Checking CCIP Message Status

In this tutorial, you will learn how to verify the status of a Chainlink CCIP transaction offchain using JavaScript. Starting with a CCIP message ID, you'll execute the script to query the current status of a cross-chain message.

Before you begin

1. Initiate a CCIP transaction and note the CCIP message ID. You can obtain the CCIP message ID by running any of the previous CCIP tutorials.
2. Complete the prerequisites steps of theTransfer Tokens between EOAs tutorial.

Tutorial

This tutorial shows you on how to check the status of a Chainlink CCIP transaction using theget-status.js script. By supplying the script with the source, destination chains, and your CCIP message ID, you can verify the current status of your cross-chain message.

Execute the script in your command line:

nodesrc/get-status.js sourceChain destinationChain messageID The script requires the following parameters:

- sourceChainis the identifier for the source blockchain. For example,ethereumSepolia.
- destinationChainis the identifier for the destination blockchain. For example,polygonMumbai.
- messageIDis the unique identifier for the CCIP transaction message that you need to check.

Example Usage:

If you initiated a transaction from the Ethereum Sepolia testnet to the Avalanche Fuji testnet and received a message ID, you can check the status of this message with the following command:

```
$ node src/get-status.js ethereumSepolia polygonMumbai
0x4ea8080f2a51377247e93b5f45c00330b8c4fde3043e99847d0cab734f473df2
```

```
Status of message 0x4ea8080f2a51377247e93b5f45c00330b8c4fde3043e99847d0cab734f473df2 on offRamp
0xAC1456bafcc3403eb6e34f08FD2945cd7B3C8F0A is SUCCESS
```

Code Explanation

The JavaScriptget-status.jsis designed to check the status of a user-provided CCIP message. The contract code includes several code comments to clarify each step, but this section explains the key elements.

Imports

The script imports the required modules and data:

- Ethers.js:JavaScript library for interacting with the Ethereum Blockchain and its ecosystem.
- Router and OffRamp Contract ABIs: These Application Binary Interfaces (ABIs) enable the script to interact with specific smart contracts on the blockchain.
- Configuration Functions: IncludesgetProviderRpcUrlfor retrieving the RPC URL of a blockchain,getRouterConfigfor accessing the router smart contract's configuration, andgetMessageStatusfor translating numeric status codes into readable strings.

Understanding thegetMessageStatusfunction

Before diving into the script execution, it's crucial to understand how thegetMessageStatus function works. This function is designed to translate the numeric status codes returned by Solidity enums into human-readable statuses so they are clear to developers and users. The function uses a mapping defined inmessageState.json , which correlates to theMessageExecutionState enum used by theChainlink CCIP's OffRamp contract.

Handling Arguments

ThehandleArgumentsfunction ensures the script operates with the correct parameters. It validates the presence of three command-line arguments – the source chain identifier, the destination chain identifier, and the message ID.

Main Function: getStatus

The script's core is encapsulated in thegetStatusasynchronous function. This function completes initialization, configuration retrieval, and contract instantiation.

Initialization

Firstly, it establishes connections to the source and destination blockchain networks using theJsonRpcProvider.

Configuration Retrieval

The script then retrieves the configuration for router contracts on both the source and destination chains. This includes the router addresses and chain selectors.

Contract Instantiation

The script instantiates the source and destination router contracts using ethers and the router contract addresses.

Status Query

To query the status of the provided CCIP message ID, the script completes the following steps:

1. Check if the source chain's router supports the destination chain
2. Fetch OffRamp contracts associated with the destination router
3. Filter these contracts to find those that match the source chain
4. Query each matching OffRamp contract for an event related to the message ID

If an event is found, the script reads the status from the arguments. It translates the numeric status into a human-readable status and logs this information.