Building a Local Proof-of-Work Blockchain with Node.js

Overview

In this project, we will create a local blockchain that mimics a simplified proof-of-work (PoW) system using Node.js. The objective is to provide you with hands-on experience in blockchain fundamentals, including block generation, transaction management, proof-of-work consensus, and wallet security. We will also build a web interface to visualize the blockchain, mempool, and to interact with transactions.

System Architecture

Components

1. **Server (Blockchain Node)**
    a. **Express Framework:** Used to set up an HTTP server that manages the blockchain and serves the web UI.
    b. **Clock and Timing:** The server controls block creation every 10 minutes (this interval is customizable).
    c. **Mempool:** The server collects and validates transactions from clients, storing them in a public mempool.
    d. **File-based Database:** Each block is stored as an individual file.
2. **Client (Miner/Bot & Transaction Sender)**
    a. **Node.js Bot:** A client developed in Node.js that performs two roles:
        i. **Mining:** Competes to generate a block by performing the PoW calculation.
        ii. **Transaction Submission:** Sends transactions to the server.
    b. **Wallet Generation:** Generates a wallet consisting of a public key (for the address) and a private key (for signing transactions). Note that transaction signatures will be created manually using a simple Node.js program, and the signing process is left for you to design.
3. **Web Application UI**
    a. **Blockchain Display:** Visualizes the list of confirmed blocks.
    b. **Mempool Display:** Shows the current pending transactions.
    c. **Transaction Interface:** Provides a section for users to create and send transactions, including input fields for transaction details and fees.

d. **Wallet Balances:** Displays the current balance of each wallet (based on public keys).

Detailed Project Workflow

1. Block Generation and Timing

- **Block Timing:** The server creates blocks at fixed intervals (default 10 minutes, but this can be adjusted). At the start of each cycle, the server updates its mempool with new transactions.
- **Mempool Refresh:** Once a block is confirmed, the server clears the included transactions from the mempool and refreshes it with any new incoming transactions.

2. Transaction Management

- **Transaction Creation:**
  - Clients can manually create transactions on a dedicated section of the web app.
  - Transactions include:
    - Sender's public key (address).
    - Recipient's address.
    - Amount of uemfCoin to transfer.
    - Transaction fee.
    - A digital signature, which the student must generate manually using a simple Node.js signing program.
- **Transaction Verification:**
  - When the server receives a transaction, it verifies:
    - The digital signature.
    - The sender's balance (sufficient funds).
    - Other transaction parameters (e.g., valid fee).
- **Adding to Mempool:** Once validated, the transaction is added to the mempool, where it will be available for inclusion in the next block.

3. Proof-of-Work Algorithm

- **Mining Process:** Every block cycle (10 minutes), clients (miners) pick transactions from the mempool and attempt to generate a block by:
  - Assembling a candidate block with selected transactions.
  - Solving the cryptographic puzzle based on a preset difficulty.

- **Block Submission:** The first client to solve the puzzle submits the block to the server.
- **Reward:** The winning miner receives a reward of 50 uemfCoin as specified in the coinbase transaction.

4. Server Customizations

The blockchain server can be tailored through the following configurable parameters:

- **Block Reward:** Default is 50 uemfCoin, but can be modified.
- **Difficulty:** Adjust the complexity of the PoW puzzle.
- **Coinbase Name:** Defaults to uemfCoin but can be renamed.
- **Block Interval:** Change the time between blocks (default is 10 minutes).

5. Wallet and Address Management

- **Wallet Generation:**
  - The client bot generates a wallet (public and private keys).
  - The public key acts as the wallet address and is shared publicly.
  - The private key remains confidential and is used to sign transactions.
- **Manual Transaction Signing:**
  - The transaction signing will be done manually using a simple Node.js program.
  - Students must devise a method to sign the transaction data and generate a digital signature.
- **Server Wallet View:** The web app will display wallet balances by reading the public keys stored in the blockchain.

6. Web UI and Express Server

- **Express Server:** The server handles:
  - API endpoints for transaction submission, block mining, and wallet management.
  - Serving the web UI, built using HTML and CSS.
- **UI Features:**
  - **Block Visualization:** Displays the blockchain as a series of blocks.
  - **Mempool Visualization:** Shows pending transactions awaiting inclusion.
  - **Transaction Section:** Allows users to create transactions, specify fees, and manually enter a signature.
  - **Wallet Balances:** Provides a view of each wallet's uemfCoin holdings.

Project Workflow Summary

1. **Server Setup:**
    a. Start the Express server.
    b. Initialize the blockchain with the genesis block.
    c. Set up a timer to trigger block creation every 10 minutes (customizable).
2. **Transaction Processing:**
    a. Users create transactions via the web app.
    b. Clients send these transactions to the server.
    c. The server validates transactions (including manual digital signature verification) and adds valid ones to the mempool.
3. **Client Mining and Block Generation:**
    a. Clients select transactions from the mempool and begin the PoW process.
    b. The first client to solve the PoW submits a valid block to the server.
    c. The server confirms the block, updates the blockchain, and rewards the miner.
4. **UI Updates:**
    a. The web app updates to display the latest blockchain, mempool transactions, and wallet balances.
    b. A transaction interface is provided for users to create and submit transactions, including specifying tx fees.

Conclusion

This project will deepen your understanding of blockchain concepts by having you build a fully functional local blockchain using Node.js. You will work through key aspects like transaction management with fees, manual transaction signing, and a PoW consensus mechanism, while also creating a web interface to interact with the blockchain.