



Department of Information and Communication Technology
Group Project

Ucoin - The truth behind blockchain-based applications

Supervisor: Dr. Nguyen Minh Huong

Members:

- | | |
|-----------------------|----------|
| • Duong Tuan Minh | BI11-172 |
| • Nguyen Thi Ha Trang | BI11-267 |
| • Nguyen Hoang Minh | BI11-181 |
| • Nguyen Tien Cong | BI11-047 |
| • Le Gia Hien | BI11-079 |
| • Nguyen Viet Hung | BI11-103 |

Contents

List of Tables	I
List of Figures	II
1 Introduction	1
1.1 Problem Statement	1
1.2 Objectives	1
2 Background	2
2.1 Definitions	2
2.2 Mechanism	3
2.3 Applications	4
3 Features Design	5
4 Startup Operations Implementation	6
4.1 Network Initialization	6
4.1.1 Initialize Sockets	6
4.1.2 Discover Peers	6
4.2 Local Chain Initialization	7
4.2.1 Load the genesis block	7
4.2.2 Understand blockchain components	8
4.2.3 Create the local database	10
4.3 Local Chain Synchronization	12
4.4 Wallet Features Initialization	13
4.4.1 Initialize address	13
4.4.2 Start API server	16
5 Normal Operations Implementation	17
5.1 Handling Messages	17
5.1.1 Overview	17
5.2 Network Protocol	17
5.3 Mining	18
5.3.1 Proof-of-work	18
5.3.2 Mining Process	19
5.3.3 Block Validation	20
5.4 Handling Transactions	22
5.4.1 Transaction Creation	22
5.4.2 Transaction Validation	23
6 Results	25
6.1 Scenario Design	25
6.2 Simulation Result	26
6.3 Mining Rate Result	34
7 Conclusion	37
Bibliography	38

List of Tables

1	Description of block_headers table	11
2	Description of transactions table	11
3	Description of tx_outputs table	12
4	Description of tx_inputs table	12
5	Difficulty and Mining Rate Summary	35

List of Figures

2.1	The structure of a blockchain [1]	2
2.2	Centralized and decentralized networks [2]	3
2.3	The mechanism of a blockchain [1]	4
3.1	The functionalities of an Ucoin node runs in different threads	5
4.1	Handshake flow between two nodes A and B	7
4.2	Node C connects to A after receiving addr message forwarded by B . . .	7
4.3	Structure of a transaction	8
4.4	Structure of a transaction input	8
4.5	Structure of a transaction output	9
4.6	Structure of a script	9
4.7	Structure of a block	10
4.8	Structure of a block header	10
4.9	Ucoin local SQLite database design	11
4.10	Node A requests missing blocks from B to synchronize its local chain . .	13
4.11	Private key, public key, and Ucoin address [3]	13
4.12	Generate Ucoin address from the public key [3]	15
4.13	Base58Check Encoding Flow [3]	15
5.1	Example Merkle tree and Merkle root computation process	20
5.2	Block validation flow	21
5.3	Transaction creation flow	23
5.4	Transaction validation flow	24
6.1	Ucoin Network Architecture of the experiment	25
6.2	Docker containers for the experiments	26
6.3	Blocks Page Interface	27
6.4	Details of block number 2	27
6.5	Information about address a particular Ucoin address	28
6.6	Peer connections status in Peers page	28
6.7	Our node's balance, transaction history, and UTXOs	29
6.8	Button to create transaction	29
6.9	Warning for sending to an incorrect address	30
6.10	Warning for not enough coins to send	30
6.11	Logs of transaction creation process	30
6.12	Transaction added to mempool	31
6.13	Transaction is automatically removed from mempool	31
6.14	Logs of mempool updates	32
6.15	The block containing the created transaction	32
6.16	Transaction history	33
6.17	The successfully created transaction	34
6.18	Difficulty of the first 20 periods (50 blocks each)	36
6.19	Mining rate of the first 20 periods (50 blocks each)	37

1 Introduction

1.1 Problem Statement

Blockchain was first introduced in 2008 as the distributed ledger behind the bitcoin network. However, the technology has taken on a life of its own, attracting interest from several industries [4]. Nowadays, blockchain goes far beyond cryptocurrency, especially with its ability to improve transparency and equality while saving time and money.

Blockchain is a decentralized, distributed database that stores a growing list of records called blocks. Each block contains a timestamp and a link to the previous block, forming a chain of blocks and making the stored data secure, immutable, and transparent [5].

Blockchain has the potential to revolutionize a wide variety of fields, including but not limited to financial transactions, supply chain management, and voting systems. Its characteristics make data more secure, transparent, and harder to change but easier to trace than traditional infrastructure. Therefore, governments, businesses, and other organizations are researching and deploying blockchain technology to meet those needs.

1.2 Objectives

More and more countries worldwide are recognizing blockchain technology's significant potential. Besides, although being more secure, like any other new and powerful technology, blockchain can still be a target for several malicious attacks. Cybercriminals may take advantage of vulnerabilities to carry out fraudulent campaigns and steal users' identities and assets. Therefore, our objectives for pursuing this project are:

1. Understand how blockchain works by researching the technical aspects of the Bitcoin network
2. Develop a Python program to reproduce a simplified blockchain-based application, specifically a cryptocurrency named Ucoin. The application must be able to
 - (a) Support networking to connect with other machines
 - (b) Manage a local copy of the blockchain
 - (c) Solve a mathematical problem to mine for blocks
 - (d) Serve an API for users to interact with the blockchain
3. Develop a user-friendly web interface using Typescript and React to create transactions and examine the status of the current blockchain
4. Simulate a blockchain network by using Docker containers to represent several machines and demonstrate adding a new record to the chain by creating transactions
5. Evaluate how the network stabilizes the mining rate of new blocks by periodically

adjusting the difficulty

6. Document the program so it can later be applied in future projects, for example, as a target to research vulnerabilities and defense strategies of blockchains.

2 Background

2.1 Definitions

A blockchain is a decentralized, distributed, and public digital ledger that records transactions across many computers so that the data cannot be altered [6]. In the blockchain, records or transactions are collected into groups called blocks. As presented in figure 2.1, these blocks are linked by their previous block's hash, forming an ordered chain.

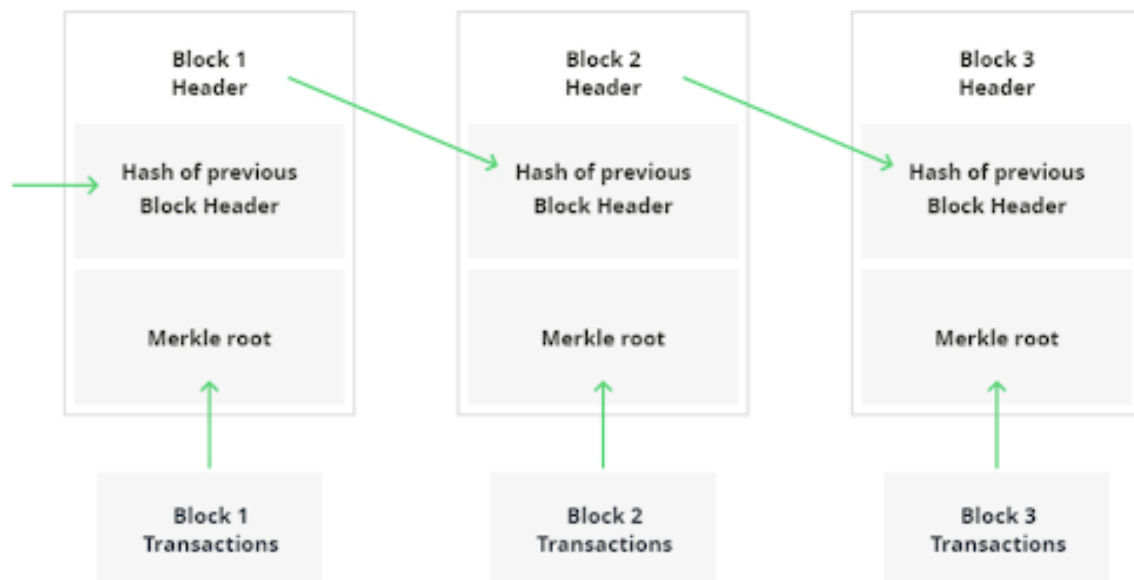


Figure 2.1: The structure of a blockchain [1]

Unlike standard databases, which store data in centralized servers, blockchain is stored across an open, peer-to-peer (P2P) network. A P2P network is a combination of computers linked to each other, allowing digital information to be distributed across all machines participating in the network instead of a central server [5]. If centralized authorities control the data, they can easily manipulate it in favor of their benefits. Therefore, the distributed property of blockchain provides transparency, trust, and data security as data is controlled by the entire network. The difference between a decentralized and a centralized network is shown in figure 2.2.

Each participant maintains, approves, and updates new entries of a local blockchain which must be synchronized across the network. Each member ensures that all records are in order resulting in validity and security. Thus, different parties or organizations can easily reach any agreement, such as business contracts, without relying on trust.

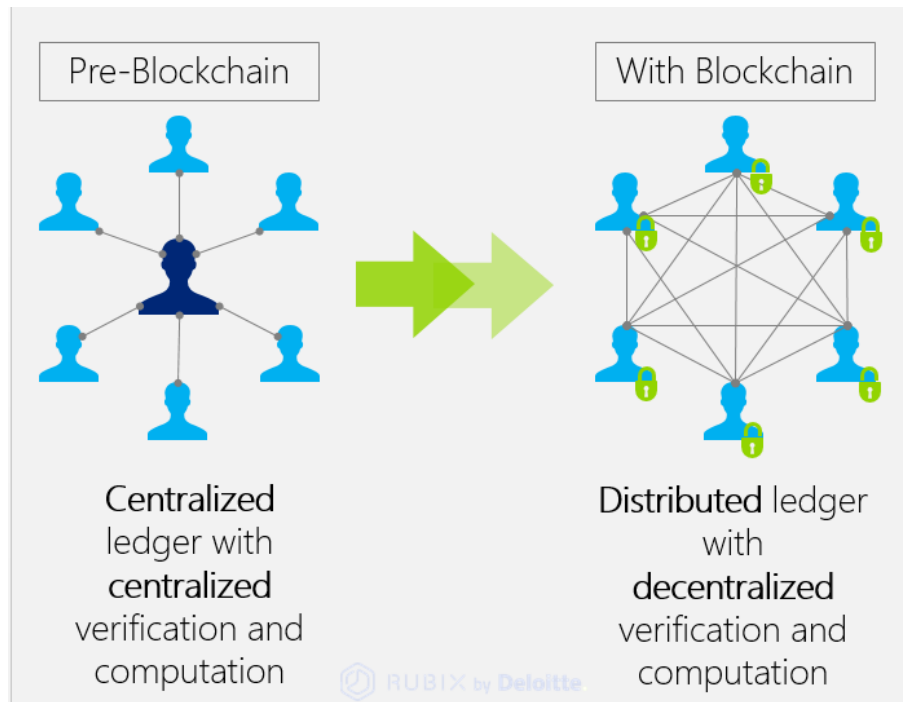


Figure 2.2: Centralized and decentralized networks [2]

2.2 Mechanism

Records or transactions are inserted into the blockchain by creating new blocks. Each record must be proven and digitally signed to ensure its genuineness. Each new user (node) joining the blockchain peer-to-peer network receives a full copy of the system. Once a new block is created, it is sent to each node within the blockchain system. Then, each node verifies the block and checks whether the information stated in it is correct. Finally, if everything is verified, each node adds the block to its local blockchain. This process can be visualized in figure 2.3.

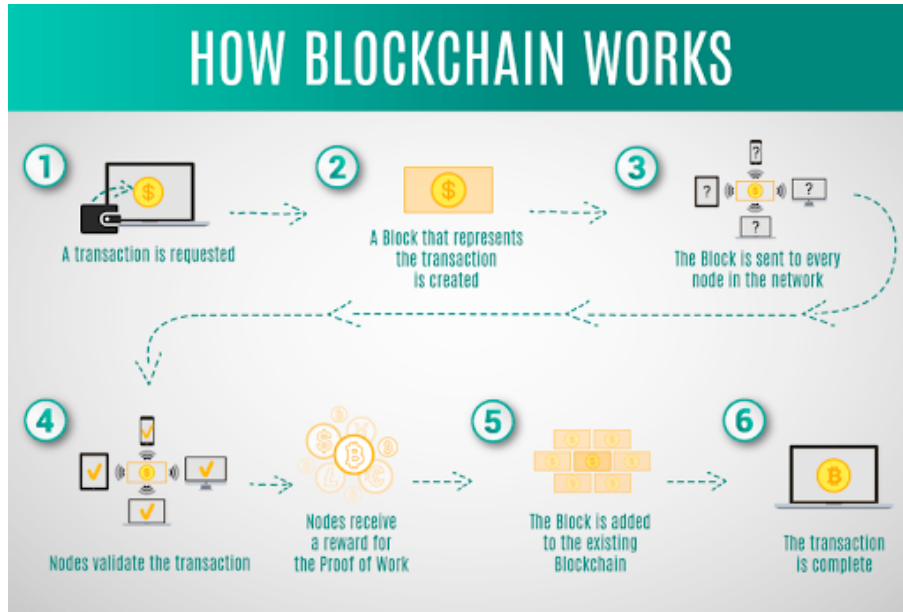


Figure 2.3: The mechanism of a blockchain [1]

2.3 Applications

Nowadays, blockchain technology is utilized in many fields:

Banking:

Blockchain improves security, reduces fraud, and increases efficiency. For example, banks could use blockchain to verify and record financial transactions and track assets or contracts to reduce the need for a middleman and improve the speed of financial transactions [7].

