Cross-Chain Price Feed Oracle Project Report

I. Project Overview

This project implements a decentralized cross-chain price feed oracle utilizing the Reactive Network. Its purpose is to mirror official Chainlink Price Feeds from an origin EVM chain (Ethereum Sepolia Testnet) to a destination chain (also Ethereum Sepolia Testnet). The destination chain exposes a read interface compatible with AggregatorV3Interface, enabling seamless consumption of price data by downstream decentralized applications.

II. Design

The system comprises two primary smart contracts and leverages the Reactive Network's event-driven architecture for cross-chain communication.

1. ChainlinkReactiveBridge (Reactive Network):

- **Deployment:** Deployed on the Reactive Lasna Testnet.
- **Functionality:** This contract serves as the active monitoring component. It is configured to subscribe to `AnswerUpdated` events emitted by specific `Chainlink AggregatorV3Interface` contracts on the Ethereum Sepolia Testnet (the origin chain).
- **Event Processing:** Upon detecting an `AnswerUpdated` event, the `ChainlinkReactiveBridge`'s `react()` function is automatically triggered within its isolated ReactVM.
- **Callback Initiation:** The `react()` function orchestrates two distinct callbacks:
    - An external callback to update the corresponding `ChainlinkFeedProxy` contract on the destination chain (Ethereum Sepolia).
    - A self-callback to update its own internal state within the `ChainlinkReactiveBridge` contract on Reactive Lasna, tracking the latest processed price, timestamp, and update count for each monitored aggregator. This self-callback ensures the bridge's internal state reflects the most recent relay activity.

2. ChainlinkFeedProxy (Destination Chain):

- **Deployment:** Deployed on the Ethereum Sepolia Testnet.
- **Functionality:** This contract acts as the local price data repository for a specific Chainlink feed. It implements the `AggregatorV3Interface` and

`IChainlinkFeedProxyExtended` to provide standard and extended price querying capabilities.

- **Callback Reception:** It receives and processes price updates via callbacks initiated by the `ChainlinkReactiveBridge`. The `updatePrice()` function is the designated entry point for these updates.
- **Data Storage:** Stores the latest `roundId`, `answer`, `startedAt`, `updatedAt`, `answeredInRound`, `decimals`, and `description`. It also maintains a configurable history of past price rounds, pruning older data to manage storage.

3. Cross-Chain Workflow:

- A Chainlink Aggregator on Ethereum Sepolia emits an `AnswerUpdated` event.
- The `ChainlinkReactiveBridge` on Reactive Lasna, due to its subscription, detects this event.
- The `react()` function of the `ChainlinkReactiveBridge` executes, extracting the price data.
- The `ChainlinkReactiveBridge` then emits a Callback event targeting the relevant `ChainlinkFeedProxy` on Ethereum Sepolia.
- The Reactive Network's infrastructure relays this Callback as a transaction to the `ChainlinkFeedProxy`.
- The `ChainlinkFeedProxy`'s `updatePrice()` function processes the incoming data, updates its state, and emits `AnswerUpdated` and `NewRound` events, making the new price available to dApps.
- Concurrently, the `ChainlinkReactiveBridge` also initiates a self-callback to update its own internal monitoring state.

III. Threat Model

1. Unauthorized Price Updates:

- **Threat:** Malicious actors attempting to inject false price data into the `ChainlinkFeedProxy`.
- **Mitigation:** The `ChainlinkFeedProxy`'s `updatePrice()` function is protected by `authorizedSenderOnly` and `rvmIdOnly` modifiers. These ensure that only the authorized Callback Proxy (acting on behalf of the `ChainlinkReactiveBridge`'s specific RVM ID) can invoke the update function. This tightly couples the update mechanism to the trusted Reactive Contract.

2. Stale Data Propagation:

- **Threat:** The system relaying or storing outdated price information.

- **Mitigation:**
    - **ChainlinkFeedProxy:** The `updatePrice()` function includes a strict check `require(_roundId > latestRound.roundId, "Stale round")` to prevent older `roundId`'s from overwriting newer data.
    - **ChainlinkReactiveBridge:** The `react()` function validates `updatedAt > lastTs` (where `lastTs` is `lastUpdateTimestampForAggregator`) to ensure only newer Chainlink updates are processed and relayed, preventing the bridge from reacting to or propagating stale events.

