This is achieved through a **shared network state** that stores and builds all smart contract interactions across blockchains, rollups and rollapps to be consumed for execution. Below we give a brief insight into the evolution of the blockchain ecosystem, presenting the continued challenges for developers to build applications.

Bitcoin created the first shared network state, better known as a blockchain. Bitcoin was created as a peer-to-peer electronic cash without financial institutions needing to facilitate transactions. Instead, independent nodes validated state transitions from the sender and receiver. All state lived on one chain, and Bitcoin was primarily focused on payments and not decentralized applications, the role of a developer in integrating Bitcoin was trivial in comparison to today.

Ethereum expanded on the shared network state concept developed by Bitcoin, by enabling the creation of smart contracts fueling the development of decentralized applications (DApps). Smart contracts are code that lives fully on the Ethereum Virtual Machine (EVM), and state transitions can be any arbitrary interaction. This led to new application categories being built such as decentralized finance (DeFi), decentralized physical infrastructure networks (DePin), non-fungible tokens (NFTs), data curation marketplaces, and much more. Each of these application categories could communicate and interact with other smart contracts that live on the Ethereum blockchain due to the shared network state. New primitives that intertwined, and composed many smart contract interactions were now being created.

Layer-1 blockchains such as Solana, Neo, Near, Avalanche, and many more were created with variations on technical implementations such as optimization, consensus and programming languages whilst continuing a shared network state. Enabling developers to build applications that interact with other smart contracts on a different framework.

Layer-2 blockchains addressed key issues in the transactional cost of users interacting with the desired blockchain, as with adoption continuing to grow the gas cost continues to rise with more requests being made to the network for including transactions in the next block. Projects like Conduit, Gelato and AltLayer developed software known as RaaS (Rollups-as-a-Service) that enabled application developers to deploy a blockchain within 5 minutes. Accelerating the amount of blockchains being deployed, and indirectly creating further usability issues as fragmentation among smart contract composability starts to increase as developers cannot easily compose smart contracts together due to the shared network state being split among many blockchains.

AppChains also known as an application-specific blockchain creates an isolated chain only for that particular application, and they wish to customize consensus, gas costs, validator set, and any other features they may require for their application's purpose. Developers wishing to interface with these applications, now have an incredibly cumbersome task of integrating a new blockchain whilst obtaining only one new application.

Evolutions throughout the years have created an alarming precedence for **usability**. Developers are now required to choose which chain they wish to integrate, and what smart contracts to interface with. Once a developer has chosen their desired blockchain, they are faced with an incredibly large task for understanding the inner workings of each smart contract, and how to compose these with other smart contracts. To interact with a smart contract, prior state may be

required, thus the developer will not only need to build one smart contract adapter but many smart contract adapters for all the required state transitions that must be obtained, regardless of the developers being aware of the desired outcome they wished to obtain. Resulting in tremendous developer resource allocation, indirectly creating vendor lock-in, and developers not being able to quickly, and safely integrate the smart contract features they want without a full code rewrite. As a consequence of these complexities, innovative products, and thus real-world adoption is being stifled.

At Enso we believe developers should easily interface with any smart contract on any chain in a safe, simple, and convenient manner to build the products they desire without the extreme burden of manually integrating everything themselves. The Enso network addresses this key issue, whereby developers can interface with one network to generate transactable data across many smart contracts on many chains, taking into account all prior state that is required to interface with their desired smart contract. Developers can now define their desired outcome and offset all of the technical implementation details to the Enso network participants. The network fosters a collaborative environment where developers contribute to the network on how to execute a particular smart contract, creating a shared network state of all smart contract interactions, and the relationship between them.

2 Enso: The shared network state

For developers to interface with blockchain conveniently, they should not be concerned with all the complexities of each smart contract and should be able to state their desired outcome as an intent with all complexities of execution abstracted away. To achieve this, Enso introduces the **first shared network** for building transactable bytecode for execution. Smart contract abstractions are contributed to the shared network by **action providers**, and once accepted as a valid contribution they are stored as an entity in the map. With each contribution, the relationship between smart contracts and their ability to interface with each other becomes clearer. Resulting in an extensive map of all executable relationships.

To build transactable bytecode for execution on smart contracts, the network not only stores the individual data required to build bytecode such as the function signature, token in, and outcome, but also pre-requirement fetchers that must be validated before building bytecode. For example, an NFT mint may have a limit of 1,000 NFTs available for purchase, so a pre-requirement fetcher for this particular abstraction would fetch the current minted amount and the maximum amount of NFTs that can be minted. Consequently, consumers are returned valid bytecode that can be executed.

