MiniCoin User Manual

Nicholas Huppert - s3729119

Edward Rossi - s3449334

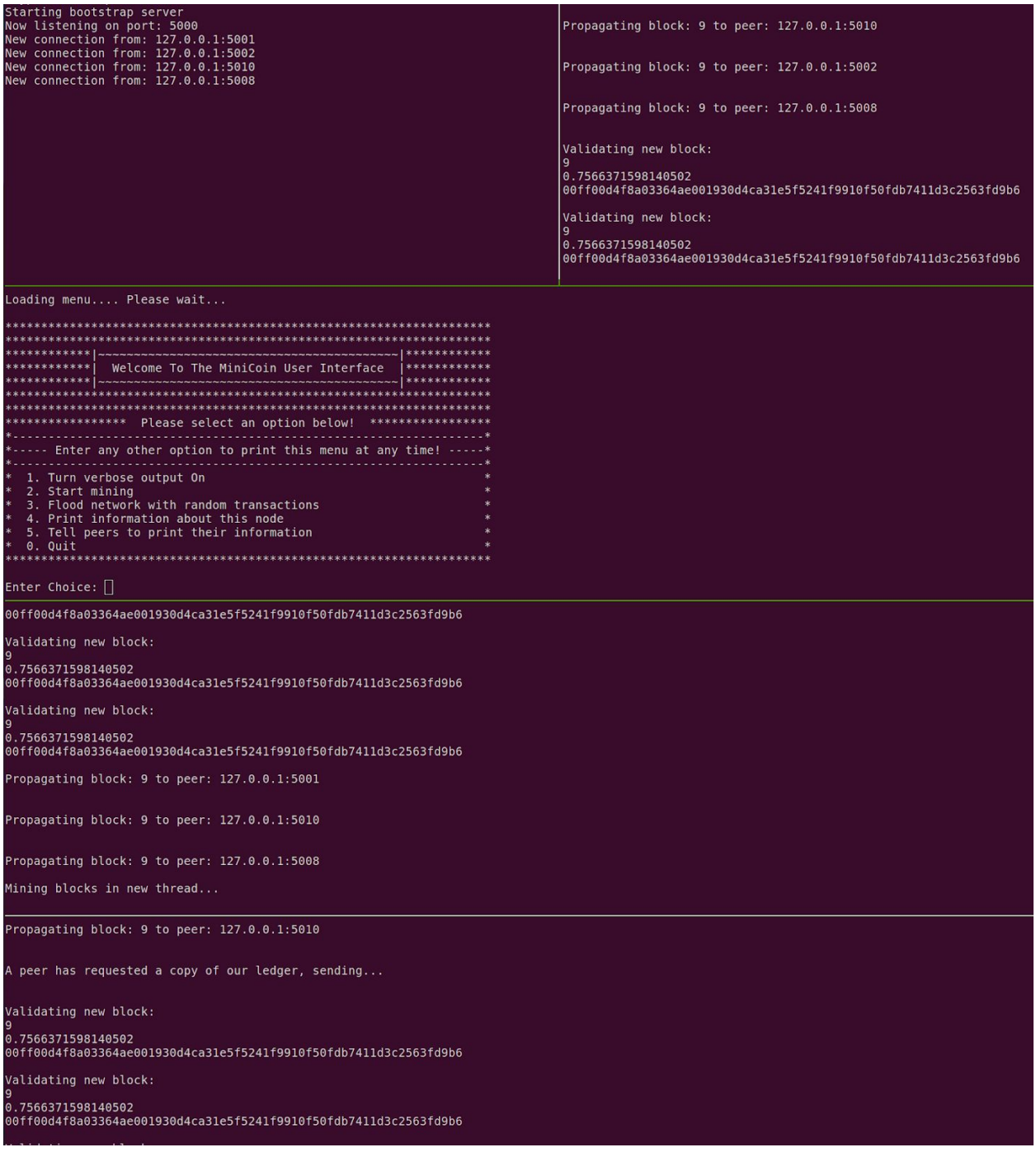


Table of Contents: 2

What it is:

MiniCoin is an application written in python that simulates a generic blockchain implementation with a scaled down set of features.
It consists of two main components, a bootstrap server and a blockchain node.
Nodes can be deployed via the command line either as a standalone node or miner non-interactively or alternatively there is a node featuring a user interface to enable interactively controlling operations and displaying general information about the state of the network.
Messaging is done with a peer to peer topology in mind, as such all communication between nodes and also the bootstrap server is done over sockets although it does not currently support interfaces other than the loopback so the network must be deployed locally.