3. Origin Chain Data Integrity:

- **Threat:** The canonical Chainlink Aggregator on the origin chain providing incorrect or manipulated data.
- **Mitigation:** This project assumes the inherent security and trustworthiness of Chainlink's decentralized oracle networks. The bridge accurately mirrors the data provided by the configured Chainlink Aggregators; it does not introduce new validation logic for the source data itself.

4. Reactive Network Liveness/Security:

- **Threat:** The Reactive Network suffering from downtime, censorship, or security vulnerabilities, impacting event detection or callback relay.
- **Mitigation:** The project relies on the operational integrity of the Reactive Network. The Reactive Network's design, including parallelized EVM and inversion-of-control, aims for efficient and robust operation. Any underlying issues in the Reactive Network would directly affect the oracle's functionality.

5. Gas and Funding Management:

- **Threat:** Insufficient REACT tokens on the `ChainlinkReactiveBridge` or native tokens (ETH) on the `ChainlinkFeedProxy` leading to failed transactions and service interruption.
- **Mitigation:** Both contracts inherit from `AbstractPayer`, providing `pay()` and `coverDebt()` functions. The `ChainlinkReactiveBridge` is funded with REACT tokens during deployment, and `ChainlinkFeedProxy` with ETH. Continuous monitoring and automated top-up mechanisms are essential for production readiness.

6. Callback Integrity:

- **Threat:** Manipulation of the callback payload during cross-chain relay.
- **Mitigation:** The Reactive Network's underlying infrastructure is designed to ensure the integrity of the relayed message. Furthermore, the `ChainlinkFeedProxy`'s `rvmIdOnly` modifier verifies that the callback originates from the expected `ChainlinkReactiveBridge`'s RVM ID, adding an extra layer of authentication.

IV. Trade-offs

1. Decentralization vs. Efficiency:

- **Trade-off:** The Reactive Network offers a highly efficient and potentially faster cross-chain relay mechanism compared to generic message passing protocols that might involve more relayers and higher latency. However, this introduces a direct dependency on the Reactive Network's specific architecture and operational model, which, while designed for decentralization, is a distinct ecosystem.
- **Choice:** Prioritizes efficiency and specialized event-driven processing for real-time price synchronization.

2. Real-time Updates vs. Operational Cost:

- **Trade-off:** Subscribing to on-chain `AnswerUpdated` events from Chainlink provides near real-time price updates. Each detected event and subsequent callback incurs transaction costs (in REACT on Reactive Lasna and ETH on Sepolia).
- **Choice:** Emphasizes real-time data availability, accepting the associated transaction costs for each update.

3. Historical Data Storage vs. Contract Size/Gas:

- **Trade-off:** The `ChainlinkFeedProxy` stores a configurable `maxHistory` of past price rounds. Storing more history increases contract storage size and the gas cost associated with updates (due to pruning older entries).
- **Choice:** Provides flexibility with `maxHistory`, allowing implementers to balance the need for historical data access with operational gas costs and contract deployment size.

4. System Complexity:

- **Trade-off:** Building a cross-chain oracle inherently adds complexity compared to a single-chain solution, requiring careful coordination across multiple blockchain environments and specialized protocols.
- **Choice:** The Reactive Network's abstract contracts (`AbstractReactive`, `AbstractCallback`, `AbstractPayer`) aim to simplify this complexity by providing reusable building blocks and standardized interfaces for event handling, callback authorization, and payment management.

V. Deployment Workflow and Transaction Hashes

The following steps detail the deployment process on Ethereum Sepolia Testnet and Reactive Lasna Testnet. The deployer account used is

`0x63eea403e3075D9e6b5eA18c28021e6FfdD04a67`.

Step 1: Deploy ChainlinkFeedProxy Contracts on Ethereum Sepolia

This step deploys three instances of the `ChainlinkFeedProxy` contract, one for each monitored Chainlink price feed (ETH/USD, BTC/USD, LINK/USD), to the Ethereum Sepolia Testnet. Each proxy is initialized with a `maxHistory` of 100 rounds and funded with 0.005 ETH.