The Enso network has four different participants:

- **Action providers** are developers that publish smart contract abstractions onchain, and earn a share of consumption fees in exchange. The network rewards action providers in proportion to the solutions generated that consume their contributions.
- **Graphers** are sophisticated individuals focused on building algorithms to solve a consumer's request. These complex algorithms traverse the Enso network for all abstractions contributed, and combine them in an optimal order to produce executable bytecode. Graphers are incentivized to continuously find the most optimal solution, as only one solution can be chosen for fulfilling the consumer's request.
- **Validators** secure the Enso network by accepting valid consumer requests, authenticating abstractions contributed, and determining the winning solution provided by the Graphers.
- **Consumers** call the Enso network with their request by stating their desired outcome and optionally pay fees.

There are three key components within the Enso network:

- **Requesting** whereby a consumer specifies their desired outcome as an intent to obtain a bytecode solution in response.
- **Action submission** determines how many solutions can be provided by the network.
- **Solution aggregation** establishes the most optimal solution proposed by the Graphers.

3 Enso: Requesting

The Enso network is built on the core premise of consumers requesting an outcome rather than explicitly stating how to derive the outcome. This paradigm shift in usability is defined as an **intent**. Intents enable consumers to state “what” they would like to obtain, not like a transaction where an expression is given on “how” an interaction should be done. Creating a transaction requires a deep understanding of particular smart contract caveats and nuances. Whereas, intents remove these complexities and allow consumers to outsource the solution creation for their desired outcome. Intents outline a set of variables that must be achieved, and the constraints around these variables, resulting in a tremendous developer experience.

Once a request has been made to the Enso network, network participants coordinate to determine the solution, and a solution is returned in the following format:

```
Unset
RequestResponse {
       IntentId string
       From string
       To string
```

```
        Data string
        Value string
        AmountOut string
        Gas string
}
```

4 Enso: Abstraction

To ensure that the Enso network has the most extensive map of smart contract interactions, a contribution mechanism with incentivization has been developed. Whereby developers (action providers) can contribute abstractions to the network, and obtain rewards if their abstraction is consumed when generating a solution for a customer's request. Creating an economy for developers to contribute abstractions they envision the ecosystem will utilize, and the more verbose an abstraction is the less likely it has already been contributed. To prevent squatting of abstractions, the network allows multiple abstractions with the same purpose to be contributed, and the abstraction that has the most amount of Enso token staked will be chosen within solution creation.

Graphers consume abstractions on the network to build solutions. To ensure scalability of consumption, each abstraction must have an associated action type. An abstraction can only be associated with one action type, and an action type may have many abstractions. An action type is a wrapper that defines a particular behaviour that the associated abstractions are doing. For example, a DeFi smart contract that requires a token or many tokens to be transferred whilst obtaining a token or many tokens in return is categorized as a deposit. When contributing an abstraction, it is first defined as a message action type:

```
Unset
MsgAddDeposit {
        Protocol string
        TokensIn []string
        PositionOut string
        Primary string
        SupportsReceiver bool
}
```

Once the message has been added, then it is parsed into the following action type:

```
Unset
Action {
       ActionId string
       ActionType string
       Protocol string
       PositionsIn []string
       PositionsOut []string
       SupportsReceiver bool
       Provider string
       Args map[]string
}
```

As there are many dynamic outcomes when interacting with smart contracts, the *positions* variable encompasses many different outcome types, i.e. if the execution returns token(s), NFT(s), state transition(s), and such. Particular smart contracts enable outcomes to be delegated to another receiver address, so SupportsReceiver has been implemented. By following this methodology, we open up the types of *consumers* such as artificial intelligence execution bots built on top of Enso with particular execution boundaries to be requested and then settled with the provided permissions.

New *PositionTypes* can be added to the network through the governance module. A position is generated as a deterministic hash from the following:

```
Unset
Position {
       PositionType string
       Chain uint64
       Address string
       TokenId string
}
```

To validate solutions generated by the network, each associated abstraction has a *ViewType* attached, where on-chain getters can be embedded to cross-reference the desired outcome vs the actuality:

```
Unset
ViewFunction {
       PositionType string
       Fragment string
       Target string
```

```
        Args []string
        ResponseIndex uint64
}
```

When an abstraction has been contributed with an associated action type, the proposal of the new abstraction will enter the following phases to ensure validity on the network:

- **Step 1:** Action provider submits action
- **Step 2:** Validators listen for new action submissions, and validate the action provider has registered within the on-chain providers list
- **Step 3:** Action msg data will be parsed into Action type format, and convert other values into the Position type format
- **Step 4:** Interface with the Action Provider indexing infrastructure, and receive the action definition
- **Step 5:** Generate transaction bytecode from the action definition
- **Step 6:** Simulate the transaction bytecode, and run to ensure the outcome in particular positions is correctly behaving
- **Step 7:** Validate positions are now stored on-chain and modify position data if any data is missing
- **Step 8:** Store the action on-chain
- **Step 9:** Emit the ActionId

Once these steps are completed, the abstraction is now part of the Enso network map and can be used for generating solutions.

