Omnite ONFT Protocol

Abstract

The Paper introduces the first offering of a new multi--chain solution aimed to revolutionize the way users create, mint, sell, swap and purchase non-fungible tokens, fundamentally altering the current NFT industry. This is achieved by utilizing multiple cross-chain message protocols and Omnite smart contracts. Today's NFT token creators are forced to either choose a specific blockchain platform on which to launch their collection or tightly couple it with a particular message broker for inter-chain operability. Omnite removes these limitations by introducing the capabilities of multichain NFT token collections, without having to rely on a centralized custodian and current slow, expensive and centralized token bridging mechanisms. This document outlines all of the disadvantages of adopting a single blockchain/bridge protocol from the perspectives of creators, investors, and marketplaces, and how they may profit from a multi-chain solution. It also describes how the Platform's approach can reduce excessive gas fees and difficulties with the process of trading NFTs while also allowing for complete decentralization of the implementation procedure.

1. Introduction

Omnite is a smart contract-based protocol that allows NFT tokens to be moved between blockchain networks in a decentralized manner without the need for a centralized custodian. It dismantles the boundaries that exist between blockchains, allowing NFTs to easily move across networks with the emergence of trustless omnichain interoperability protocols. The platform operates on all leading EVM networks as ETH, POLYGON, AVAX, FANTOM, BSC. It is intended to enable networks based on various ecosystems in future phases: Cosmos, Solana, Tezos as a result of the message brokers' development. The Omnite protocol addresses major issues that affect platforms/tools that transfer NFT tokens between networks: centralization, a convoluted token transfer procedure, excessive transaction fees, and a long wait time. The Omnite's procedure of transferring Tokens between networks is based solely on smart contracts and it doesn't require any additional steps from the user's point of view other than when transferring NFT tokens within the same network. Moreover, collections existing in the Omnite ecosystem are not strictly tied to any network nor inter-chain message broker. Omnite protocol works above these parts of the system and gives an abstraction layer and seamless operability so the users can enjoy a true omnichain experience. Since the beginning of 2022 and with a development of cross-chain message protocols, the NFT community has seen a rapid growth of different ONFT standard proposals and chaotic creation of tools meant to support them. Current problems related to this area are as follows:

- Lack of cross-chain token verification
- Various different ONFT standards
- Lack of support for multiple communication protocols
- Overwhelming development costs for creators

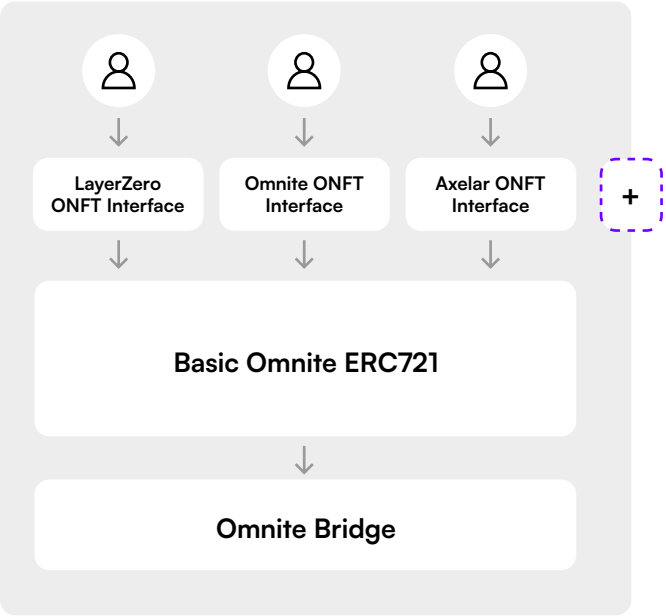
Omnite aims to fix these issues by building an ecosystem unifying different ONFT standards and protocols. This paper will highlight the most important parts of the solution.

2. Message Brokers

With a rapid growth of cross-chain communication protocols (Axelar, LayerZero, Chainlink, and more to come), the blockchain ecosystem is on the edge of becoming a new internet. Same as with the world wide web, there is a need for a common messaging interface. Omnite currently works with various inter-chain message brokers to give users a feeling of a seamless transition between the networks. By placing the abstraction layer on top of cross-chain protocols, Omnite will unify all of them and optimize costs, time and security for all actors in its ecosystem.