- **Command:** `npm run deploy:proxy`
- **Deployer Account Balance (before):** 0.314808304045330297 ETH

Deployment: ChainlinkFeedProxy for ETH/USD

- **Deployed To:** `0x120a855499cC8e777cC1D06078Aff78d78bAa8f2`
- **Transaction Hash:** `0x54a0338718008727754994f2c0a1e66373aa45112d3283bb4e8b2678680770d6`
- **Etherscan Verification:** https://sepolia.etherscan.io/address/0x120a855499cC8e777cC1D06078Aff78d78bAa8f2#code

Deployment: ChainlinkFeedProxy for BTC/USD

- **Deployed To:** `0x9a1b222D6AC1467b4302CEE2fd28074B0bD8367f`
- **Transaction Hash:** `0xdbe1333523bfeeb09e7248f54d7e922bfc2f332bdea19b1d9c75d0166fa43612`
- **Etherscan Verification:** https://sepolia.etherscan.io/address/0x9a1b222D6AC1467b4302CEE2fd28074B0bD8367f#code

Deployment: ChainlinkFeedProxy for LINK/USD

- **Deployed To:** `0xDBF1CdB10956ba2474B12042f78DB1A61061032F`
- **Transaction Hash:** `0xfda2c751387567e06975c246828b55f92d6f93dd5cf79990c4bdf686eb73be19`
- **Etherscan Verification:** https://sepolia.etherscan.io/address/0xDBF1CdB10956ba2474B12042f78DB1A61061032F#code

Step 2: Deploy ChainlinkReactiveBridge Contract on Reactive Lasna

This step deploys the `ChainlinkReactiveBridge` contract to the Reactive Lasna Testnet. It is configured to monitor the three Chainlink aggregators deployed in Step 1 and target their respective `ChainlinkFeedProxy` contracts. The bridge is funded with 0.1 REACT.

- **Command:** `npm run deploy:bridge`
- **Deployer Account Balance (before):** 3.326816624 REACT

Deployment: ChainlinkReactiveBridge

- **Deployed To:** `0x7603470ae0116957ce2bC85929f63319dc07c7ef`

- **Transaction Hash:** `0x81d1d220b147745375d43ceafb051bfb3143fe892bd2feecb3e66434ab57aec1`
- **Reactive Sourcify Verification:** Verification failed due to chain ID not being recognized by the plugin. Manual verification or custom network configuration would be required.
- **Reactscan RVM ID:** https://lasna.reactscan.net/rvm/0x63eea403e3075D9e6b5eA18c28021e6FfdD04a67

Step 3: Initial Monitoring (Post-Deployment State)

After both sets of contracts are deployed, an initial monitoring run shows the current state of the system.

- **Command:** `npx hardhat run scripts/monitorAllFeeds.js --network reactiveLasna`
- **Timestamp of Monitor Run:** 2025-11-29T18:10:27.261Z

Status for ETH/USD, BTC/USD, LINK/USD Feeds:

- **Chainlink Source:** Shows current data from the canonical Chainlink aggregators on Sepolia. Example, ETH/USD Round ID: 28470, Price: 2979.971985 USD, Updated At: 2025-11-29T17:35:12.000Z.
- **Our FeedProxy:** Displays ⏳ FeedProxy Status: execution reverted: "No data present". This is expected immediately after deployment as the `ChainlinkReactiveBridge` has not yet processed any `AnswerUpdated` events from Chainlink and thus has not initiated any callbacks to update the `ChainlinkFeedProxy` contracts. The proxies are "Waiting for first update from reactive bridge."
- **Reactive Bridge Status (for this feed):** Shows Updates Processed: 0, Last Price Seen: N/A USD, Last Update: N/A. This confirms that the `ChainlinkReactiveBridge` has not yet processed any events from the Chainlink aggregators.