What it does:

The Bootstrap Server:

The bootstrap server is a required and integral part of this application. It is hardcoded to listen on the loopback interface at port 5000 and exists for the sole purpose of being the first point of contact for any peer joining the network.
This server maintains a list of unique nodes that have contacted it to join the network. It will reply to any node with a set of randomly selected addresses from this list to facilitate peer discovery although it is not technically part of the blockchain network itself.

The MiniCoin Nodes:

MiniCoin nodes can be started with or without an interactive user interface.
Each node, when started, will first connect to the bootstrap server and ask for a list of up to 5 peers on the network if any exist. Should it receive the address of any peers it will then contact each of them and check if their blockchains are larger than the default one created at startup. If a larger one exists, the node will then request a copy of the longest blockchain found on the network and verify that the genesis block of this copy is identical to the hardcoded genesis block. If the validation passes the node accepts this new blockchain as its own.
Each node maintains its own unique copy of the blockchain.
If the node is started as a miner it will now start a new thread and begin mining for the next block.
Each node also maintains its own memory pool of transactions that have been announced across the network.
Any node, miner or static, is able to receive announcements whenever a new block is mined or a new transaction is announced.
If a miner discovers a new block it will double check the block is a valid addition to its current ledger to ensure that, in the case where a new block is received and accepted whilst mining, the block can be successfully appended to its ledger. If it appends this new block it will then contact each of the peers it is aware of and announce to them the existence of this new block.
Whenever any node receives a new block it will first verify it and if it passes the verification it will append it to the ledger and propagate the block to all of its peers. In the event that this block is not a valid addition to a nodes local ledger for any reason it will discard it and not propagate the block.
A similar process occurs for transactions. Whenever a new transaction is sent to a node it will first check that it does not already contain it in its mempool. If it is new it will add it to the mempool and propagate it to its peers, otherwise it discards it.
Periodically each node, in a new thread, will contact each of its peers to request the status of their blockchain. Should a peer have a longer blockchain than the node it will request a copy and validate it as it does when the node first starts.
When a node sends any kind of announcement to any other node, the receiving node will check its list of peers. Should the receiver have less than 5 peers it will add the address of the

announcing node to its list if it was not previously aware of it and any messages will now be sent to this new node in addition to the previously known nodes.
Should a node attempt to connect to another node and fail due to some connection error, the node will remove the details of this peer from its list and not contact it again unless it receives a message back at a later date. In the event that a node removes the last peer from its list due to a failed connection and has no contact details of any other nodes it will contact the bootstrap server for an updated list.
As such, nodes should discover each other dynamically as each node participates in the network and all nodes will ensure they have the most up to date copy of the ledger.

Mining:

Any node may act as a bystander simply maintaining a local copy of the blockchain and propagating messages to its peers or a miner who will actively participate in minting new blocks and announcing them to the network.
When a node is run as a miner it will create a new thread to mine for blocks. This will continue indefinitely until the node is told to stop by the user.
Whenever a valid block is minted it is verified against the miners current copy of the blockchain and appended if possible, only then is the existence of this block announced to each of the nodes peers. The miner will then return to mining the next block in its sequence.
Should a miner receive a new block from a peer which is deemed valid or discover that its copy of the blockchain is out of sync it will discard its current mining progress until the blockchain has been updated before attempting to begin mining again.
Should a miner have transactions in its mempool it will select up to 10 at random and include these in the block it is currently mining.
Should any node receive a new and valid block containing one or more transactions it will first append this block to its ledger and then remove any of the transactions in the block from its mempool.

The ledger, blockchain, blocks and validation:

Each node keeps a local copy of the blockchain as a ledger.
The ledger itself simply keeps an ordered list of blocks beginning from a hardcoded **genesis block** and ending with the last discovered and valid block. This is the blockchain.
The **genesis block** has been created in advance. Whenever a node receives a copy of the blockchain from a peer it will always check to ensure the new blockchain and its current blockchain share the same genesis block, if they do not or the new blockchain is not larger than the current copy it is not accepted.
Each block in the blockchain contains the following information:

- A block id which is a sequential counter indicating where in the chain it lies, with the genesis block having an id of 0.
- A copy of the previous blocks hash value.
- An optional list of transactions included in the block.
- A nonce which is a randomly generated float included with the block to be used for hashing.
- A hash of the block itself.