IoT:

Blockchain technology securely stores and manages data from connected devices, as well as facilitates communication and automation between these devices. For example, a smart home system could use a blockchain-based platform to store data from connected devices such as thermostats and security cameras and to automate tasks using this data, such as adjusting the temperature or turning on the light [8].

Healthcare:

Blockchain improves data security and patient privacy, as well as increases the efficiency of medical processes. One example is the Electronic Medical Record (EMR) system, which can use this technology to securely store and manage patient data, such as medical history, test results, and prescriptions [9].

Government:

Blockchain has the potential to cut down millions of hours of red tape every year, hold public officials accountable through smart contracts, and provide transparency by recording a public record of all activity. Blockchain-based voting could improve civic engagement by providing a level of security and incorruptibility that allows voting to be done on mobile devices [10]. The government could drastically reduce

identity theft claims and improve the security and privacy of personal identity information by keeping social security numbers, birth certificates, birth dates, and other sensitive information on a decentralized blockchain ledger [11].

Media:

Blockchain technology manages and tracks the ownership and facilitates the distribution and payment of digital content such as music, videos, and images. For instance, a music streaming service could use a smart contract to automatically pay artists each time their songs are played, with the contract terms and the transaction history recorded on the blockchain [12].

3 Features Design

All machines (nodes) in Ucoin peer-to-peer network are identical in terms of functionalities. As illustrated in figure 3.1, each node utilizes multithreading to handle four primary functionalities simultaneously, including routing messages, managing blockchain, mining blocks, and serving API.

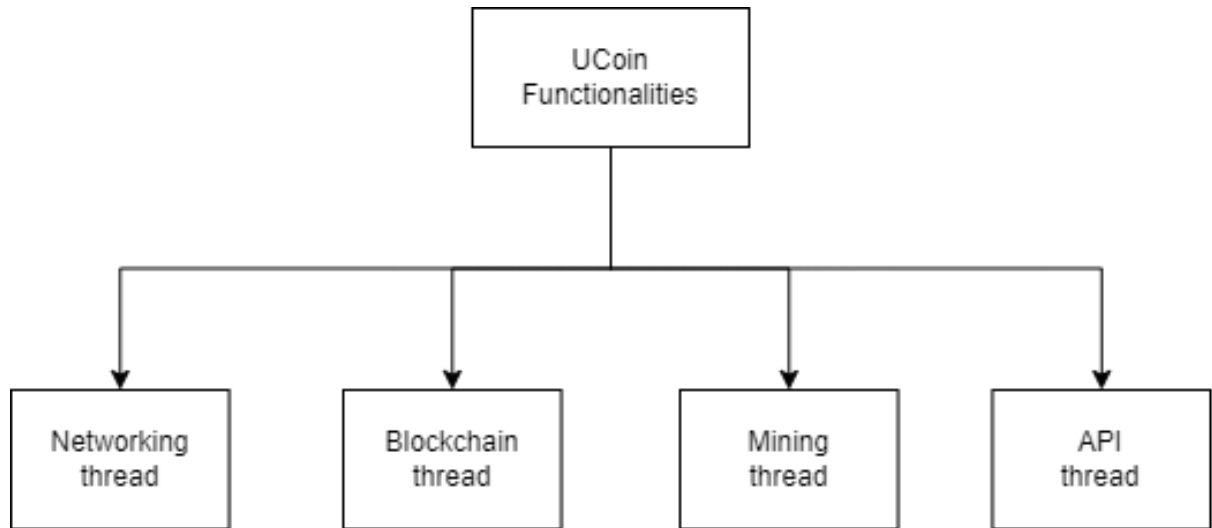


Figure 3.1: The functionalities of an Ucoin node runs in different threads

The networking/routing functionality is required for every machine to participate in the Ucoin network. It uses a custom-designed application-layer network protocol on top of TCP/IP to facilitate communication between nodes in the P2P network by sending, receiving, and propagating messages.

The blockchain functionality is responsible for maintaining a complete and up-to-date copy of the blockchain in a local SQLite database. It runs in a thread that continuously checks an event queue for newly received blocks or transactions. It then validates these new blocks and transactions and decides whether to drop or continue processing them.

The mining functionality of each node loops constantly to solve a complex mathematical problem called the Proof of Work problem to find the next valid block before

other peers. The first miner to solve the problem receives a small block reward for their efforts. These block rewards incentivize miners and serve as the mechanism to create new coins through mining new blocks. The mining process is essential to secure the network and ensure that new transactions are valid and can be trusted.

The API functionality is an interface that runs via HTTP protocols and Web-Socket. This function allows node owners to create transactions and access the data stored inside their local blockchain. Additionally, it supports bi-directional, event-based communication to send data updates in real-time. It also functions as a backend server for a wallet web application to manage the account and balance with a more user-friendly UI.

These are the core functionalities that are required for Ucoin node to operate properly in two phases of the application: the startup phase and the normal operation phase. The startup phase includes initializing the network functionality, the local blockchain, and the wallet. The normal operations are handling network messages, mining for new blocks, as well as creating and validating new blocks and transactions.

4 Startup Operations Implementation

4.1 Network Initialization

4.1.1 Initialize Sockets

The network or routing function of each node in UCoin's peer-to-peer network plays the role of both a server and a client. Each node can listen as a server and send messages across the network as a client.

On the one hand, the server socket is initialized in a thread by binding and listening on a configurable port (by default, the server port is 8333). This server thread then loops endlessly to check for new incoming messages. The program spawns a new thread for each received message and handles the message differently depending on the content.

On the other hand, whenever a node needs to send data to its peer, it creates a new TCP socket on a random port. It uses this socket to connect to the peer (using the peer's IP address and server port), send the required data, and close the socket shortly afterward.

4.1.2 Discover Peers

After initializing the sockets, the program tries to discover peers to connect by iterating through a list of known IP addresses and ports embedded in the configuration file. It initiates a handshake with each known peer unless it has reached the configured maximum number of peers.

Every two nodes on the network must perform a "handshake" as described in figure 4.1 to establish a connection and begin exchanging information. The handshake starts by exchanging messages of type "version," which contain details about each node, such as its address, port, or the height of its local chain. Through this "version" message, each

node can record the maximum chain height of its peers, which is later used in the chain synchronization phase. Then, as a response, two nodes exchange “verack” messages, which acknowledge the receipt of “version” messages.

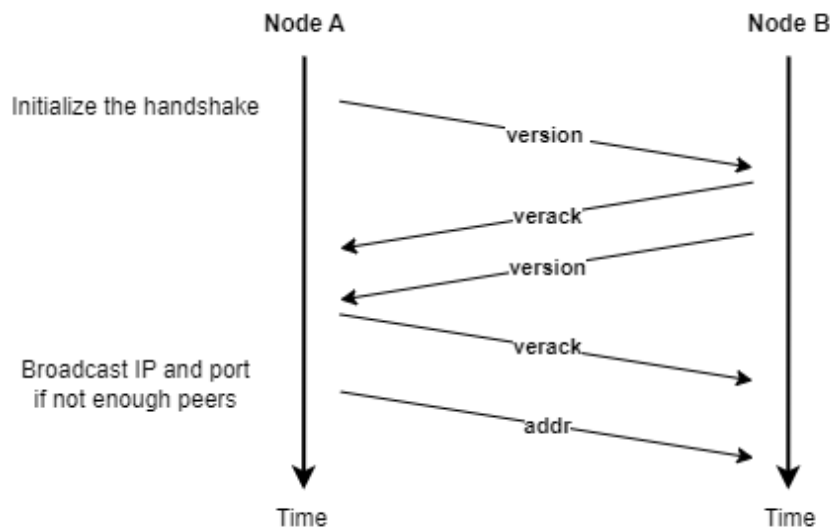


Figure 4.1: Handshake flow between two nodes A and B

In case node A has yet to reach the minimum required number of peers to function reliably, it starts broadcasting its IP address and port to all connected peers. These nodes then forward node A’s address to their peers, allowing new peers to connect with A. This process continues periodically until node A finds enough peers.

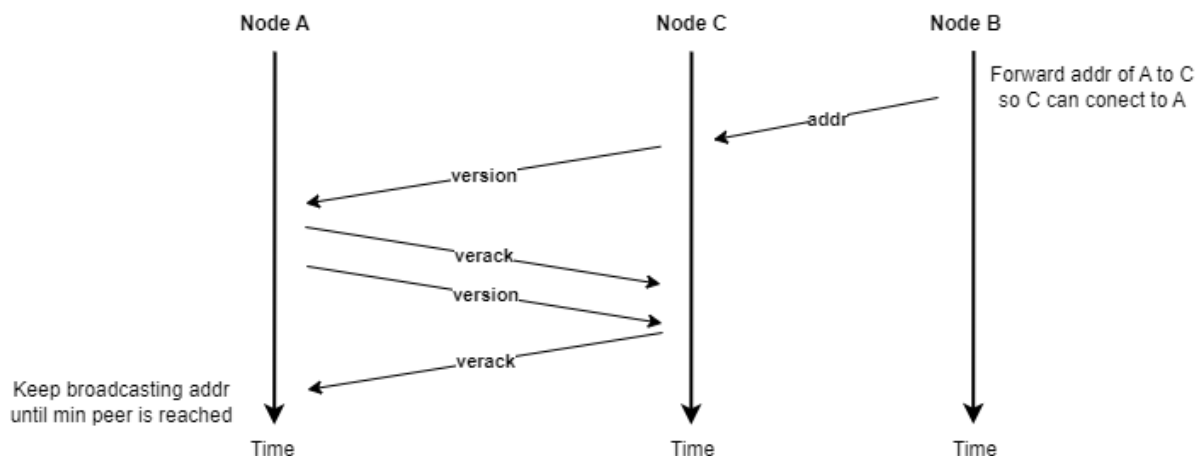


Figure 4.2: Node C connects to A after receiving addr message forwarded by B

4.2 Local Chain Initialization

4.2.1 Load the genesis block

The genesis block is the first block in the blockchain. It is a special block that is hardcoded into the source code. It is the starting point to initialize the local blockchain

of each node to ensure that all copies have a consistent and well-defined beginning.

In the Ucoin application, the genesis block is pre-mined and stored in a binary file called `genesis_block.dat`. This file is included in the source code and must be the same for all nodes running Ucoin. The program loads this file and parses the binaries into a Block object. This block is then inserted into the database at height 0 if the database is empty. Any modification of `genesis_block.dat` can cause the program to fail to parse it into a Block or to synchronize the local chain with peers.

4.2.2 Understand blockchain components

Transactions are the most critical components of the Ucoin blockchain as it allows users to transfer value between each other [13]. In figure 4.3, it can be seen that each transaction consists of inputs and outputs. Inputs define what coins are being spent, whereas outputs define where the coins will be transferred.

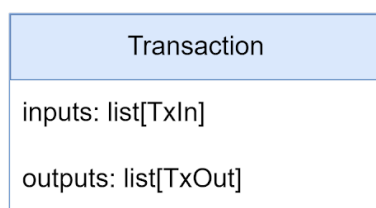


Figure 4.3: Structure of a transaction

In each transaction, the inputs attribute is a list of transaction inputs (TxIn). Each TxIn points to an output of a previous transaction, i.e., the source of the coin being spent. The TxIn refers to coins that have been sent to a user. As shown in figure 4.4, TxIn consists of a reference to the previously received coin and an unlocking script to prove coin ownership [13]. The total amount of input coin is not included in a transaction but instead is calculated by aggregating the amount assigned in all the referenced output.

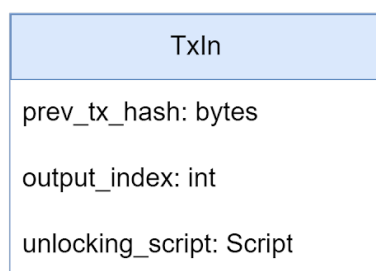


Figure 4.4: Structure of a transaction input

Similar to the inputs field, the outputs field of a transaction contains one or more transaction outputs (TxOut). Figure 4.5 presents two fields of a TxOut: amount and a locking script. The amount field is the number of coins assigned to a TxOut. The locking script can be understood as a locked box where anyone can deposit money, but only the

key owner can open the box to spend the money [13]. Once a transaction is created, a new unspent TxOut is generated until it is referenced in another transaction by the owner and becomes spent TxOut. The entire set of unspent transaction outputs (called UTXO set) represents all the available coins to spend in the network. This UTXO set helps prevent double-spending as the TxIn of a new transaction cannot refer to TxOut that is not in the set [14]. Additionally, the Ucoin blockchain does not manage user accounts and balances. It is up to wallet applications to scan the blockchain and aggregate the value of all UTXOs controlled by a key to calculate the total balance.

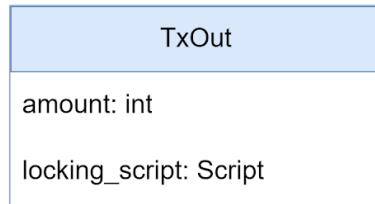


Figure 4.5: Structure of a transaction output

Inside each TxIn and TxOut, there are Script objects called unlocking and locking scripts. Script is a simple stack-based programming language used to determine whether a transaction is valid and can be added to the blockchain. A Script typically comprises a series of data and operations (e.g., comparison, arithmetic, and logical operations) stores as bytes as depicted in figure 4.6 [14]. A locking script of a TxOut contains the Ucoin address of its owner along with some operations. To create a transaction and spend this TxOut, the owner must sign the transaction with their private key and add an unlocking script containing the signature and their public key to the TxIn. Every node in the network can combine the unlocking with the locking script and execute them to verify the transaction.

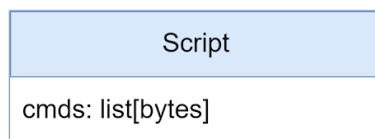


Figure 4.6: Structure of a script

Apart from transactions created by users, there is a unique type called coinbase transaction that does not have a previous TxOut referenced as input. A coinbase transaction is the first transaction in each block and functions as the reward for the miner who successfully mined the block [13]. Instead of consuming UTXO, coinbase transactions create new coins to reward miners, which is also the money supply mechanism for the blockchain network.