OVERALL REACTIVE BRIDGE STATUS:

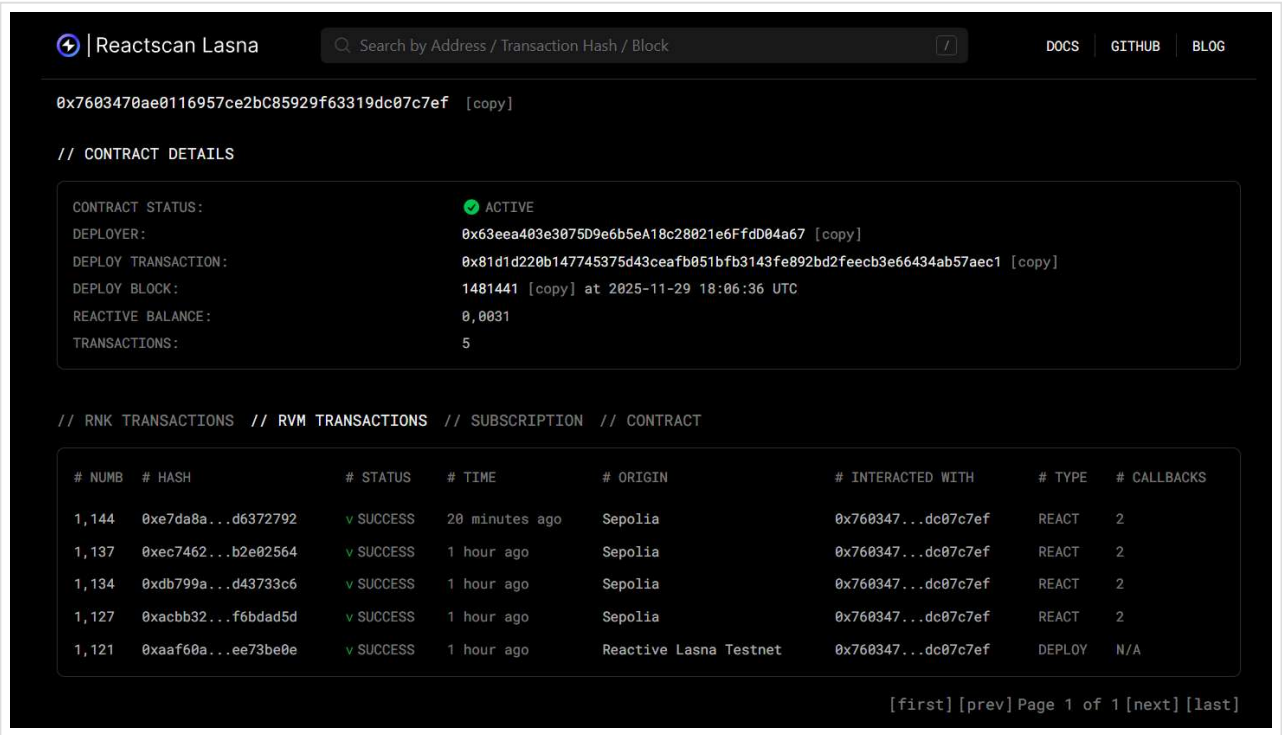
- **Bridge Contract Address:** `0x7603470ae0116957ce2bC85929f63319dc07c7ef`
- **Balance:** 0.1 REACT
- **Outstanding Debt:** 0.0 REACT
- **Owner/RVM ID (Deployer):** `0x63eea403e3075D9e6b5eA18c28021e6FfdD04a67`
- **Origin Chain ID:** 11155111 (Sepolia)
- **Destination Chain ID:** 11155111 (Sepolia)
- **Total Monitored Feeds:** 3
    - Aggregator 1: `0x719E22E3D4b690E5d96cCb40619180B5427F14AE` (ETH/USD)
    - Aggregator 2: `0x17Dac87b07EAC97De4E182Fc51C925ebB7E723e2` (BTC/USD)

- Aggregator 3: `0x5A2734CC0341ea6564dF3D00171cc99C63B1A7d3` (LINK/USD)