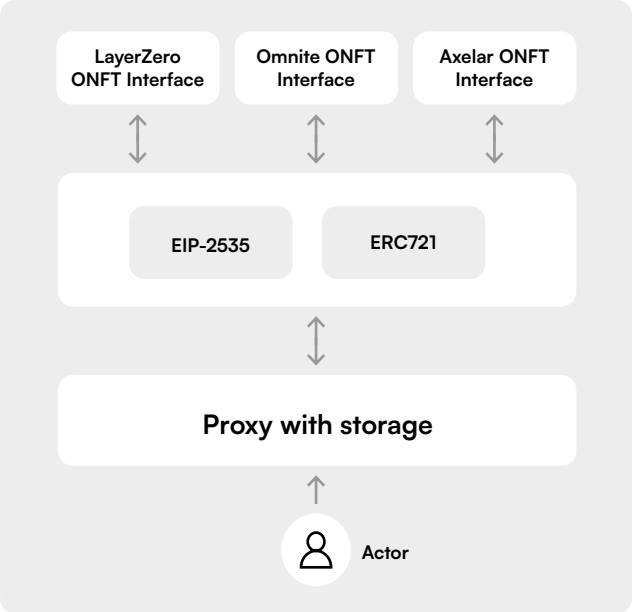
3. Protocol design

The protocol supports tokens created using the OMNITE launchpad (hereafter referred to as “native tokens”) as well as all other NFT tokens created in the ERC721 standard since the inception of the blockchain network. The idea behind the Protocol is to integrate all existing interfaces for Omnichain NFTS. To do so, they had to be included into Omnite’s ONFT implementation, which resulted in ERC721 codesize growth. Currently, most blockchains have limits on the size of a smart contract’s bytecode[1]. To downsize it, a hybrid of diamond and upgradable proxies[2] patterns have been introduced[3].



Users can choose which interface they are using.

The diamond pattern and upgradable proxies



Whenever a user or some smart contract calls the Proxy (which is the address of ONFT collection), it delegates the call to the diamond smart contract, which holds the basic ERC721 implementation, token storage and references to functions in the facets. The latter hold particular implementations needed to communicate with their cross-chain protocols. The diamond does another delegate call to the given facet, which is responsible for communication with its specific OmniteBridgeSender. The proxy[4], which is explicitly called by the user, is an actual contract that is deployed whenever a collection is created. It holds the storage of the token along with the diamond’s address, which is shared between other collections. It uses UpgradableBeacon[5], which keeps track of the newest diamond implementation. The diamond in itself is a blueprint, so it can be swapped whenever a new ERC721 implementation is needed. Besides fixing the bytecode size issue, other advantages of the given solution are:

- Possibility to extend Omnite Protocol with more cross-chain solutions
- Easy way to add new features to the smart contracts
- Modularity of the code, each interface is separate from the others
- Cheaper deployment for the users

1. https://eips.ethereum.org/EIPS/eip-170
2. https://docs.openzeppelin.com/contracts/3.x/api/proxy#UpgradeableProxy
3. https://eips.ethereum.org/EIPS/eip-2535
4. https://docs.openzeppelin.com/contracts/3.x/api/proxy#UpgradeableProxy
5. https://docs.openzeppelin.com/contracts/3.x/api/proxy#UpgradeableBeacon

Omnite ONFT interface

Any ERC721 created in the Omnite ecosystem supports the native ONFT interface which allows users to easily bridge the tokens between networks.

```
function moveTo(uint16 chainId, uint256
_tokenId, uint256 _gasAmount)
```

The moveTo function can be called either by the owner of the token or a sender with allowance. To execute it, the user provides numeric chainId in the Omnite ecosystem, NFT id and gas amount needed to execute bridging on the target network. Gas can be estimated by calling Omnite Bridge Sender.
Although this paper is focused around ERC721 (which is the most widely used standard), Omnite will also support ERC1155 tokens.

Upgrading code on multiple chains at once (Hot swaps)

All modern EVM applications use a Proxy pattern to be able to update smart contracts code in case of new features or security fixes. Omnite Hot Swaps introduces a way to upgrade multiple instances of the same contract at once in a secure-multichain way. Hot Swap consists of 3 components - Blueprints (which are representing various smart contract implementations (e.g. NFTs), UpgradeableBeacon which keeps track of the newest possible implementation for a particular contract type (e.g. the newest implementation for ERC721), and BeaconProxy which is visible to the user as an NFT contract and is responsible for delegating calls to real implementation. Our approach of using a proxy pattern reduces gas consumption of deploying a new ERC721 NFT collection by more than 75% and gives Omnite users the ability to receive new features implemented by the team and community in the future with no effort.