Ucoin blocks are collections of valid transactions that have been added to the blockchain. Each block also has a block header containing the metadata about the transactions in

the block, including the previous block's hash, the Merkle root, the timestamp, the bits, and the nonce. These properties are illustrated in figure 4.7 and 4.8

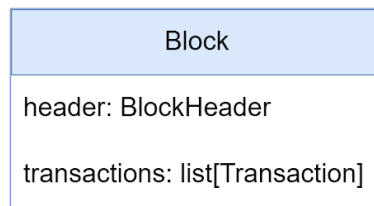


Figure 4.7: Structure of a block

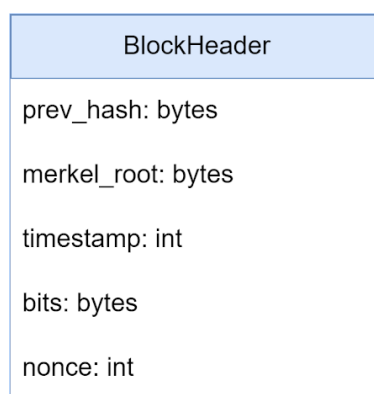


Figure 4.8: Structure of a block header

The previous block's hash is a hash of the previous block's header. It functions as a link that forms the chain of blocks, which makes any malicious attempts to modify any block in the chain can be easily detected and rejected by the network. The Merkle root summarizes a block's transactions into a single hash value. It allows the network to quickly verify the transactions' integrity without processing each transaction individually. The timestamp is a Unix-style timestamp representing the number of seconds since January 1, 1970. It stores the creation time of a block and is used for re-evaluating a new difficulty after a certain number of blocks. The bits field encodes the difficulty of the proof-of-work problem required to mine a block, whereas the nonce is the result that miners must look for to solve the problem and mine a block successfully.

4.2.3 Create the local database

The database design presented in figure 4.9 and the following tables is stored in a JSON file called "db.json" to improve readability and make it simpler to modify. If the program starts for the first time or the node owner decides to reset the database, the program loads this JSON file from the disk to get the table's property and execute SQL commands to create the SQLite database. As tablesBesides the field described in the previous section, special fields such as the block hash, the transaction hash, or the address of TxOut's owner were added to support the functionalities of a digital wallet.

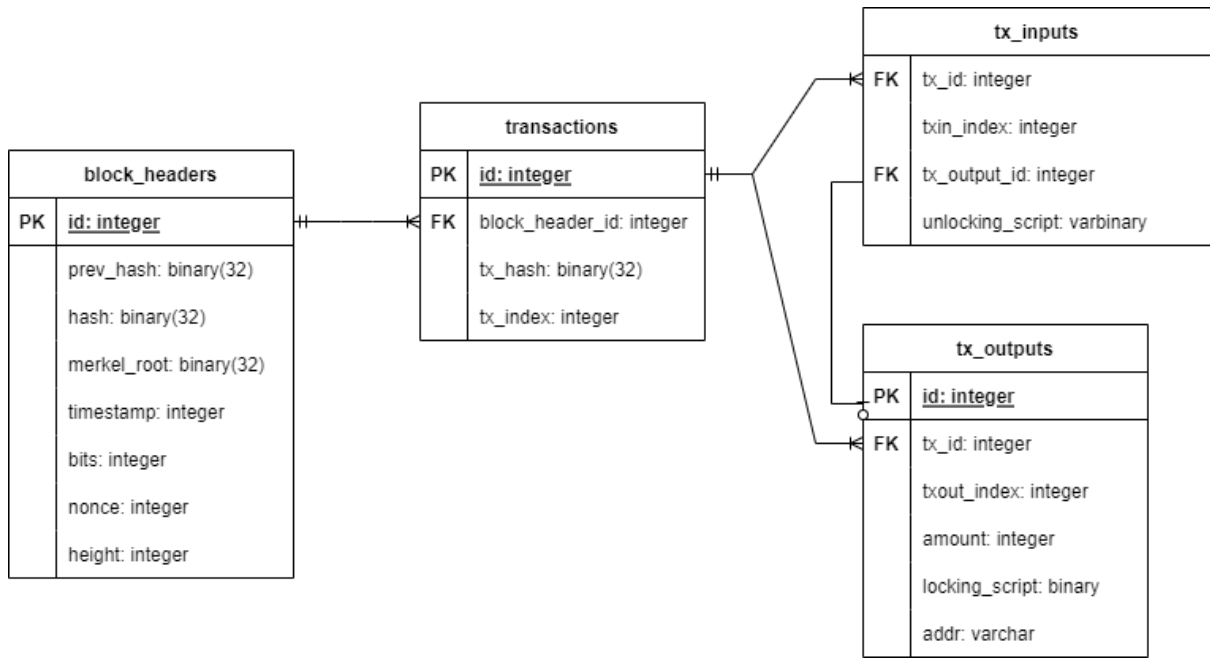


Figure 4.9: Ucoin local SQLite database design

Table 1: Description of block_headers table

Field	Description
id	The primary key uniquely identifies each row
prev_hash	The previous block's hash in the blockchain
hash	The current block's hash
merkel_root	The root of the Merkel tree as a representation of all transactions
timestamp	The block's creation time
nonce	The solution to the PoW problem
bits	The a representation of the target required to mine the block
height	The block's height in the chain

Table 2: Description of transactions table

Field	Description
id	The primary key uniquely identifies each row
block_header_id	A foreign key that references the "id" field in the "block_headers" table
tx_hash	The hash of the transaction
tx_index	An integer storing the index of the transaction within a block.

Table 3: Description of tx_outputs table

Field	Description
id	The primary key uniquely identifies each row
tx_id	A foreign key that references the "id" field in the "transactions" table
txout_index	An integer storing the index of the output within a transaction
amount	An integer storing the amount of coin associated with the output
locking_script	A binary string storing the locking script for the output
addr	A string storing the address of the output's owner

Table 4: Description of tx_inputs table

Field	Description
tx_id	A foreign key that references the "id" field in the "transactions" table
txin_index	An integer storing the index of the input within a transaction
tx_output_id	An integer field storing the ID of the TxOut referenced by this TxIn
unlocking_script	A binary string storing the unlocking script for the output

4.3 Local Chain Synchronization

After the sockets are successfully initialized, and the number of peers is higher than a pre-configured minimum value, each node starts to synchronize its local chain with peers.

This process begins with choosing a set of representative blocks from the local blockchain and collecting their hashes. These hashes are then broadcast to all peers using a protocol called "getblocks." After receiving a "getblocks" message, each peer compares the received hashes with its hashes to determine the latest shared blocks. Then each peer responds with an "inv" message containing the hashes of up to a maximum of 500 blocks after the shared one. When receiving the "inv," nodes divide the received hashes into different "getdata" messages and send each of them to their peers. The 500 block limits and the division of "getdata" messages are necessary to avoid overloading any node with excessive network traffic. For each block hash in the received "getdata" message, each peer will send back a "block" message containing the block data that the requested node is missing. Each received block is validated carefully before finally being added to the local chain.

Each node exchanges chain height with all its peers at the handshake stage and keeps track of the highest value. A node finishes synchronizing when its height reaches this value. Otherwise, the entire flow described in figure 4.10 repeats periodically until the local chain is in sync with other peers.

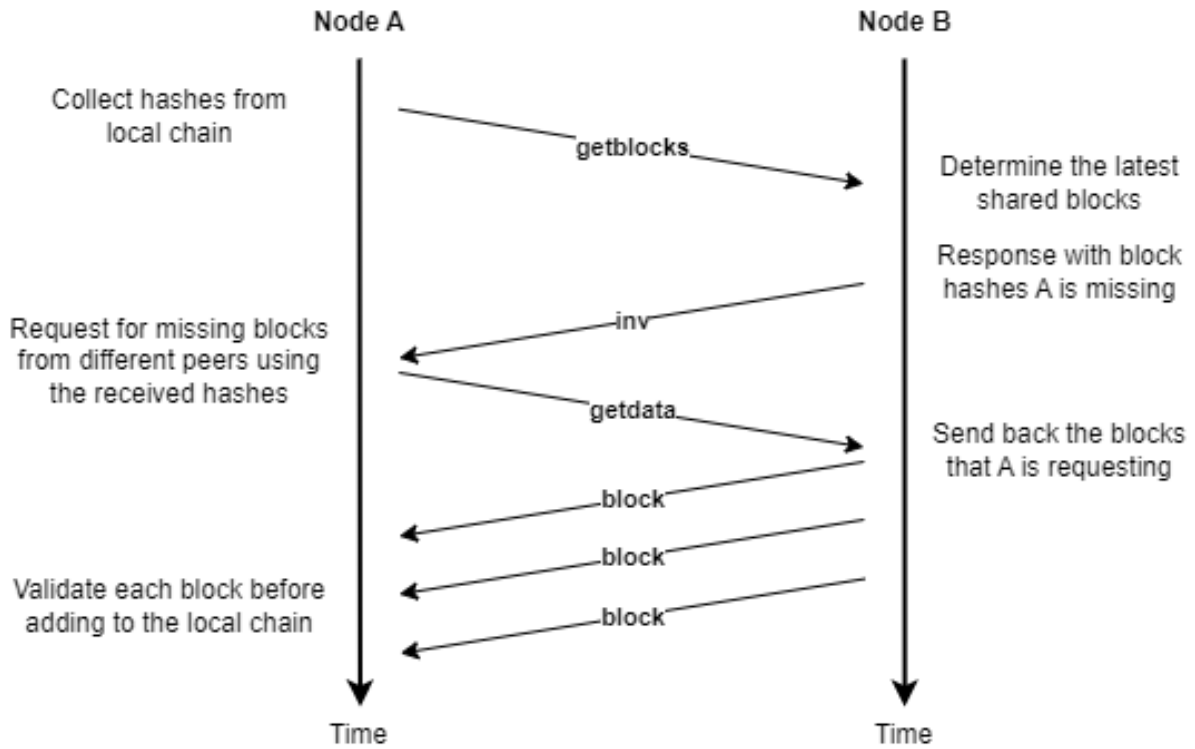


Figure 4.10: Node A requests missing blocks from B to synchronize its local chain

4.4 Wallet Features Initialization

Ucoin node allows user to manage their keys and address, which is a critical feature when creating transactions. With this feature and the API functionality, Ucoin program is able to support coin management functions such as sending money and checking balance, similar to a normal digital wallet.

4.4.1 Initialize address

Ucoins requires users to have an address in order to send and receive money to each other. The process of generating an address is illustrated in figure 4.11

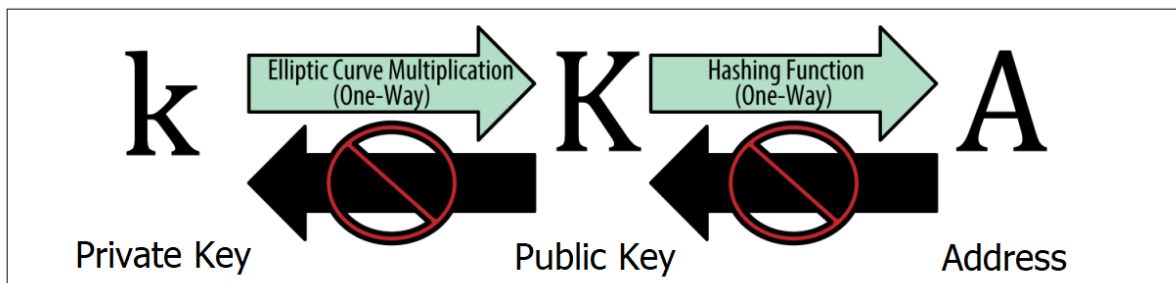


Figure 4.11: Private key, public key, and Ucoin address [3]

Ucoin uses Elliptic Curve Digital Signature Algorithm (ECDSA) to generate key pairs. This cryptography algorithm uses the mathematics of elliptic curves to create

digital signatures. It operates on a pair of keys, a private key used to sign digital messages and a public key used to verify the authenticity of messages. The process involves using the private key to generate a unique digital signature for a message and the public key to verify that the signature is authentic. In our case, users signed each transaction they created using their private key. The resulting signature provides authenticity and integrity to the transaction, as any changes will be easily detectable by other nodes in the network.

The private key is a large random number that the program can automatically generate or that users can generate by themselves. The exact method of choosing the number does not matter, but users should use a secure source of randomness that is not predictable and repeatable. Regardless of the method used, it is up to users to store their private key securely and keep it confidential, as compromising the key can result in permanent loss of access to their coins.

After loading the private key, the program derives the public key using elliptic curve multiplication. Users must provide their public key and a signature for the transaction as proof of ownership whenever they want to spend their coins. With the provided public key and signature, other nodes in the network can effortlessly verify the transaction's validity.

Lastly, as described in details in figure 4.12, an address is derived from the public key using a hash function and then encoded using Base58Check encoding. Base58 is a binary-to-text encoding scheme used in a more compact and human-readable form. It uses 58 characters from a selected character set to encode the data, which includes the digits 0-9 and the letters A-Z and a-z, excluding the characters that might look similar to each other, such as 0 (zero), O (capital o), I (capital i), and l (lowercase L). The “Check” part refers to an error-detection code included in the encoding, which allows users to detect and correct any typos in the address. The process of encoding a public key's hash into base58check format is shown in figure 4.13.