5 Solution aggregation

When a request has been made to the Enso network, network participants will coordinate and compete in solution proposals to provide the best result for the consumer. Competitiveness occurs between Graphers providing solutions, ensuring the Enso network continuously provides the best possible solution rather than the network becoming stagnant over time. Resulting in consumers continuously using Enso as their one source of executable data rather than going to another source.

Validators listen to requests being made to the network confirming the correct format is used, and if correctly submitted the request will be propagated into the network's mempool. Whereby the Graphers run their proprietary algorithms to find the most optimal path across all actions in the network. The underlying algorithms used can be incredibly complex, or as simple as K shortest path.
As long as the solution generated can be formed as bytecode it is an acceptable proposal from the Graphers. Once a solution has been generated, the bytecode will be associated to an *IntentID* that it intends to solve. The Validator's role is now to simulate each solution, by forking

the desired settlement chain with all recent state and executing the bytecode to ensure that it can be executed, and confirming correct state transition by calling the *ViewFunction* helpers in each action. Validators may also run custom modules for balance checks, overrides, delegate calls and such.

Once the state transitions have been detected individually by each validator, each validator propagates their findings back into the mempool with the appropriate *IntentID* and *SolutionID.* Upon which the solution with the highest dollar output with the lowest $ value cost for execution is chosen as the winning solution. All other solution proposals are removed from the mempool. Resulting in no unnecessary bloat within the network itself as these solutions are not relevant for the future, this can be viewed as an inbuilt caching layer. The winning solution is then provided back to the *consumer* for their execution.

The Enso network operates on many different blockchains and solutions may have fees embedded into the bytecode used for execution. Instead of attempting to build a system that bridges all fees generated back to the Enso blockchain, an auction occurs on the Enso network where an address can bid an amount of Enso tokens to have the right to claim these fees on the respective blockchains. By following this methodology, Enso tokens that are used in acquiring these fees are distributed among the network participants based on their contributions such as abstraction consumption, winning solutions, validation and more. Thus, Enso has a built-in circular economy that is extensible and adaptable for the continued rise of blockchains deployed.

6 Applications

The architecture of Enso enables any application that wishes to interact with smart contracts on any blockchain, with the most optimal execution. As Enso is agnostic in the type of smart contracts that can be interacted with through the network, the variety of applications that can be built using Enso is infinite. The universal gateway to interact with all smart contracts on all blockchains enabling developers to build truly unstoppable, and composable applications.

You can build whatever, however, and wherever using Enso.

7 Token

Enso is the native token used within the Enso tendermint-based network. The Enso token is a critical component of the Enso ecosystem, and the network cannot function without the token. The token has key functionality:

- **Gas** enabling requests, and modification of state on the Enso network.
- **Governance** protocol upgrades are facilitated by token voting.

- **Network participation** of Validators, Graphers, and Action Providers requires staked Enso tokens to slash malpractice.
- **Delegation** of staking to network participants to further secure the network.

The total supply for the Enso token is 100,000,000.

8 Roadmap

The Enso network will have different phases of release. The first phase will be the initial network launch having independent validators simulate bytecode solutions, where a centralized Enso service will still co-exist with the network. During this phase, developers wishing to be action providers can contribute to a centralized service that Enso hosts to ensure the network is functioning correctly, and the process for contribution is seamless. The second phase enables a fully permissionless contribution environment for the action providers, and a wide variety of graphers can participate in the network. At this stage, the Enso network is a fully sustainable ecosystem and will expand from only Ethereum Virtual Machine (EVM) but also into Solana Virtual Machine (SVM), and Move Virtual Machine (MVM) frameworks to further enhance the developer experience for these ecosystems.

9 Conclusion

In this paper, the **first shared network state** enabling developers to interact with any smart contact on any blockchain by one source was introduced. This paradigm-shifting concept is the first of its kind and fixes blockchain usability created by continued fragmentation among the ecosystem. Each network participant was thoroughly detailed, and the process by which contributions are made, and validated within the network was made clear. Enso acts as an interface layer between all ecosystems creating a vibrant environment for developers to interact with blockchain technology, bringing further utility and usage of blockchain technology as a whole. You can build whatever, however, and wherever using Enso.