For simplicity, hashing and verification of blocks is done by first converting blocks into a string representation in a specific format and then hashing this string.
The **hash** of a block is derived by hashing a string of new line separated data contained in the block including its *ID, nonce, the hash of the previous block and the list of all transactions within the block if present.*
For a blocks' hash to be considered valid the first 6 characters of the hash hex digest must be of the form "**00ff00"**.
For a block to be successfully appended to the current blockchain it must have the following characteristics:

1. The hash must be considered valid as described above.
2. The id of the block must be of the correct sequence based on the length of the blockchain.
3. The hash of the previous block referenced by this new block must match the hash of the current head of the blockchain on the node.
4. The block is hashed again by the receiving node independently and it is found that its hash value is correct.

Only if all these criteria are met is the block considered valid and accepted as the new head of the blockchain.

Technical details:

- Hashing is performed by using the SHA3-256 algorithm and always returned as a base16 encoded string.
- The hash acceptance criteria is a block that evaluates to a hash beginning with "**00ff00**".
- Miners brute force a nonce to derive an acceptable hash value using random number generation. They will not stop until a user requests them to and will discard invalid blocks or once being currently brute forced in the event that the local ledger changes.
- When mining, a node will use a random number generator to create a float value between 0 and 1. This becomes the **nonce**. The block is then hashed with this nonce to verify it meets the hash acceptance criteria. If it does not, a new nonce is generated and the process repeats.
- Preliminary testing indicated that on average a new block can be mined in approximately 45 seconds.
- Transactions are represented as a transaction object containing a string field "id" and a string field "data" purely to simulate how they would act in a real environment.
- Each block may contain at most 10 transactions.
- **Blocks and Ledgers are NOT persistent.**
- Every node will attempt to verify and sync its ledger approximately every 45 seconds.
- Every node acts as both a client and server in terms of network connectivity and can both send and receive messages.
- Mining, message handling and syncing are all done in separate threads outside of the main thread so they do not block each other and can operate asynchronously.
- Each node is capable of printing out a human readable representation of its current local ledger.
- Since each node will operate multiple threads, *semaphores* are extensively used to prevent race conditions and block certain actions such as appending blocks to a ledger that is currently being synced.
- Each node can run as a simple node or a miner, each of which can be run with or without a ui.
- The ui enables additional features such as turning mining on and off and changing the verbosity of messages printed to the screen.
- Currently only the loopback interface is supported with unique ports used for messaging.
- The bootstrap node is not smart and only exists to provide an entry point to new nodes entering the network.
- Nodes are able to enter and leave at any time without disrupting the network.
- Mining may not always restart immediately following a ledger update and in the event that a miner continues to mine an obsolete block it will either discard and restart at its earliest convenience or mint a block and discard it. Should it propagate an obsolete block, peers will discard it anyway.
- Nodes started without the UI are verbose by default, nodes started with the UI are not.
- Nodes will dynamically update their list of peers as connections are received from new nodes or connections fail when attempted, storing a maximum of 5 total peers.
- Nodes reducing their list of peers to 0 will request an updated list from the bootstrap if available.

How to run MiniCoin:

Step 1: Start the bootstrap server.

This is required as without the bootstrap new peers will be unable to discover existing peers.
This bootstrap server will listen for new connections on port 5000.

- python3 minicoin.py --type bootstrap

Step 2: The easy way (recommended).

With the bootstrap server running you may now start as many nodes as you like.
For ease I recommend starting each node with the optional user interface in a new window.
You must specify a unique port number when starting a node of any type by changing the value after the “--port” option.

- python3 minicoin.py --type node-ui --port 5001

Step 2: The “hard” way (no user interface).

Not really that hard.
With the bootstrap server running you may start as many new nodes as you like but each node started without a user interface is quite verbose and will print updates to the terminal for many events.
As such it is advised that each node is run in a unique terminal or window.
You may start a node with no UI as either a simple node or a miner.
You must specify a unique port number when starting a node of any type by changing the value after the “--port” option.

- Simple node:
    - python3 minicoin.py --type node --port 5002
- Miner node:
    - python3 minicoin.py --type node --port 5003 --mine