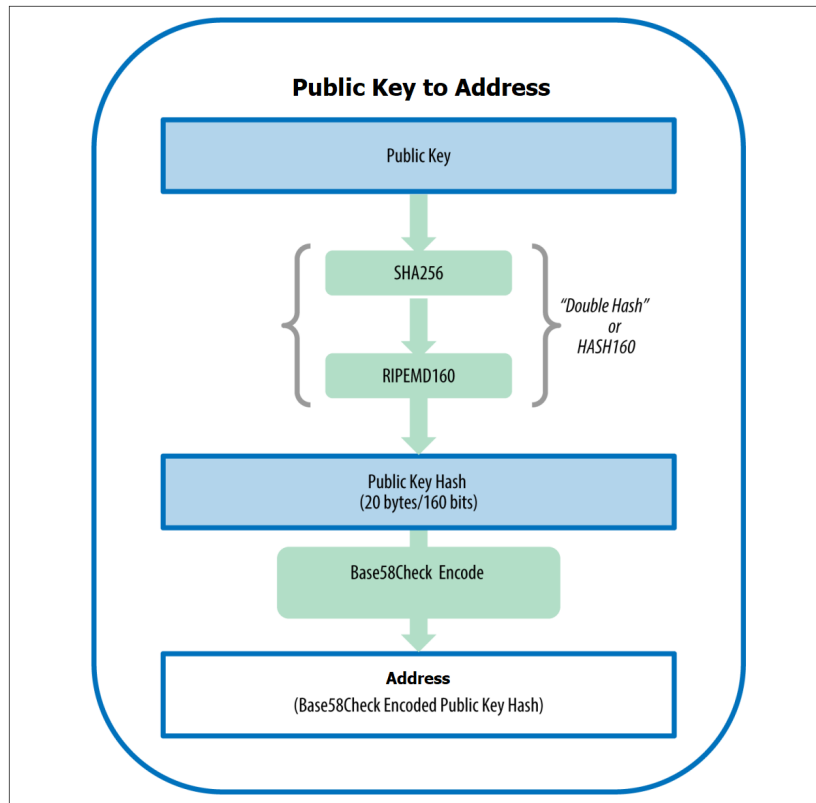


Figure 4.12: Generate Ucoin address from the public key [3]

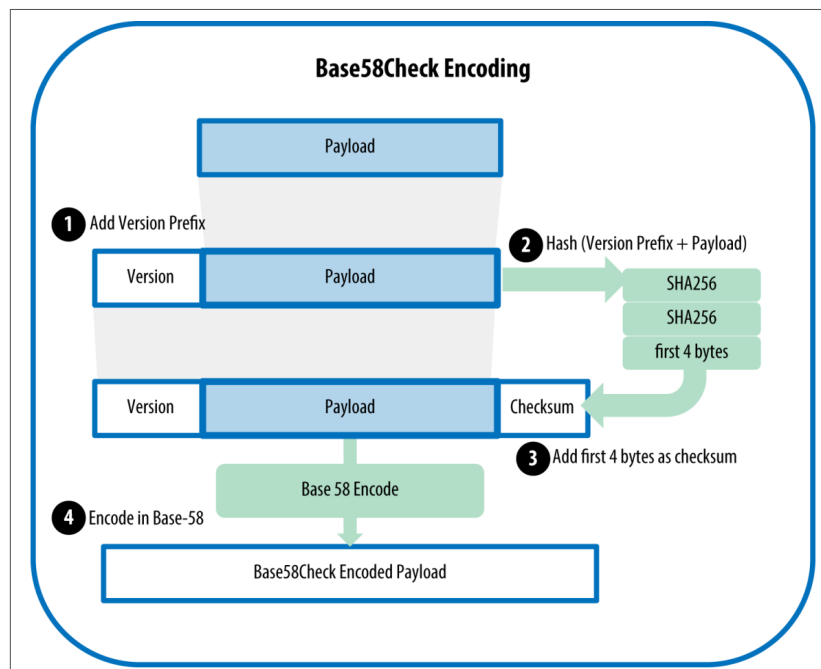


Figure 4.13: Base58Check Encoding Flow [3]

4.4.2 Start API server

The API server is built using the Flask framework and the Socket.IO library. When initialized, the server runs in a new thread and listens on the pre-configured port (3000 by default). There are four main groups of HTTP endpoints to access the local database and two events to emit updates in real-time.

1. Blocks endpoints

`/blocks:`
Get all blocks in the chain (supports pagination)

`/blocks/<int:block_height>:`
Get all details of a block by height

2. Transactions endpoints

`/transactions/<string:tx_hash>:`
Get a transaction by its hash

`/transactions/block/<int:height>:`
Get all transactions of a block based on its height (supports pagination)

`/transactions:`
Create a transaction using POST method to send details about the amount and the receiver

3. Address endpoints

`/addr/me:`
Get the node owner's address

`/addr/utxo/<string:addr>:`
Get all unspent transaction outputs belonging to an address (support pagination)

`/addr/balance/<string:addr>:`
Get the balance of an address after aggregating all UTXO amount

`/addr/history/<string:addr>:`
Get the transaction history of an address (supports pagination)

4. Web socket events

`mempool:`
Emits whenever a new transaction is added or removed from the memory pool

block:

Emits whenever a new block is added to the local blockchain

5 Normal Operations Implementation

5.1 Handling Messages

5.1.1 Overview

As previously mentioned, the routing function of each node acts as both a server and a client. The server function constantly listens and spawns a new thread to handle each incoming request, whereas the client function creates a new TCP connection whenever the node sends data to its peer. Every node's server and client functions must follow a similar protocol to communicate successfully.

Data is transmitted between nodes using a data structure called “network envelope,” which is a wrapper for messages before transmitting. A network envelope consists of a header and a payload. The header contains metadata about the message, including the network magic (a 4-byte value that identifies the start of a message and helps differentiate Ucoin messages from messages of other networks), the command (a string that specifies the type of message), and the length of the payload. The payload of the network envelope is the actual message being transmitted. It can contain various types of messages for different stages of the program, such as the peer discovery stage, the chain synchronization stage, or the transaction creation process. All messages are represented as objects in Python and can be serialized into bytes to transmit across the network and parsed back to objects in receiving nodes.

5.2 Network Protocol

version:

The first message a new node sends to existing nodes in the network as a handshake to announce its present, network address, and chain status before forming connections.

verack:

A message sent in response to a version message and serves to acknowledge the acceptance of the version message and confirm the connection between two nodes

addr:

A message contains the IP address and port number of one or more nodes the sending node is aware of. In the peer discovery stage, a node can broadcast its address in an addr message. The receiver can connect to the sender or forward the message to its peer if they are already connected.

getblocks:

A message sent to ask for missing block hashes of the blockchain from other nodes. It contains a list of the ten most recent hashes followed by representative hashes that go back in time to the genesis block. The representative hashes are spaced

out at an increasing exponential of two, where each successive block hash goes further back with the respective interval of 1, 2, 4, 8, 16, etc. blocks. This selection method provides a compact representation of the requesting node's blockchain state and allows the receiving node to quickly identify the latest common block.

inv (inventory):

A message that a node uses to announce the existence of transactions or blocks to others. Inv messages include a list of items representing a block or a transaction with an upper limitation of 500 items to avoid overloading the network. Each item consists of a type identifier and a hash. The type indicates whether the corresponding hash is of a block or a transaction. Inv can be used to respond to getblocks messages and allows the requesting node to learn about its missing blocks quickly.

getdata:

A message used to request data of blocks or transactions from other nodes. Similar to inv, the getdata message contains a list of blocks and transactions' hashes representing the data that the sender requests. Upon receiving the request, the receiving node checks its local database for the requested data and sends a response back if the data is found.

block:

A message used to transmit the entire data of a block, including the header and all transactions. A node can send block messages in response to getdata messages or broadcast them directly when it successfully mines a new block.

transaction:

A message used to transmit a single transaction, including all inputs, outputs, and the digital signature between nodes. Apart from responding to getdata messages, a node can also broadcast transaction messages directly when it generates or receives a new transaction.

5.3 Mining

5.3.1 Proof-of-work

The proof-of-work (PoW) is a mathematical/computational problem that is difficult to solve but easy to verify. The requirement is to find a block's hash less than a target value. The hash function used in Ucoin is SHA256, and the block headers are the data to be hashed. SHA256 generates uniformly distributed values. Ucoin uses two rounds of SHA256; therefore, the block header's hashes can be treated as random numbers. The probability of a random 256-bit number being smaller than a target can make finding the hash extremely difficult. For example, if a target starts with 73 zero bits, the odds of finding a solution hash is approximately 0.5^{73} or 1 in 10^{22} .

The target of the PoW problem is dynamically adjusted to ensure a consistent rate of block creation. If the blocks are generated too quickly, the target is decreased, making the problem more challenging, and vice versa. This difficulty adjustment occurs every 50 blocks to ensure a creation rate of approximately one block per minute. The formula to

adjust the target is

$$NewTarget = PreviousTarget \times \frac{ActualTime}{ExpectedTime},$$

where actual time is the time to find the previous 50 blocks and the expected time is

$$\frac{50(blocks)}{1(block/minute)} = 50(minutes)$$

Ucoin blockchain stores the target for each block in the block headers in a field called **bits**. The bits field is a structure of four bytes. The first byte represents the *Exponent*, and the last three represent the *Coefficient*. The formula to convert from **bits** to *Target* is

$$Target = Coefficient \times 256^{Exponent-3}$$

Although storing targets as bits reduces their accuracy, it is negligible compared to the amount of conserved bandwidth and storage space when using this compact form.

5.3.2 Mining Process

All new transactions that the node creates or receives are stored in the mempool. The mempool, also known as the memory pool or transaction pool, is a temporary storage of each node, where unconfirmed transactions (i.e., transactions that have not been added to the blockchain) are kept.

Once the chain synchronization process completes, a node can start mining for new blocks. The mining process begins with creating a candidate block. Candidate blocks are blocks created by miners that have yet to satisfy the PoW problem. The process of mining a block is constantly modifying the candidate block's header to find a valid hash value.

Creating a candidate block requires the previous block's hash, the transactions inside the mempool, and the PoW target bits. The previous hash is obtainable from the top block of the local chain, whereas the bits can be derived from the next block's height. Afterward, the miner must filter the mempool to remove transactions that have already been included in the chain. The remaining transactions are copied and prepended with a coinbase transaction (which sends block rewards to the miner) before being added to the candidate block.

The Merkle root of all the added transactions must be computed to complete the candidate block. As visualized in figure 5.1, a Merkle root is obtained by hashing all transactions individually, dividing the results into pairs to concatenate and performing another hash (duplicating the last hash if there is an odd number of hashes). This pairing and hashing process continues until one single hash value remains. Once the candidate block is ready, miners can mine the block by changing the nonce. The nonce is a field inside a candidate block's header and can contain any arbitrary integer. Miners keep incrementing the nonce, which changes the candidate block's hash until they find a hash that meets the current target's criteria. When a miner finds a valid hash, it adds the

block to the local chain, broadcasts a block message, and immediately restarts mining for the next block.

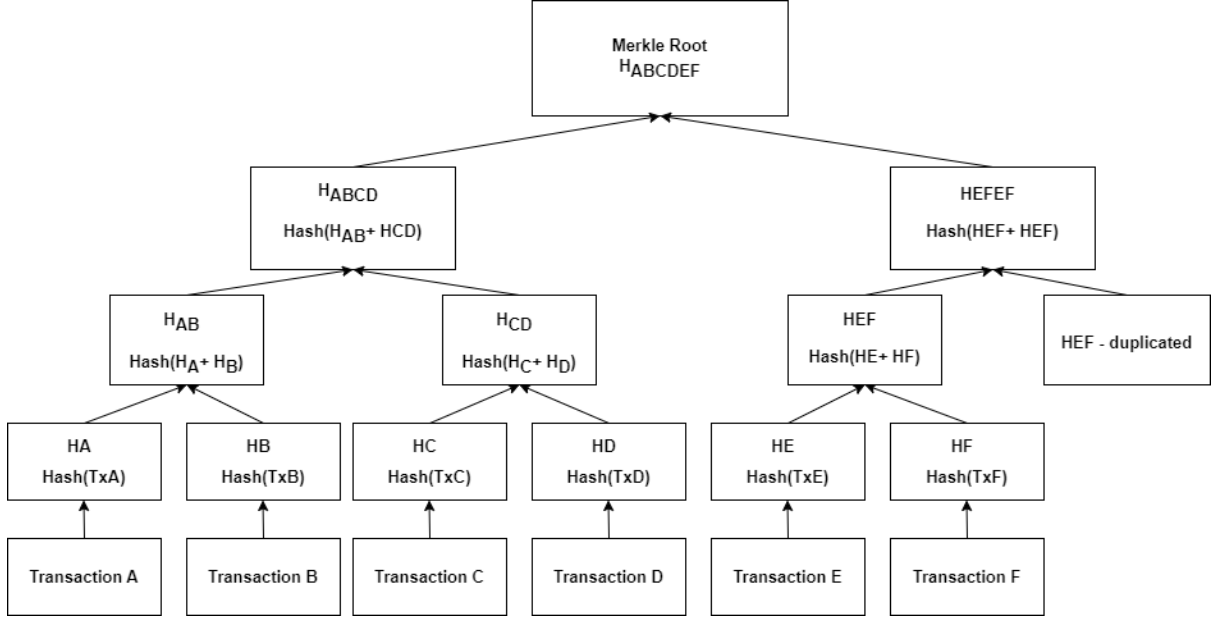


Figure 5.1: Example Merkle tree and Merkle root computation process

5.3.3 Block Validation

Whenever a node receives a block message from a peer, its blockchain thread will validate the block before adding it to the local blockchain. If the received block fails to meet any criteria listed in figure 5.2, it will be dropped immediately.