DDOS, bridge block protection and gas estimation

A problem that is impossible to solve on-chain is gas estimation on the different blockchain networks, as one blockchain network does not have access to the data of other networks. Message brokers require senders to pay for a particular amount of gas needed to process operation, thus one can never be sure that the gas amount provided by the frontend application is not malicious (or calculated wrongly) and will be sufficient to execute the transaction on the destination network correctly. Omnite sets a minimum gas that the bridge can accept and takes the approach that the transactions never revert with errors, producing error events instead. This was achieved by using sophisticated low-level calls on the very entry stage of message broker's callback in the bridge.

Extending Omnite with new networks and message brokers

All modern EVM applications use a Proxy pattern to be able to update smart contracts code in case of new features or security fixes. Omnite Hot Swaps introduces a way to upgrade multiple instances of the same contract at once in a secure-multichain way. Hot Swap consists of 3 components - Blueprints (which are representing various smart contract implementations (e.g. NFTs), UpgradeableBeacon which keeps track of the newest possible implementation for a particular contract type (e.g. the newest implementation for ERC721), and BeaconProxy which is visible to the user as an NFT contract and is responsible for delegating calls to real implementation. Our approach of using a proxy pattern reduces gas consumption of deploying a new ERC721 NFT collection by more than 75% and gives Omnite users the ability to receive new features implemented by the team and community in the future with no effort.

4. Protocol design

Besides ERC721 diamonds, proxies and facets, Omnite protocol consists of the following components:

Omnite Bridges

The heart of multi-chain communication. They have direct access to cross-chain brokers' endpoints and work as a message layer between Omnite and other protocols. They consist of Bridge Sender and a Bridge Receiver. The contracts are responsible for encoding and decoding low-level calls, executing them on different networks, and receiving messages on the other side. They are trustless by design, for instance, any sender can deploy multiple contracts to different networks in just one transaction. Each message broker has a corresponding sender and a receiver. They are called by ERC721 diamond's facets directly.

Contract Factory

Is a contract holding blueprints of the tokens. It is used to deploy new contract instances (e.g. Different ERC721 implementations) by utilizing diamond and proxy patterns with upgradable contracts and beacon proxies. It supports versioning of the blueprints and deployment by contract type, which allows saving gas: Omnite doesn't have to transfer the whole bytecode in order to bridge the contract, it only needs a blueprint name along with constructor params needed to instantiate a storage for the proxy contract.

Collection Registry

A problem that is impossible to solve on-chain is gas eHolds the records of all Omnite's ERC721 native and non-native tokens. Each ERC721
deployed or bridged by the service is stored in this registry. Each record holds the owner, address, and name of the collection. CollectionRegistry provides the data required to validate incoming and outgoing bridge requests. This smart contract also is a fundament for ONFT token verification, which can be used to check a **cross-chain history** for a given collection

Access Control List (ACL)

Specifies which contracts are granted access to other contracts, as well as what operations are allowed on given contracts. It is based on OpenZeppelin's AccessControl[1] library, but extended to support non-EVM networks' address structures.

System Context

Keeps track of smart contracts in the Omnite system, allowing contracts to subscribe for address changes using callbacks. For example, when a new version of the ContractFactory registry is set, the system context notifies all interested parties.

Native Tokens

Contracts deployed directly from Omnite Launchpad. Owner of the collection can easily mint them on the platform and bridge them afterwards. Each token's smart contract on particular networks has a minting range, which restricts minting to particular ids. Native tokens are deployed as upgradable contracts with diamond pattern to support different ONFT interfaces.

Non-native Tokens

Contracts which were deployed outside of the Omnite platform, i.e. Invisible Friends, Bored Apes, etc. They have to implement ERC721Metadata interface. To bridge those tokens between networks, they need to be wrapped first. There is no possibility to mint them through Omnite.

5. Omnite DAO

The Omnite governance framework will comprise of 2 approaches:

Community Voting

Where there are any new proposals, the community can vote for or against the proposals until consensus is reached.

1. https://docs.openzeppelin.com/contracts/4.x/api/access#AccessContro

Advisory Board

Consisting of Omnite employees as well as 3rd-party independent advisors with
technical, insurance, compliance and/or other required expertise. They will work as the
oversight committee to set certain rules, review the community proposals, as well as
executing a contingency plan when the community voting mechanism fails.
The fundamental voting mechanism is that the Omnite token shares held stands for the
voting rights with a cap per member set to avoid the concentration risk. The voting
outcome will be based on factors such as the quorum, majority, voting right weight etc.