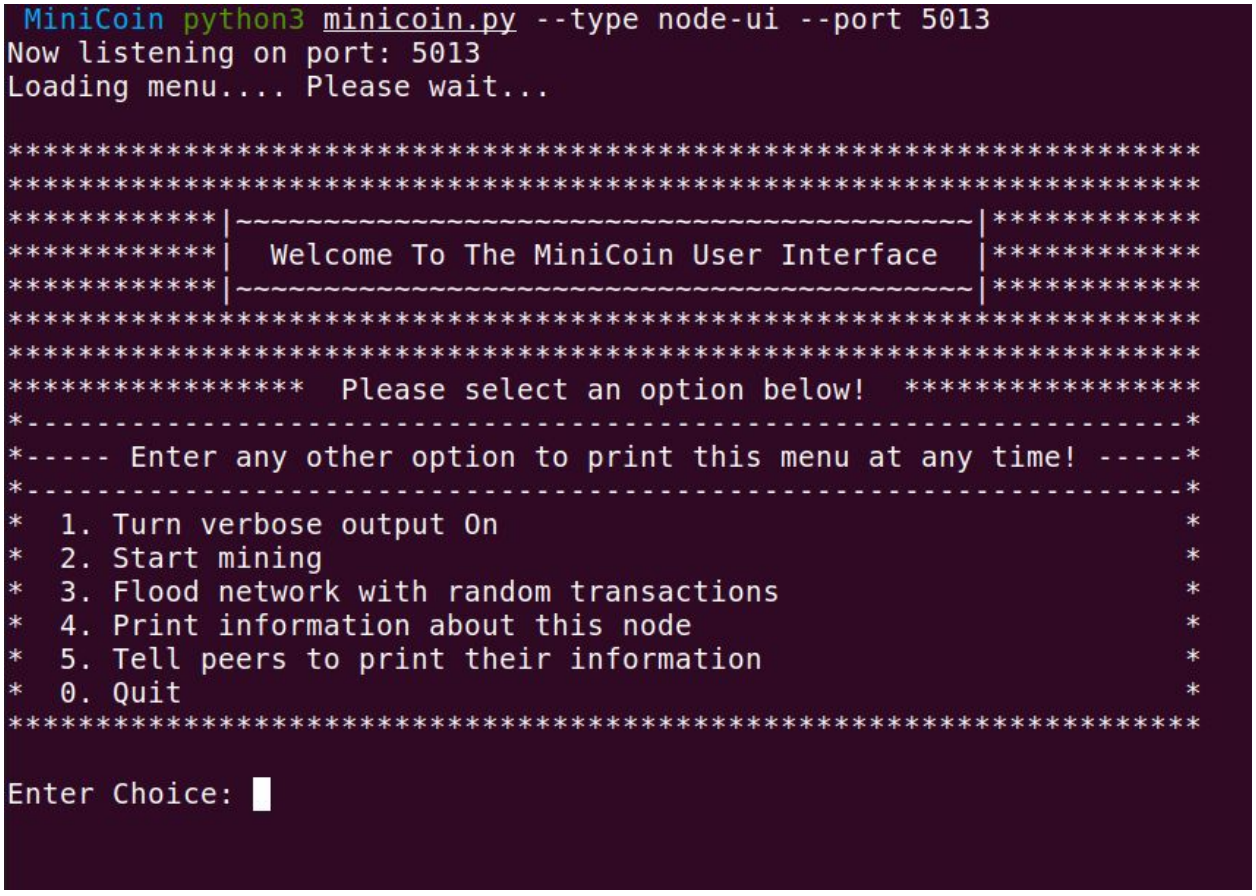
A miner node will mine until a user presses enter. It will then stop its mining thread and revert to being a simple node.
For convenience, you may also start a simple node that will periodically print its own ledger out in human readable format and then request each of its peers do the same.

- Periodic printing node:
    - python3 minicoin.py --type node --port 5004 --print

A Mining node cannot operate as a printing node and vice versa.

The MiniCoin User Interface:

A brief overview as the UI trivialises the need for a comprehensive manual.



If you have opted to run one or more nodes with a user interface you will see the above menu displayed in your terminal.
Since each of the options is somewhat self explanatory, rather than going into detail I will provide a brief overview as to what each option does.
For best results I advise running multiple nodes, either with or without a user interface alongside this one including at least one mining node though you can make the current node mine using the UI.
*Note: You must have a bootstrap server running **BEFORE** you opt to run a node of any type, you may shut down the bootstrap server once all your nodes are connected if you do not plan on adding any more to the network but I would advise you leave it running regardless.*

The menu options:

At any time, should you wish to display the menu again, pressing *ENTER* alone or entering any *invalid choice* will reprint it.

1. This toggles verbose output of messaging on the current node. By default it is off. Turning this on will cause this node to print status messages to the screen and may result in pushing the menu off screen.
2. This will cause the current node to begin mining for new blocks in a new thread. Pressing enter again will stop the mining process and return to this menu.
3. This option will create 20 randomly generated dummy transactions after 3 seconds. It will then announce these transactions to all of its peers.
4. This will print out the current nodes' local copy of the ledger and its mempool in a human readable format. The output can be quite large depending on the total transactions on the network and current length of the blockchain.
5. This will make the current node contact each of its peers and request that they print out their ledger and mempool in their respective windows just as option 4 does for the current node.
0. Shut down the current node.

Congratulations, you are successfully operating your own scaled down blockchain network.