First of all, the node query the received block's hash in its local database to search for any existing block. Only blocks not inserted into the local chain can proceed to the next step. Then the blockchain thread checks for the PoW by recalculating the block header and comparing it with the required target. Subsequently, the node verifies that the first transaction of the block is a coinbase transaction and that the received Merkle root is indeed the Merkle root of all transactions in the block. Afterward, apart from the first one, each transaction is tested individually to ensure that none are coinbase transactions and all are valid transactions.

At this point, the node will consider the parent block or the previous block of the one received. If the previous hash is not in the local blockchain, the block is an “orphan block” and is moved to an orphan pool in the memory instead of the database to wait for its parent block. Once the parent block is inserted into the database, the orphan block will also be added automatically.

If the block is not an orphan block, its height will be deducted based on its parent's height. The node can use this height to calculate the required block bits, and if the bits inside the header do not match this value, the block will be invalidated.

Finally, the chain can add the received block into the database as well as its offspring

(if any) in the orphan pool. The block can still be inserted even if there is already another block with the same height in the local chain. This rare situation is called a fork and occurs when two miners solve the PoW problem nearly simultaneously. This fork can create a secondary chain, but only the longest chain is valid as it requires more PoW. If the added block is the top block, the node will continue to broadcast it and notify the miner thread to restart and mine for the next block.

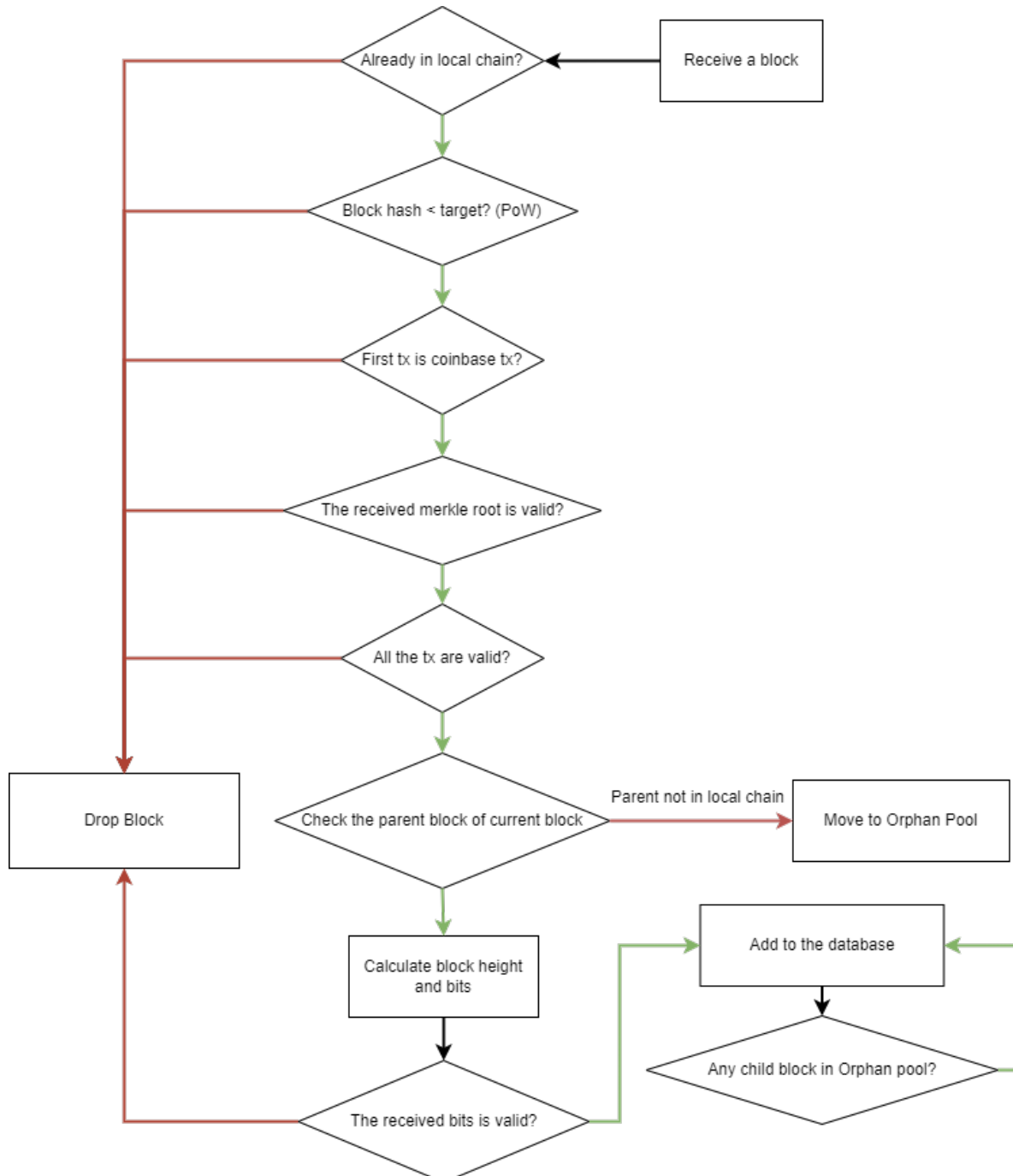


Figure 5.2: Block validation flow

5.4 Handling Transactions

5.4.1 Transaction Creation

Node owners can create transactions by sending POST requests containing the amount and address of the receiver to the local API. The transaction creation process is demonstrated figure 5.3. Firstly, the node checks whether the receiver address has the correct Base58check format to avoid sending coins to the wrong address. Then it queries the local blockchain for all UTXOs belonging to the address owner. From this set of UTXOs, the node will select UTXOs to spend using a FIFO algorithm. If the total value of the UTXO set is less than the amount, the program will stop the process and notify the user about the insufficient balance. Otherwise, the selected UTXO will be used to craft the list of transaction inputs. Concerning the transaction outputs, the node will generate a TxOut with the receiver address and the specified amount of coins. If the selected UTXOs' total value exceeds the amount, another TxOut will be created to send the change back to the sender's address. Lastly, the node will sign the transaction with the owner's private key and broadcast a transaction message to its peers.

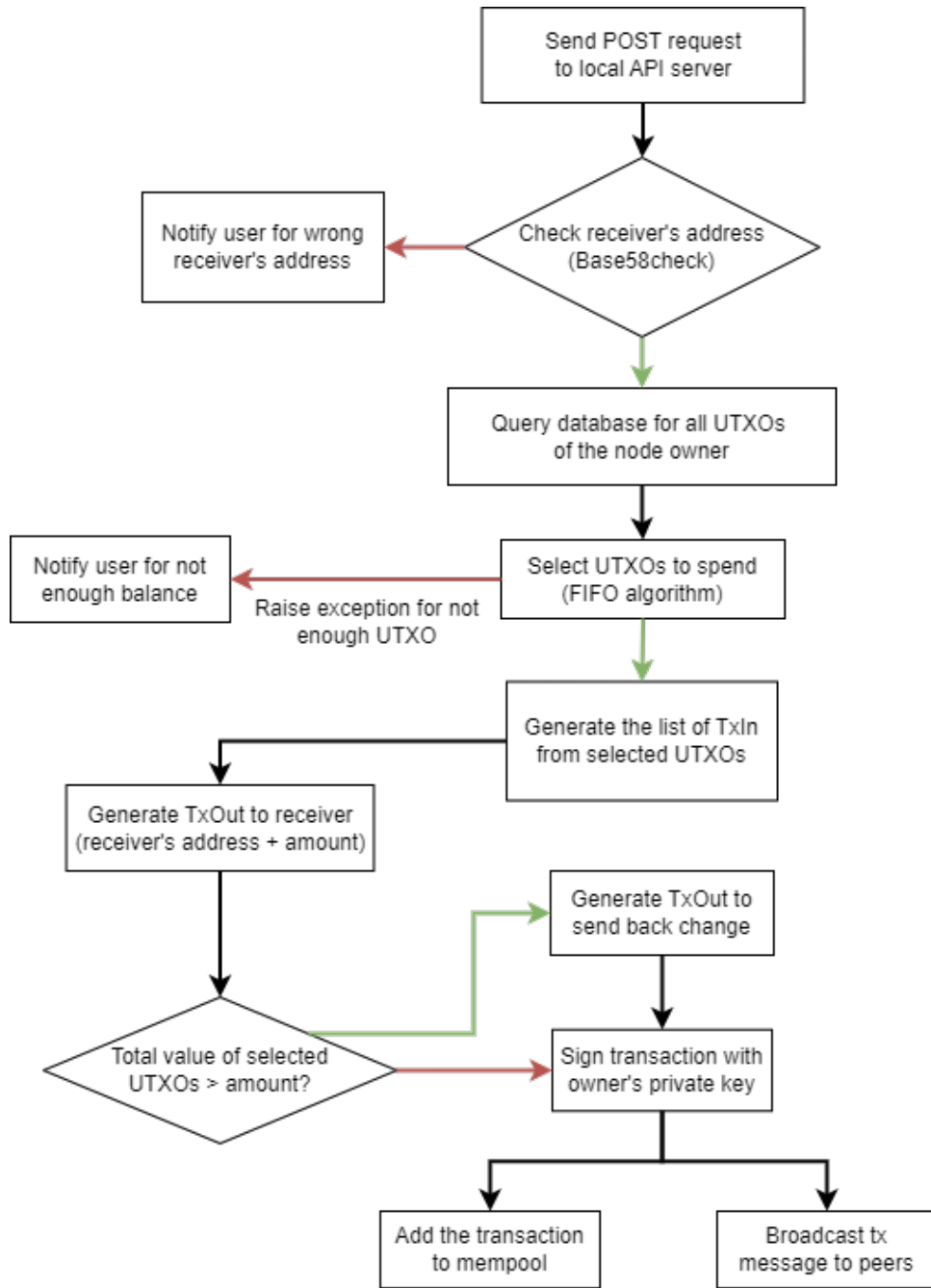


Figure 5.3: Transaction creation flow

5.4.2 Transaction Validation

Similar to block messages, transaction messages must be validated before further processing and are dropped once it fails any validation step. Firstly, the transaction must not be a coinbase transaction as coinbase transactions cannot be broadcasted. Secondly, it must not already exist in the mempool or the local blockchain. Thirdly, the inputs and outputs fields of the transaction must not be empty. More importantly, the total coins used in transaction inputs must be greater than or equal to the total coins spent in transaction outputs. In other word, total input coins must not be smaller than the total output coins. Then the program iterates through all transaction inputs to verify that

the referenced TxOut is in the UTXO set (i.e., an unspent transaction output). Finally, the program combines the unlocking script in each TxIn with the corresponding locking script in the referenced TxOut. It evaluates the result to ensure the provided signature is valid and the sender is indeed the owner of these UTXOs. If a transaction satisfies all the mentioned requirements, it is broadcast to other peers and added to the local mempool waiting to be added to a candidate block and then the blockchain. The transaction is confirmed when it is in the blockchain; therefore, it may take some time to transfer coins successfully.

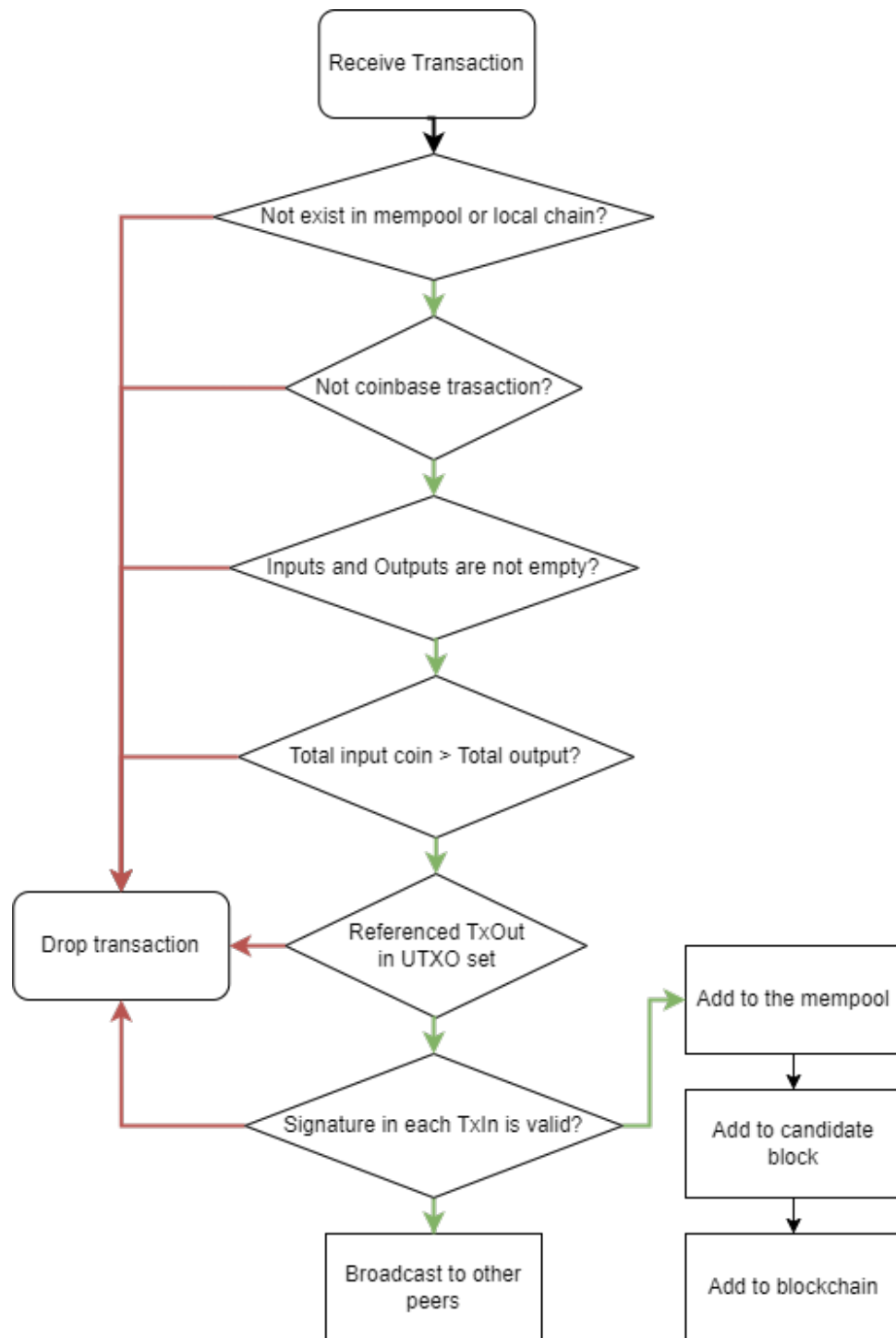


Figure 5.4: Transaction validation flow

6 Results

6.1 Scenario Design

The Ucoin program is packed into a Docker image to make it easier to develop and deploy. Docker is a technology that packages applications and their dependencies into a single container that can run on any system, making it easier to run the application in different environments.

