Blockchain-Based Peer-to-Peer Data Exchange System

A decentralized system for secure data exchange between peers using Ethereum smart contracts and SQLite databases.

Table of Contents

System Overview

Architecture

- **Blockchain Layer:** Ethereum smart contract handles request routing and response storage.
- **Peer Layer:** Python nodes listen for requests and execute SQL queries.
- **Data Layer:** SQLite databases store peer-specific data.
- **Network Layer:** Ganache provides local Ethereum blockchain.

Key Features

- **Targeted Requests:** Send queries to specific peers using Ethereum addresses.
- **Automatic Response:** Peers automatically respond to targeted requests.
- **Blockchain Verification:** All transactions are recorded on-chain.
- **Dynamic Gas Management:** Automatic gas estimation prevents failures.
- **SQL Query Execution:** Execute arbitrary SQL queries on peer databases.

Prerequisites

Software Requirements

- **Node.js** (v14+): For Truffle framework
- **Python** (v3.8+): For peer node scripts
- **Solidity Compiler** (v0.8+): For smart contract compilation
- **Ganache:** Local Ethereum blockchain
- **Truffle:** Smart contract deployment framework

Python Dependencies

```
pip install web3 getpass sqlite3 eth-utils
```

Node.js Dependencies

```
npm install -g truffle ganache
```

Installation & Setup

1. Project Structure

```
Peer_Network/
├── contracts/
│   └── DataTransfer.sol
├── migrations/
│   └── 2_deploy_contracts.js
├── build/
│   └── contracts/
├── peer_node.py
├── peer1.db
├── peer2.db
├── peer3.db
└── truffle-config.js
```

2. Initialize Truffle Project

```
mkdir Peer_Network && cd Peer_Network
truffle init
```

3. Configure Truffle

Create `truffle-config.js`:

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*",
      gas: 6721975,
      gasPrice: 20000000000
    }
  },
  compilers: {
    solc: {
      version: "0.8.19"
    }
  }
};
```

Smart Contract

`contracts/DataTransfer.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DataTransfer {
    struct Request {
        address requester;
        address target;
        string dbQuery;
        bool fulfilled;
        string response;
    }

    mapping(uint => Request) public requests;
    uint public nextRequestId = 1;

    event RequestCreated(
        uint indexed requestId,
        address indexed requester,
        address indexed target,
        string dbQuery
    );
```

```
    event ResponseSent(
        uint indexed requestId,
        address indexed responder,
        string response
    );

    function createRequest(address _target, string calldata _dbQuery)
        external returns (uint)
    {
        uint requestId = nextRequestId++;
        requests[requestId] = Request({
            requester: msg.sender,
            target: _target,
            dbQuery: _dbQuery,
            fulfilled: false,
            response: ""
        });
        emit RequestCreated(requestId, msg.sender, _target, _dbQuery);
        return requestId;
    }

    function submitResponse(uint _requestId, string calldata _response)
external {
        Request storage req = requests[_requestId];
        require(!req.fulfilled, "Request already fulfilled");
        req.fulfilled = true;
        req.response = _response;
        emit ResponseSent(_requestId, msg.sender, _response);
    }

    function getRequest(uint _requestId) external view returns (
        address requester,
        address target,
        string memory dbQuery,
        bool fulfilled,
        string memory response
    ) {
        Request storage req = requests[_requestId];
        return (req.requester, req.target, req.dbQuery, req.fulfilled,
req.response);
    }
}
```

Migration Script

Create `migrations/2_deploy_contracts.js`:

```
const DataTransfer = artifacts.require("DataTransfer");

module.exports = function(deployer) {
  deployer.deploy(DataTransfer);
};
```

Python Peer Node

Core Features

- **Event Listening:** Monitors blockchain for new requests.
- **Target Filtering:** Only responds to requests directed at this peer.
- **SQL Execution:** Runs queries on local SQLite database.
- **Dynamic Gas:** Estimates gas requirements automatically.
- **Transaction Confirmation:** Waits for blockchain confirmation.