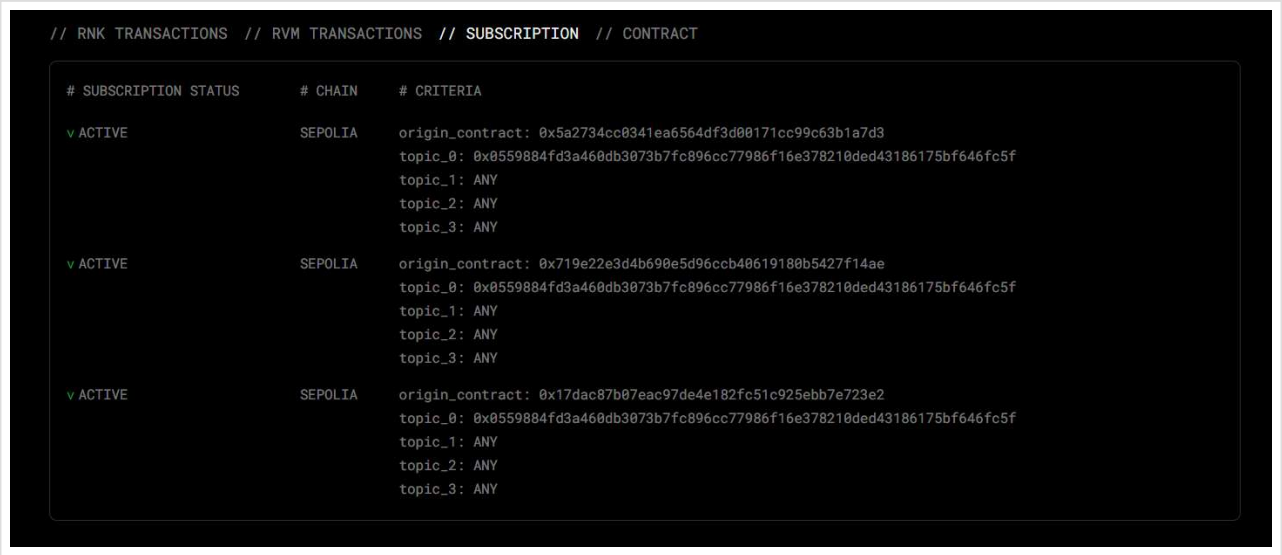
Manual cast Queries (initial state):

- `cast call $BRIDGE_ADDR "lastUpdateTimestampForAggregator(address)" $CHAINLINK_BTC_USD_AGGREGATOR --rpc-url $REACTIVE_RPC | cast to-dec` returns 0.
- `cast call $BRIDGE_ADDR "lastAnswerForAggregator(address)" $CHAINLINK_BTC_USD_AGGREGATOR --rpc-url $REACTIVE_RPC | cast to-dec` returns 0.
- `cast call $BRIDGE_ADDR "updateCountForAggregator(address)" $CHAINLINK_BTC_USD_AGGREGATOR --rpc-url $REACTIVE_RPC | cast to-dec` returns 0.

Screenshot: Contract Details & RVM Transactions



Screenshot: Subscriptions



Step 4: Funding ChainlinkReactiveBridge

The `ChainlinkReactiveBridge` contract was initially inactive due to insufficient funds. It was funded with 0.1 REACT tokens via the Reactive Network's system contract.

- **Command:** `cast send --rpc-url $REACTIVE_RPC --private-key $PRIVATE_KEY 0x0000000000000000000000000000000000fffFfF "depositTo(address)" 0x7603470ae0116957ce2bC85929f63319dc07c7ef --value 0.1ether`
- **Transaction Hash:** `0xe86219a1fc602ca0998e9f42da489fef085b7eba3858cee19a13b3a8b5df6004`
- **Balance Update:** The contract balance increased to approximately 0.0031 REACT, and the contract status on Reactscan became 'Active'.

VI. Deployed Contract Addresses

Ethereum Sepolia Testnet:

- **Deployer Address (RVM ID for Reactive Contract):** `0x63eea403e3075D9e6b5eA18c28021e6FfdD04a67`
- **ChainlinkFeedProxy (ETH/USD):** `0x120a855499cC8e777cC1D06078Aff78d78bAa8f2`
- **ChainlinkFeedProxy (BTC/USD):** `0x9a1b222D6AC1467b4302CEE2fd28074B0bD8367f`
- **ChainlinkFeedProxy (LINK/USD):** `0xDBF1CdB10956ba2474B12042f78DB1A61061032F`

Reactive Lasna Testnet:

- **ChainlinkReactiveBridge Address:**
  `0x7603470ae0116957ce2bC85929f63319dc07c7ef`