To test the Ucoin network, we set up five Docker containers running the same Ucoin program on one machine. These containers can be seen in figure 6.2. Each container will act as a node in the Ucoin network and has the full functionality of routing messages, managing blockchain, mining blocks, and serving API. One container called “unode” has its port and volumes mapped to our machine, allowing us to control this node directly. The other four containers run automatically to simulate our peers in the network. Figure 6.1 provides a visualization of the Ucoin network design. In addition, we also developed a web interface running locally and directly connecting to the API backend to make it more user-friendly to interact with the “unode” container and monitor the changes in our blockchain.

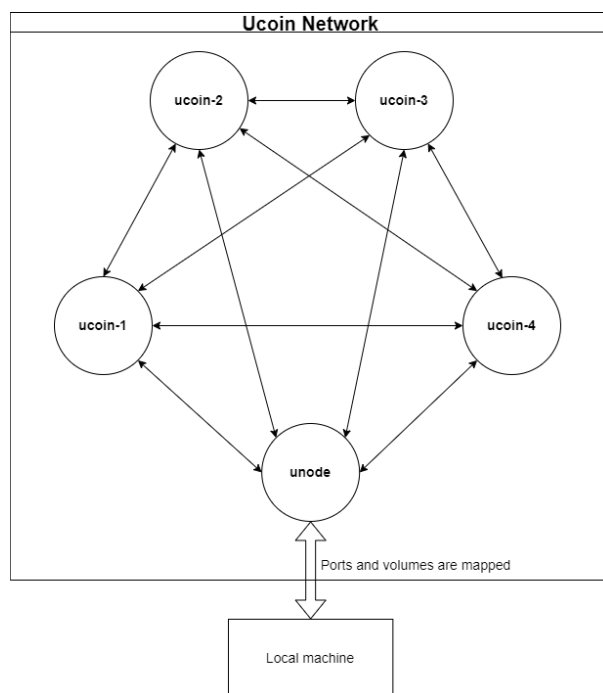


Figure 6.1: Ucoin Network Architecture of the experiment















 bc-ui b37ebf25f631 	ucoin-web_ui:latest	Running	3001:3000 
 ucoin-1 2a59b1351010 	ucoin:latest	Running	
 ucoin-2 eab9fe4a16cb 	ucoin:latest	Running	
 ucoin-3 8beb2d2dd588 	ucoin:latest	Running	
 ucoin-4 d8ba58c8c1dd 	ucoin:latest	Running	
 unode 804f4afafa6b 	ucoin:latest	Running	3000:3000 

Figure 6.2: Docker containers for the experiments

The experiment scenario is creating a transaction from the “unode” container. We can then observe it being added to the mempool and broadcast across the network. All nodes will then add this transaction to their own candidate block, and the fastest node to mine this block will manage to add our transaction to the blockchain. We will be able to follow this process using the web interface as well as the logs of the API.

The web interface is also built into a docker image. This web interface will run as a container mapped on port 3001 allowing users to access the UI at <http://localhost:3001>. Meanwhile, unode container serve the API for the blockchain on port 3000. The web interface and unode containers will communicate by transmitting raw json data through port 3000. Users can also interact with unode through localhost port 3000; however, the web UI provides much more convenience.

6.2 Simulation Result

After building the `ucoin` image and running the 5 specified containers as mentioned in figure 6.2, we manage to reach the web interface container at <http://localhost:3001> and can see all the blocks on the chain being updated in real time

⋮

Blocks

👤

Me

☰

Mempool

👥

Peers

Blocks

Height	Hash	Miner	Nonce	Timestamp	Tx count
#2	000034e157f36b5c3db0578176bc30cfc5fa3a1b17d50be9c543856d20068c5a	3cr6qX5pQLRqxMNA3z5gXLq5PgNkqZUCS	810371	06/02/2023 05:03:39	1
#1	0000932c41f6ca6d4be802d073f813280db9d7978548d152f854ab0a8126c3ae	3cr6qX5pQLRqxMNA3z5gXLq5PgNkqZUCS	570362	06/02/2023 05:03:24	1
#0	0000f6d888728e422d6cfd01ab1c525643f43e64572f4872dba5dcb1ff64e481	3nIE5xEaHsHjKDFRaQu3wEssD7vYNwmED	892255	06/02/2023 05:02:48	1

Total 3 rows, 1 ~ 3 rows

< 1 >

Figure 6.3: Blocks Page Interface

We can click on any block on the **Blocks** page as shown in figure 6.3 to view its details

Blocks

Me

Mempool

Peers

Detail of Block #5

Overview

Time	Nonce	Bits	Miner
06/02/2023 05:04:50	140140	0x1f0fffff	McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN

Block hash

0007b6c609ca6a3a8d3f5a0d306101d8b679d48f401ecf3a888bf46e006d5357

Merkel Root

a7cf4785976ac7f34109ccd160c07cd2a7a3dc39bafbf18459587f944684d835

Previous Block Hash

[0009f5c7266d2bb034965f6ed9e4aab1ad73e96830030e58a2eee3f7ac02a088](#)

Transactions

Transaction #1 - (a7cf...d835)

Figure 6.4: Details of block number 2

We can also click on the Miner to see the information about the address that mined this block in figure 6.5

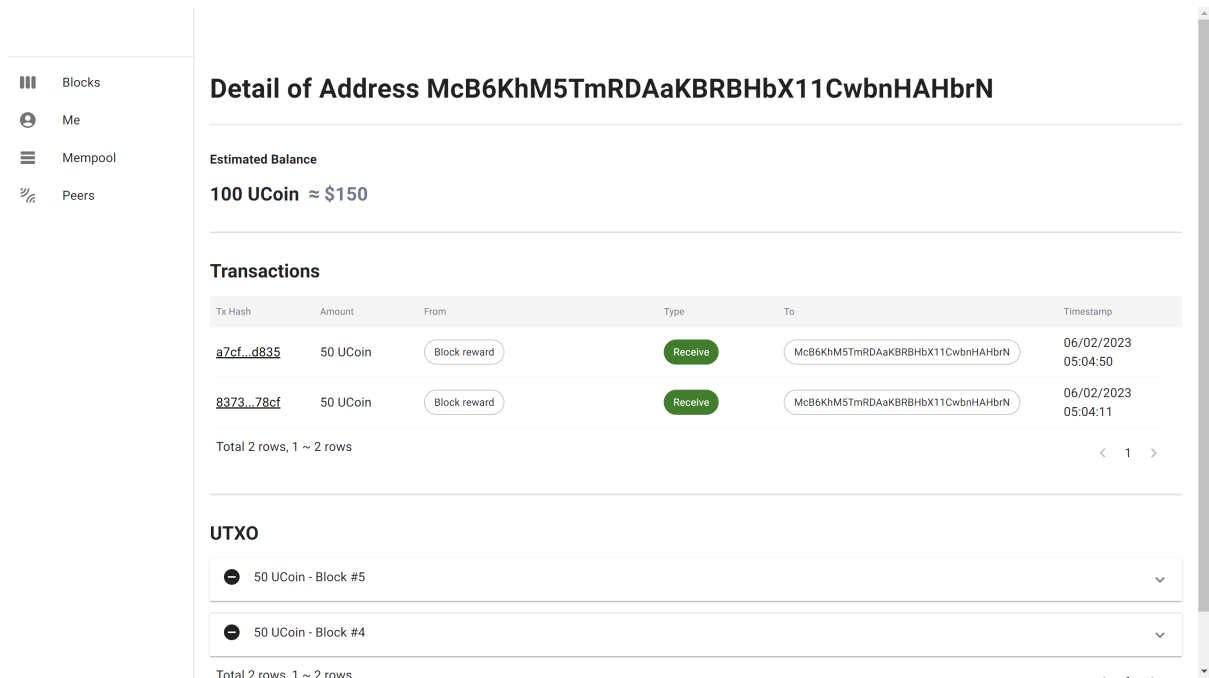


Figure 6.5: Information about address a particular Ucoin address

Before creating a transaction, we navigate to **Peers** page to verify that we are connected to our peers

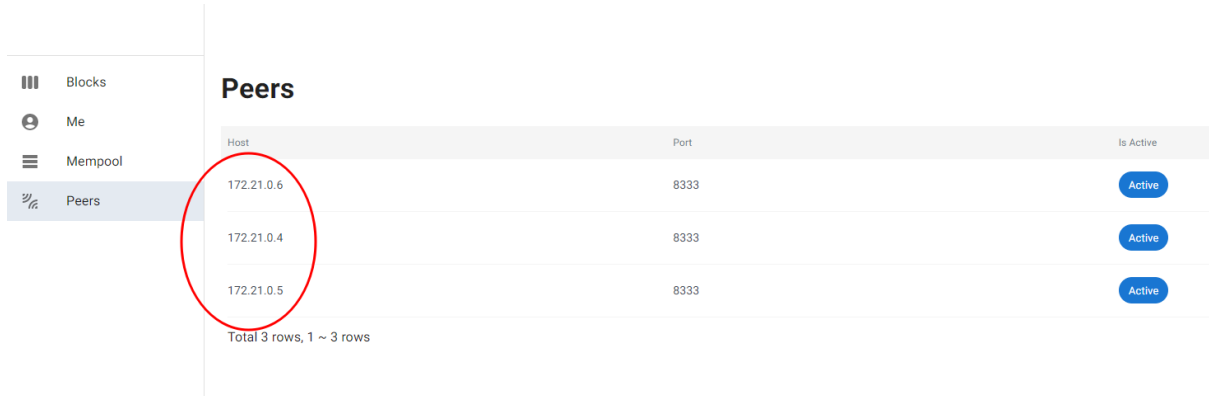


Figure 6.6: Peer connections status in Peers page

Next we navigate to the **Me** page to view our current balance, including our transactions history and UTXOs. As can be seen in figure 6.7, we have successfully mined three blocks and received 3 UTXO of 50 coins as block rewards.

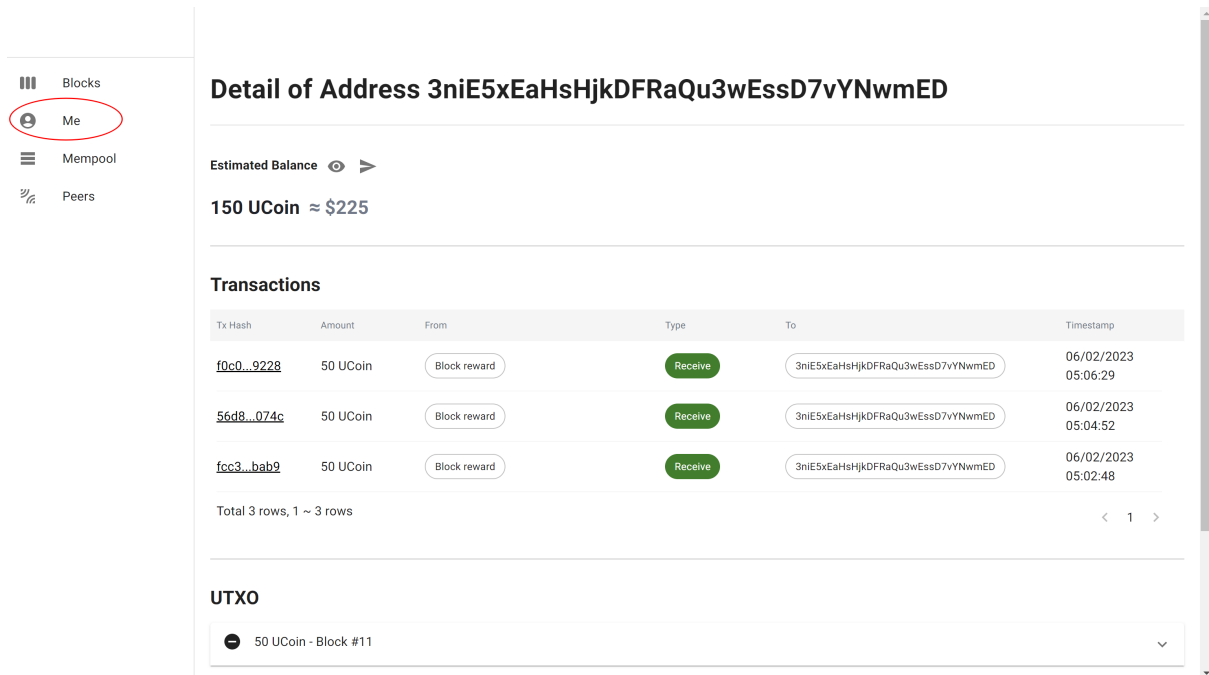


Figure 6.7: Our node's balance, transaction history, and UTXOs

A difference between the **Me** page in figure 6.7 with the profile of any other address in figure 6.5 is that the **Me** page also allows us to send coins to others. We can press on the button in figure 6.8, to try this feature.