6. Protocol Fees and payments

Omnite project due to its multi-platform nature is equipped with a payment mechanism using many native coins. Depending on which network the user calls the service (creates collections, mints tokens, or sends a token to another network), payment will be in the source network coin (native currency, stable coins and Omnite tokens). Fees are accumulated on wallet contracts deployed on all supported networks. These contracts act as a treasury where the capital is accumulated. The fees charged represent a profit for the project. Thanks to support of multiple message brokers, Omnite applications can estimate which protocol is the cheapest at a given time and decide accordingly.

6. Case Study: Bridge Non-native NFT

In this section, we briefly describe the details of how we implemented the basic feature of the Omnite platform, which is bridging existing collections and sending the user's token to the target blockchain.
Most of today's existing collections rely on a set of Openzeppelin's libraries and contracts, which follow the ERC721Metadata standard. This interface exposes basic fields and functions, such as name, symbol, and tokenURI. They are needed to properly manage the collection in the Omnite ecosystem. When a user holds a particular token for the collection which hasn't been yet brought to the Omnite's platform, first, it needs to be wrapped by Omnite's ERC721 wrapper, which follows the interface used by the platform's bridge (ITokenBridgeable). All bridgeable tokens on Omnite's platform have to implement this interface, therefore wrapping the source contract is crucial. This step is executed along with multichain deployment. The step consists of:

1. Deploying the wrapper on the source network
2. Deploying the Omnite's ERC721 contracts on target networks

Both operations are executed in one transaction. The caller needs to provide the original contract's address, chain IDs that identify particular networks in the Omnite ecosystem, and gasAmount for each deployment (gas amount is calculated in separate calls). The user also specifies which message broker he wants to use. He can either use native Omnite ONFT interface, or any other supported by the protocol.
The function being called is payable, since the gas cost for target networks is deducted from the value sent by the user. The call is delegated to the particular ERC721 facet responsible for calling its corresponding Bridge Sender. Omnite's bridge, in order to deploy contracts on target networks, packs the deployment message by encoding it to the Data struct with a Deploy operation type, contract type to be deployed, and collection ID, which is generated by encoding the block timestamp and caller's address. Next, it sends the message to the bridge's contracts on target blockchains. The message goes through the message broker's endpoint and is forwarded to the receiving Omnite bridge. The receiver unpacks the data and the deployment request is forwarded to ContractFactory. ContractFactory stores all the data needed for deployments and versioning of the contracts. Omnite utilizes beacon proxies: each new ERC721 is in fact a set of contracts following this pattern. They all share one diamont blueprint (via beacon) and facets and can be upgraded if needed. After deploying the BeaconProxy with

the UpgradableBeacon and constructor params needed to initialize the new collection with ERC721 fields, the new collection is registered in CollectionRegistry, which is responsible for storing all the collections which were deployed or wrapped by Omnite smart contracts.
After the deployment, the token is now ready to be bridged. When the approval transaction is executed, the wrapper is able to transfer the user's asset and the bridging begins. This transaction consists of:

1. Locking the token on the source network - it can be unlocked when the asset is bridged back. It is a much safer solution than burning the token, since the asset will always be unlocked. Token is locked on its smart contract and can be released whenever it is bridged back.
2. Calling the bridge to mint the token on the target network.

The move call is packed with the CallData operation type and packedData bytes. It includes the function selector for the minting, an ID of the token to be minted, and a tokenURI for this id. There is also the gas amount value and message value, similar to the deployment transaction. After receiving the message on the target network, the bridge unpacks the data and executes the minting function on the target ERC721. The address of the token is retrieved from CollectionRegistry. At the end of this whole process, there is an unlocked token on the source chain and a newly minted token on the target network. If the user decides to go back with his token, the asset is locked once again and an original one is unlocked. The process can be executed in perpetuity. If a particular target network holds the locked asset, there is no need to mint a new one.

8. Marketplace compatibility

Omnite smart contracts are designed to be fully compatible with leading NFT marketplaces. By following the metadata standard, each collection created on the Omnite platform has a metadata URI and an ownership management. After bridging the token to the target network, the user is still the owner of the NFT and can manage it with complete freedom. It can be listed on Opensesa[3], Rarible, etc. After selling the token, the new owner can bridge it back through the Omnite platform.

9. Conclusion

This paper highlights current problems and challenges related to the lack of common platform for omnichain NFTs and a role the Omnite Protocol can play in mitigating them. With a unifying standard of ONFTs and set of tools Omnite provides, boundaries between the networks will soon become a past and cross-chain asset moving consists of just a few clicks.