Key Functions

Request Handler

```
def handle_query(db_path, query):
    """Execute SQL query on local database"""
    try:
        conn = sqlite3.connect(db_path)
        cur = conn.cursor()
        cur.execute(query)
        result = cur.fetchall()
        conn.close()
        return json.dumps(result)
    except Exception as e:
        return f"Error: {str(e)}"
```

Event Listener

```
def listen_for_requests():
    """Listen for blockchain events and respond if targeted"""
    while True:
        # Monitor new blocks for RequestCreated events
```

```
        # Filter by target address
        # Execute query and send response
```

Request Creator

```
def make_request():
    """Create new targeted request"""
    query = input("Enter SQL query: ")
    target = input("Enter target peer address: ")
    # Validate address and send transaction
```

Deployment Guide

1. Start Ganache

```
ganache --wallet.deterministic --accounts 10 --host 0.0.0.0
```

Note the output:

- Available Accounts (addresses)
- Private Keys
- RPC Server address

2. Compile Smart Contract

```
truffle compile
```

3. Deploy to Ganache

```
truffle migrate --network development
```

Save the contract address from output:

```
DataTransfer: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

4. Create Peer Databases

```
# Create sample data for testing
sqlite3 peer1.db << EOF
CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT);
INSERT INTO users VALUES (1, 'Alice', 'alice@example.com');
```

```
INSERT INTO users VALUES (2, 'Bob', 'bob@example.com');
EOF

sqlite3 peer2.db << EOF
CREATE TABLE products (id INTEGER PRIMARY KEY, name TEXT, price REAL);
INSERT INTO products VALUES (1, 'Laptop', 999.99);
INSERT INTO products VALUES (2, 'Mouse', 25.99);
EOF
```

Usage Instructions

1. Start Peer Nodes

Terminal 1 (Peer 1):

```
python peer_node.py
```text
Enter Ganache URL [default: [http://127.0.0.1:8545](http://127.0.0.1:8545)]:
Enter contract address: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Enter your private key: [paste private key]
Enter database path: peer1.db
✅ Connection successful! Account:
0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
```

Terminal 2 (Peer 2):

```
python peer_node.py
```text
Enter database path: peer2.db
✅ Connection successful! Account:
0x70997970C51812dc3A010C7d01b50e0d17dc79C8
```

2. Make Targeted Request

From Peer 1:

```
> request
Enter SQL query: SELECT * FROM data
Enter target peer address: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
📨 Request sent to 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
✅ Request confirmed in block 5
```

Peer 2 automatically responds:

```
📩 New request 1 (TARGETED): SELECT * FROM data
📤 Response submitted for request 1
✅ Response confirmed in block 6
```

3. Check Response

From Peer 1:

```
> response
Enter request ID: 1
🔍 Request 1 details:
    Requester: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
    Target: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8
    Query: SELECT * FROM data
    Status: ✅ Fulfilled
    Response (parsed):
      [X, 'XXXXX', XXXX]
```

Troubleshooting

Common Issues

1. "Out of Gas" Errors

- **Problem:** Transactions fail with insufficient gas
- **Solution:** Script uses dynamic gas estimation with 50k buffer

```
gas_estimate = contract.functions.createRequest().estimate_gas()
gas = gas_estimate + 50000
```

2. "Address Invalid" Errors

- **Problem:** Invalid Ethereum address format
- **Solution:** Use addresses from Ganache "Available Accounts" section

```
if not w3.is_address(target_address):
    print("❌ Invalid Ethereum address")
```

3. "Connection Failed" Errors

- **Problem:** Cannot connect to Ganache
- **Solutions:**
  - Ensure Ganache is running on port 8545
  - Check firewall settings
  - Verify RPC server address

4. "Request Already Fulfilled" Errors

- **Problem:** Multiple peers responding to same request
- **Solution:** System now filters by target address

```
if target.lower() == acct.address.lower():
    # Only process if we're the target
```

5. Database Errors

- **Problem:** SQL syntax or missing tables
- **Solutions:**
  - Verify table names exist: `.tables` in sqlite3
  - Check SQL syntax: `SELECT * FROM table_name`
  - Create test data as shown in deployment guide

API Reference

Smart Contract Functions

`createRequest(address _target, string _dbQuery)`

- **Purpose:** Create new data request
- **Parameters:**
  - `_target`: Ethereum address of target peer
  - `_dbQuery`: SQL query to execute
- **Returns:** Request ID (uint256)
- **Events:** `RequestCreated`

`submitResponse(uint _requestId, string _response)`

- **Purpose:** Submit response to request
- **Parameters:**
  - `_requestId`: ID of request to respond to
  - `_response`: JSON string response
- **Events:** `ResponseSent`

`getRequest(uint _requestId)`

- **Purpose:** Retrieve request details
- **Returns:** (`requester` , `target` , `dbQuery` , `fulfilled` , `response`)

This documentation provides a complete guide for deploying and operating the blockchain-based peer-to-peer data exchange system. The system enables secure, targeted data requests between peers using Ethereum smart contracts and local SQLite databases.