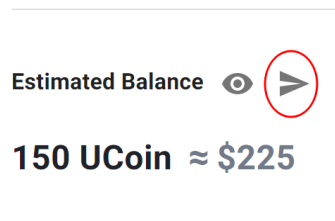


Figure 6.8: Button to create transaction

A prompt will pop up allowing us to type in the amount and the receiver's address to send money to. The address and the amount will be checked by "unode" before sending the coins to avoid any mistakes. The mistake will be logged and notified by to users as we can see in figure 6.9 and 6.10.

```
[DEBUG] - [Candidate transactions: 6865] - [Miner]
[DEBUG] - [Candidate bits: 0x1f0ffff] - [Miner]
[INFO] - [nonce: 100000] - [Miner]
[WARNING] - [Invalid address: McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrA]
[INFO] - [Recv: b'block' from 172.21.0.4] - [network.Network]
[INFO] - [Recv: b'block' from 172.21.0.5] - [network.Network]
[INFO] - [Recv: b'block' from 172.21.0.6] - [network.Network]
```

(a) Warning in logs

Transfer Money

Internal transfers are free on UCoin Network

Amount

Receiver

Test a wrong address, the correct letter is "N"

Error: Invalid address

SEND

(b) Warning by Web UI

Figure 6.9: Warning for sending to an incorrect address

Estimated Balance

200 UCoin \approx \$300

Transactions

Tx Hash	Amount	Timestamp
2069...6cc8	50 UCoin	06/02/2023 05:07:07
f0c0...9228	50 UCoin	06/02/2023 05:06:29
56d8...074c	50 UCoin	06/02/2023 05:04:52
fcc3...bab9	50 UCoin	06/02/2023 05:02:48

Total 4 rows, 1 ~ 4 rows

Transfer Money

Internal transfers are free on UCoin Network

Amount

Receiver

Error: Not enough coin. Current balance: 200

SEND

Figure 6.10: Warning for not enough coins to send

Once we create a transaction with the valid amount and address, we can see the transaction is created, broadcasted and added to the mempool in the log of the “unode” container as shown in figure 6.11. We can also navigate to “Mempool” page and verify that the transaction is in the mempool like in figure 6.12.

```
[DEBUG] - [Created transaction: 3fea] - [api.transaction_endpoints]
[INFO] - [Client 6xn4mfJOQa8b0JH1AAAD connected] - [api]
[INFO] - [Broadcasting b'transaction', excludes: [None]] - [network.Network]
[INFO] - [Sent: b'transaction' to 172.21.0.6] - [network.Peer]
[INFO] - [Sent: b'transaction' to 172.21.0.4] - [network.Peer]
[INFO] - [Sent: b'transaction' to 172.21.0.5] - [network.Peer]
[DEBUG] - [Added new tx to mempool: 3fea] - [Miner]
[INFO] - [nonce: 550500] - [Miner]
[INFO] - [nonce: 551000] - [Miner]
[INFO] - [nonce: 551500] - [Miner]
[INFO] - [nonce: 552000] - [Miner]
```

Figure 6.11: Logs of transaction creation process

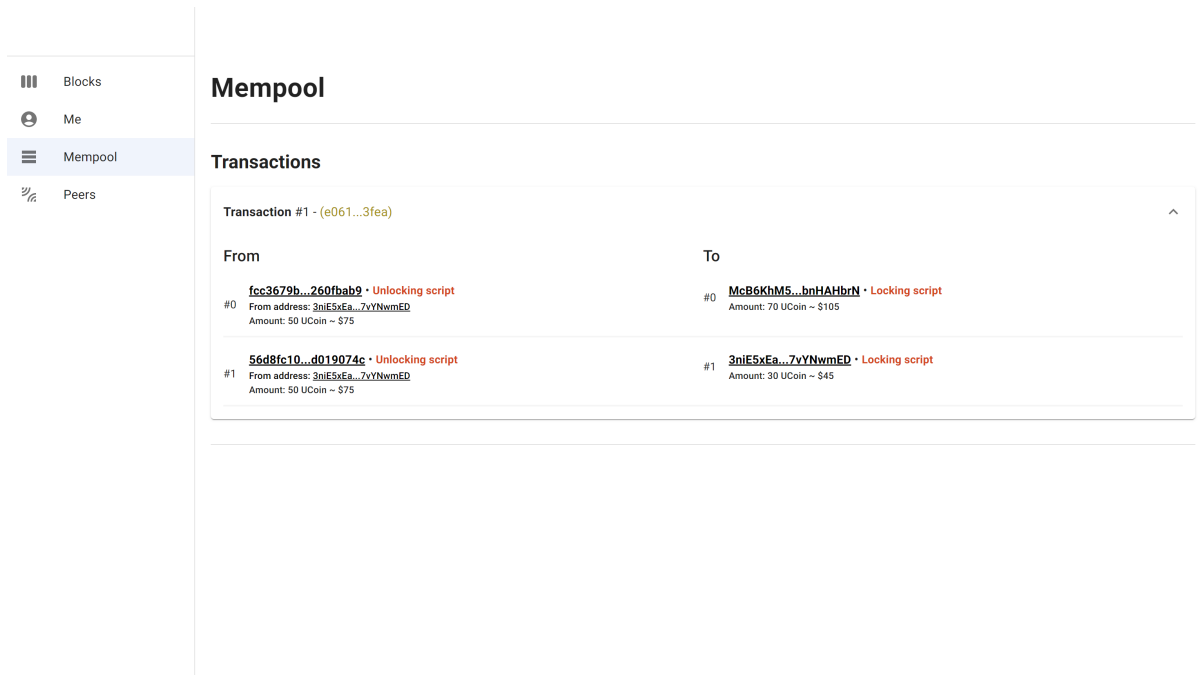


Figure 6.12: Transaction added to mempool

After a while, the mempool automatically updates once again and this time the transaction is removed, indicating that the transaction has successfully been added to the blockchain. The moment when mempool updated is captured in figure 6.13. In the log in figure 6.14, we can also see that a block numbered 36 has been inserted into the chain and our transaction has been removed from mempool.

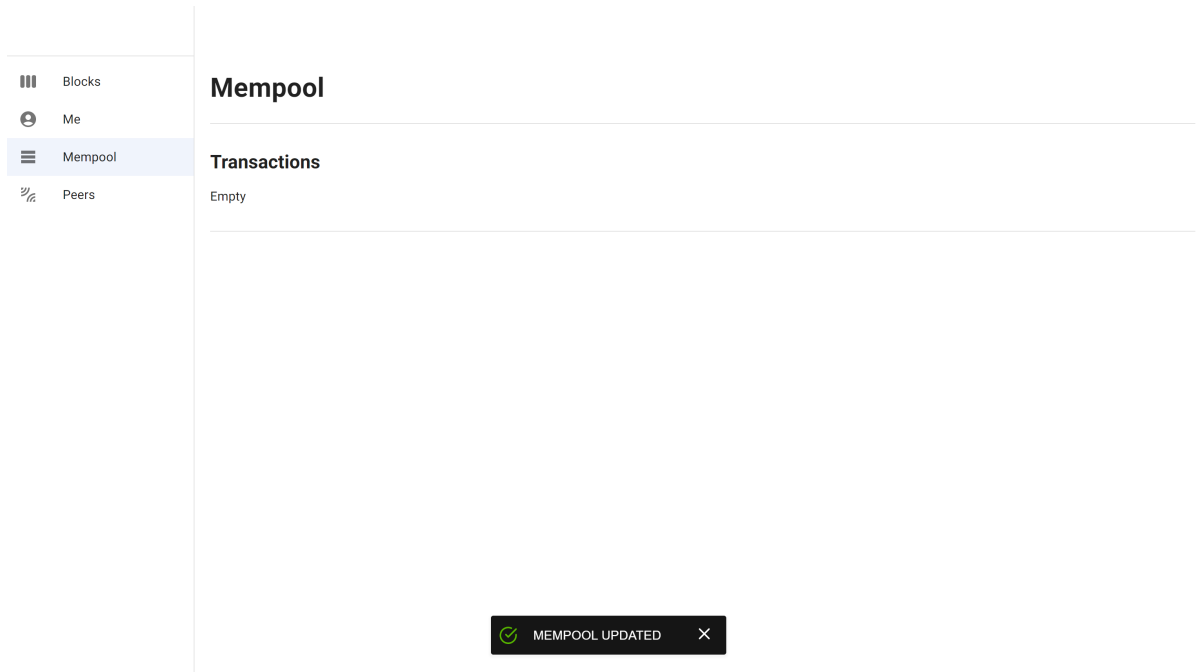


Figure 6.13: Transaction is automatically removed from mempool

```

[INFO] - [nonce: 840000] - [Miner]
[DE 1] [Block eaa7 inserted at height 36] - [blockchain.Blockchain]
[INFO] - [Broadcasting b'block', excludes: [None]] - [network.Network]
[INFO] - [Sent: b'block' to 172.21.0.6] - [network.Peer]
[INFO] - [Sent: b'block' to 172.21.0.4] - [network.Peer]
[INFO] - [Sent: b'block' to 172.21.0.5] - [network.Peer]
[DE 2] [Removed tx from mempool: 3fea] - [Miner]

```

Figure 6.14: Logs of mempool updates

Since every block has at least 1 transaction as the coinbase transaction, we can see that block 36 in figure 6.15 has an extra transaction which is the one we created.

#40	0002f2554d053cca207b0769f30867781ac1037974b1a2f363ae99c904fb1ebf	McB6KhM5TmRDAaKBRBhbX11CwbnHAHbrN	860349	06/02/2023 05:14:47	1
#39	0006ef1c30b067009062e2bc43e1aab005530fdd4b9d242234088dfa0d2dc30b	9eCRm4vVwcUSR845WgJShvDuZb1GB4yNu	481424	06/02/2023 05:14:22	1
#38	000cdae47d9b694779efd2aa4c3043cc77b88c337bd551e246fb4bb39cce3f2b	9eCRm4vVwcUSR845WgJShvDuZb1GB4yNu	580681	06/02/2023 05:14:11	1
#37	00070706781fd019508a246e81fcd9cd1d01756939a74ad0cd8d91b828aa0053	3kdyYlgbRd8cSmhFMZQtSh6rBscTEFvPy	761007	06/02/2023 05:13:53	1
#36	000e8f2af944d9efba48cfc7fb3688851c252f24b456b1912a306f689ae7eaa7	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED	530944	06/02/2023 05:13:37	2
#35	000ea921793d55fb699d98c765fdea7b4822cea1f3d0138f0b244bf3e670a2b4	3cr6qX5pQLRqxMNA3z5gXLq5PgNkqZUCS	332098	06/02/2023 05:13:01	1
#34	000bfa8078b3cb9b68df7a68e59f9619216a89f6bba13465392643544d4e410f	9eCRm4vVwcUSR845WgJShvDuZb1GB4yNu	323089	06/02/2023 05:12:08	1
Total 54 rows, 11 ~ 20 rows					< 2 >

(a) Block with 2 transaction

Blocks

Me

Mempool

Peers

Detail of Block #36

Overview

Time	Nonce	Bits	Miner
06/02/2023 05:13:37	530944	0x1f0fffff	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED

Block hash
000e8f2af944d9efba48cfc7fb3688851c252f24b456b1912a306f689ae7eaa7

Merkel Root
24a2a61045349ca47045a7e0a4a0c294ccbc155c93ed45615e3dbe0036268200

Previous Block Hash
[000ea921793d55fb699d98c765fdea7b4822cea1f3d0138f0b244bf3e670a2b4](#)

Transactions

Transaction #1 - (71d7...7dd0)



Transaction #2 - (e061...3fea)

(b) Coinbase and user created transaction

Figure 6.15: The block containing the created transaction

We can check the transaction history of our address and the receiver's address in figure 6.16 to see our transaction. Besides the obvious 70 coins, we can also see that we received 30 coin from our address, which is the change of this transaction. Additionally, based on the transaction details in figure 6.17, 2 UTXOs of 50 coins were used as transaction inputs and we can press on each UTXO to verify their origins.

Detail of Address 3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED

Estimated Balance  

330 UCoin ≈ \$495

Transactions

Tx Hash	Amount	From	Type	To
71d7...7dd0	50 UCoin	Block reward	Receive	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED
e061...3fea	30 UCoin	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED	Receive	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED
e061...3fea	70 UCoin	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED	Spend	McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN

(a) Our address

Detail of Address McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN

Estimated Balance

470 UCoin ≈ \$705

Transactions

Tx Hash	Amount	From	Type	To
b130...5af1	50 UCoin	Block reward	Receive	McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN
e061...3fea	70 UCoin	3niE5xEaHsHjkDFRaQu3wEssD7vYNwmED	Receive	McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN
119a...c570	50 UCoin	Block reward	Receive	McB6KhM5TmRDAaKBRBHbX11CwbnHAHbrN

(b) The receiver's address

Figure 6.16: Transaction history

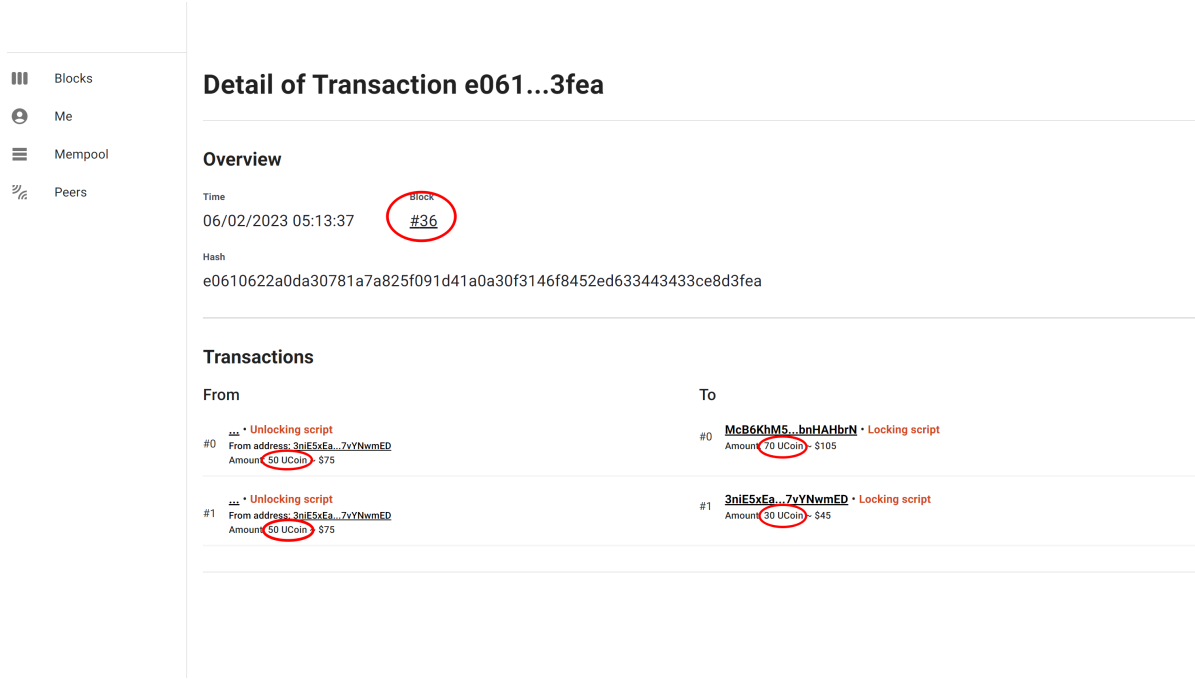


Figure 6.17: The successfully created transaction

6.3 Mining Rate Result

As previously mentioned, the target of the PoW problem is dynamically adjusted every 50 blocks to ensure blocks are generated at a rate of **1 block per minute (bpm)**. If the mining rates of new blocks is slower than 1bpm, the problem should be easier and if the rates is higher than 1bpm, the problem should be harder. The difficulty of the PoW problem during a period can be evaluated using a *Difficulty* value, which is computed using the formula:

$$Difficulty_i = \frac{target_0}{target_i},$$

where:

- $Difficulty_i$ is the *Difficulty* at period i
- $target_i$ is the *target* at period i
- $target_0$ is the initial target chosen before starting the network. (By default, the initial target is calculated from the bits value `0x1f0fffff`)

This formula reflects the fact that the smaller the target, the more difficult it is to find a solution for the PoW problems. Or in other word, the higher the difficulty.

We collected have analyzed 1000 blocks from our simulation to examine the ability to stabilize the mining rate of the Ucoin network. The data is recorded in table 5. From the data as well as the visualization in figure 6.18 and 6.19, we can see that the mining rate is stabilized around 1bpm despite minor fluctuations.

Table 5: Difficulty and Mining Rate Summary

Blocks	Difficulty	Mining Time (minutes)	Mining Rate (blocks per minute)
0-49	1.00	11.37	4.40
50-99	4.55	45.82	1.09
100-149	5.17	52.47	0.95
150-199	5.06	48.00	1.04
200-249	5.39	65.50	0.76
250-299	4.21	52.27	0.96
300-349	4.05	45.08	1.11
350-399	4.82	55.90	0.89
400-449	4.38	43.62	1.15
450-499	5.21	50.23	1.00
500-549	5.43	58.40	0.86
550-599	4.68	47.78	1.05
600-649	4.98	63.50	0.79
650-699	3.95	55.53	0.90
700-749	3.66	44.67	1.12
750-799	4.36	58.63	0.85
800-849	3.89	39.72	1.26
850-899	5.12	57.98	0.86
900-949	4.57	52.27	0.96
950-999	4.57	48.70	1.03

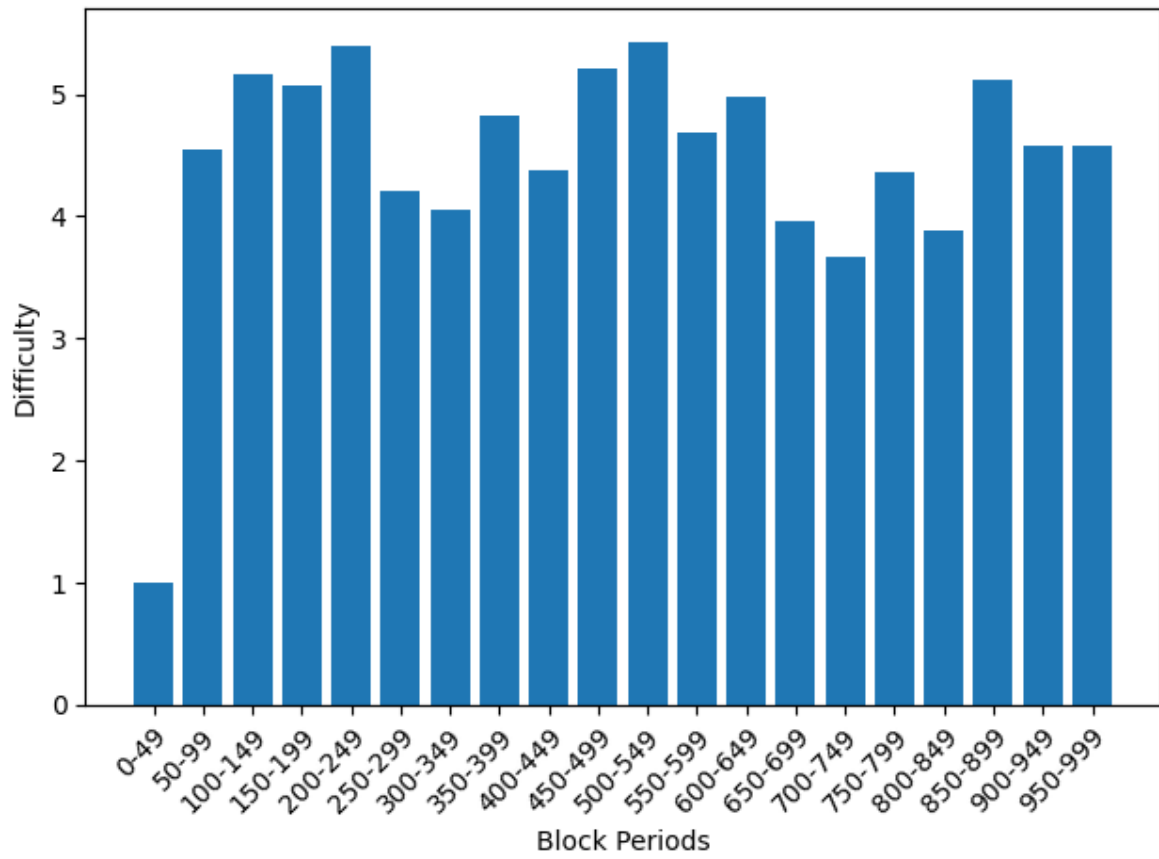


Figure 6.18: Difficulty of the first 20 periods (50 blocks each)

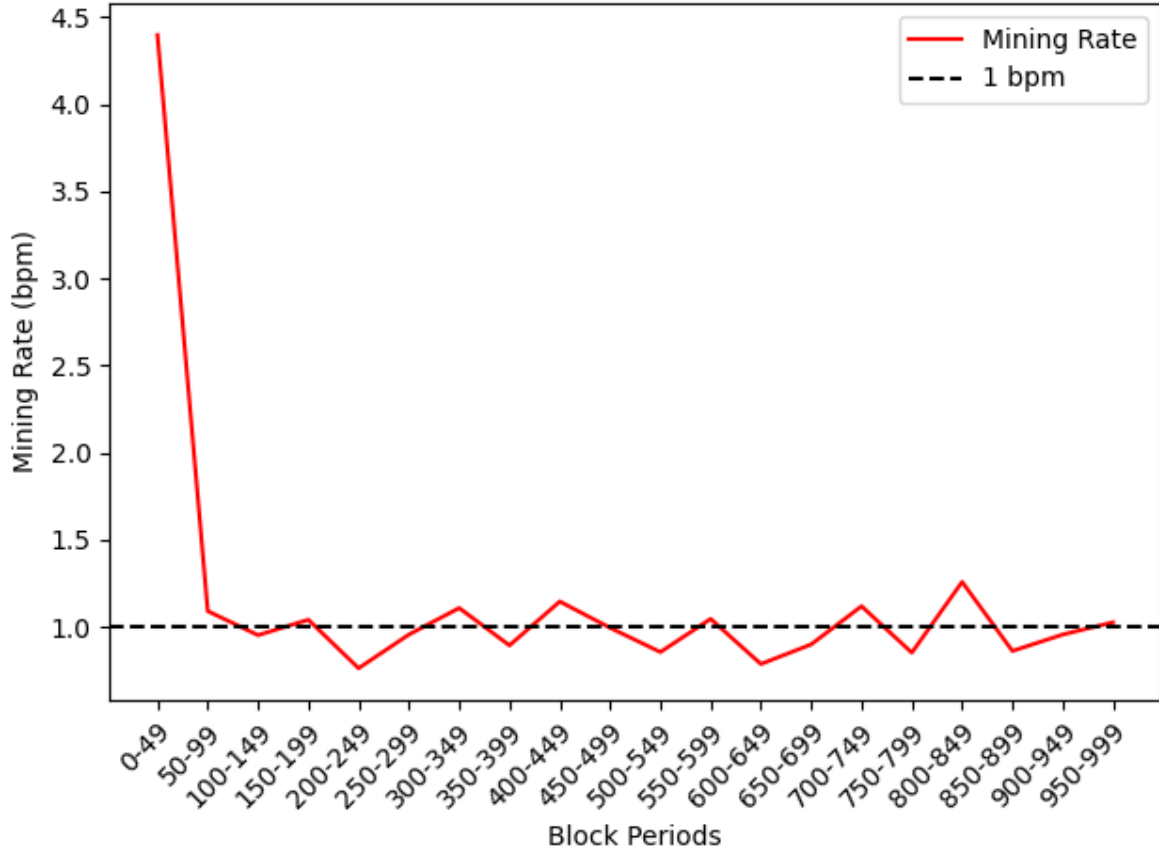


Figure 6.19: Mining rate of the first 20 periods (50 blocks each)

7 Conclusion

In this project, Ucoin - a simplified blockchain-based application - is reproduced using Python programming language with a local SQLite database. The Ucoin blockchain network is successfully simulated using Docker containers functioning as different nodes in an interconnected P2P network. Each node allows users to join the network, create new transactions and interact with the blockchain by serving four core functionalities in parallel as previously designed. In addition, our web-based UI, which is built using TypeScript and React, can connect directly to the API served by Ucoin node and improve the user experience of our program. We managed to simulate the blockchain to generate hundreds of blocks and collected data to analyze the performance of the difficult adjustment aspect of our network.

Our program is documented in this report, and the source code is hosted as an open-source project on our Github repository. Therefore, this program can be applied in future projects researching blockchain technology's potential applications as well as vulnerabilities.

Bibliography

- [1] Anastasiia Lastovetska. Blockchain architecture basics: Components, structure, benefits & creation — mlsdev. URL <https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture>.
- [2] Hossein Abbaspour. Potential benefits of blockchain technology for mining industry: with a case study of truck dispatching system in open pit mines. 06 2018.
- [3] Andreas M. Antonopoulos. Mastering bitcoin: Programming the open blockchain. chapter Keys, Addresses. O'Reilly, Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo, 2018.
- [4] Robert Sheldon. A timeline and history of blockchain technology., 08 2021. URL <https://www.techtarget.com/whatis/feature/A-timeline-and-history-of-blockchain-technology>.
- [5] Wikipedia contributors. Blockchain — Wikipedia, the free encyclopedia, 2023. URL <https://en.wikipedia.org/w/index.php?title=Blockchain&oldid=1137539840>.
- [6] Synopsys. What is blockchain and how does it work? URL <https://www.synopsys.com/glossary/what-is-blockchain.html>.
- [7] FasterCapital. How blockchain is revolutionizing the future of fintech. URL <https://fastercapital.com/content/How-Blockchain-is-Revolutionizing-the-Future-of-Fintech.html>.
- [8] Adam Hayes. Smart home: Definition, how they work, pros and cons, 12 2022. URL <https://www.techtarget.com/whatis/feature/A-timeline-and-history-of-blockchain-technology>.
- [9] Muhammad Usman and Usman Qamar. Secure electronic medical records storage and sharing using blockchain technology. *Procedia Computer Science*, 174:321–327, 2020. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2020.06.093>. URL <https://www.sciencedirect.com/science/article/pii/S1877050920316136>. 2019 International Conference on Identification, Information and Knowledge in the Internet of Things.
- [10] Sam Daley. 34 blockchain applications and real-world use cases, 08 2022. URL <https://builtin.com/blockchain/blockchain-applications>.
- [11] Hugo Cespedes A. Blockchain and its applications today, 06 2022. URL https://www.linkedin.com/pulse/blockchain-its-applications-today-hugo-cespedes-a-?trk=pulse-article_more-articles_related-content-card.
- [12] Rodolfo Bianchi Guimarães. Music streaming meets blockchain: How tokens could be implemented?, 03 2019. URL <https://www.linkedin.com/pulse/music-streaming-meets-blockchain-how-tokens-could-bianchi-guimar%C3%A3es>.
- [13] Andreas M. Antonopoulos. Mastering bitcoin: Programming the open blockchain.

chapter Transactions. O'Reilly, Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo, 2018.

- [14] Jimmy Song. Programming bitcoin: Learn how to program bitcoin from scratch. chapter Transactions. O'Reilly, Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo.