Developing a Mining Pool for SmileyCoin

Breki Ingibjargarson
Brynjar Ingimarsson
Valdimar Björnsson
Emma Líf Jónsdóttir

December 14, 2019

Contents

1 Introduction

1.1 Motivation

The most simple method of mining cryptocurrencies is individual mining. However as competition increases the chances of an individual miner finding a new block becomes infinitesimal. Mining pools are used to make it realistic for smaller miners to return profit. In a large mining pool new blocks are regularly found and the profits are distributed in proportion to each miners computational effort.

Our goal is to understand the protocols behind mining pools and develop a functional mining pool for SmileyCoin (SMLY), which is a fork of Bitcoin. The mining pool is written in Django, a popular web framework for Python. We start by going over the foundations of the blockchain and its protocols, followed by a general description of how mining pools work, and finally we describe our own implementation.

2 Foundations

Before we develop a functional mining pool, we need to understand the foundations of the blockchain, transactions and the protocols that miners, pools and wallets communicate with.

2.1 The Blockchain

The blockchain is a data structure in which each block contains a number of transactions. The header of each block contains the hash of the previous block, forming a chain. To add a new block to the top of the chain, miners need to find a solution to a difficult hashing problem, or else it will be rejected from the network.

The Structure of a Block

This section describes the data structure behind blocks.

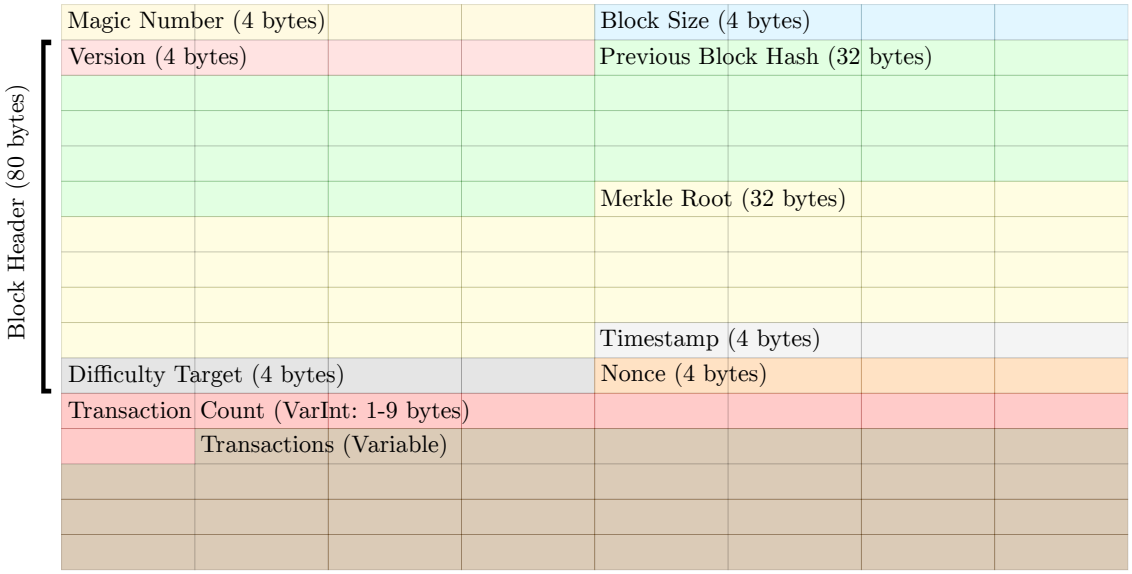

Figure 1: The Block Header

Magic Number

This field is always left as `0xD9B4BEF9`, the magic number for Bitcoin. Magic numbers are commonly used in computer science to identify file formats and protocols.

Block Size

The total size of the block in bytes. As the field is 4 bytes long, the theoretical maximum size of a block is 2^{32} or around 4 gigabytes.

Version

The version of the software. Usually left as 1 or `0x0001`.

Previous Block Hash

The SHA256 hash of the previous block. This is how the block is linked to its previous block, thus forming a chain. Note that there is no field that specifies the height of the block.

Merkle Root

The Merkle root is computed from the hashes of all the transactions in the block. Thus to check if a transaction is included in a block, you only need to calculate $\ln_2(n)$ hashes.

Timestamp

The UNIX timestamp of when the block was found. In other words, the number of seconds between the date it was found and January 1, 1970.

Difficulty Target

The difficulty of mining an acceptable block is determined by the difficulty target. To successfully submit a block to the network, miners need to find a hash for the block header such that the result is below the difficulty target.

Nonce

The nonce can take any value. Miners iterate through random values and apply a hashing function (such as SHA256) until they find a nonce that satisfies the difficulty target. The output of the hashing function is a 256 bit integer so miners might have to go through millions of hashes before finding a solution.

2.2 The Transaction

Here we describe how transactions are encoded as bitstrings. For the purpose of developing a mining pool we only need to understand the coinbase transaction.

version	4 bytes	Transaction version number.
tx_in_count	1 byte	Number of inputs in this transaction, coinbase has one input.
hash	32 bytes	A 32-byte null as the coinbase has no previous outpoint.
index	4 bytes	Left as 0xFFFFFFFF for the coinbase.
script_bytes	variable	The length of the coinbase script, up to 100 bytes.
height	4 bytes	The block height.
coinbase_script	variable	The coinbase field, can contain any data up to 100 bytes.
sequence	4 bytes	An end sequence for the input 0x00000000
tx_out_count	1 byte	Number of outputs in this transaction.
value	8 bytes	The amount that should be spent in satoshis.
pk_script_bytes	variable	The length of the pubkey script.
pk_script	variable	The pubkey script that needs to be satisfied to spend this value.
lock_time	4 bytes	The block height or timestamp when this output can be spent.

Table 1: Structure of the coinbase transaction, colored rows may be repeated.

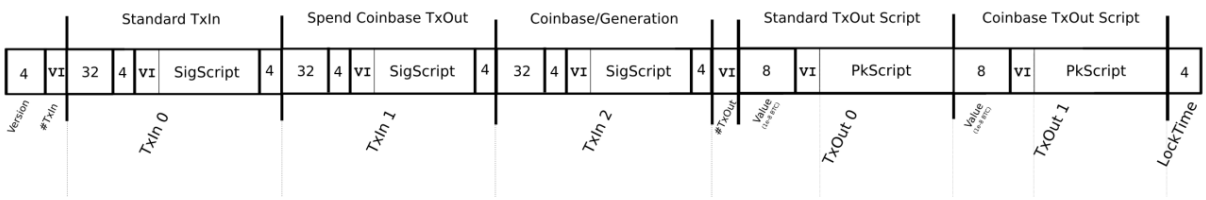

Figure 2: Structure of a Typical Transaction.

The coinbase transaction is a special transaction that miners are allowed to include in their blocks, intended as a reward for their work. It does not need previous transaction outputs to spend like normal transactions.

For SmileyCoin the coinbase transaction has a value of 10000 SMLY and is cut in half every 1.22 million blocks. Two 45% shares must go to charity funds and the other 10% can be used by the miner. This means that the coinbase transaction usually has three outputs, but it would be possible to distribute the miner reward by adding more outputs.

2.3 GBT

Here we describe how the GBT (getblocktemplate) protocol for mining works. The simplest way to communicate with the wallet is using its HTTP interface which uses JSON-RPC.

getblocktemplate

Miners can use the `getblocktemplate` command to get information about what to mine.

```
$ curl --data-binary '{"jsonrpc":"1.0","id":0,"method":"getblocktemplate","params":[]}' \
    -H 'content-type:text/plain;' http://user:password@127.0.0.1:14242/
{
    "result" : {
        "version" : 2,
        "previousblockhash" : "42158955048b9f47415e91e97e7717be5e045f5f450f49d4fe3a3f92633078bd",
        "transactions" : [
        ],
        "coinbaseaux" : {
            "flags" : "062f503253482f"
        },
        "coinbasevalue" : 1000000000000,
        "target" : "0000000006240d0000000000000000000000000000000000000000000000000000",
        "mintime" : 1576188585,
        "mutable" : [
            "time",
            "transactions",
            "prevblock"
        ],
        "noncerange" : "00000000ffffffff",
        "sigoplimit" : 20000,
        "sizelimit" : 1000000,
        "curtime" : 1576190134,
        "bits" : "1c06240d",
        "height" : 608403,
        "richaddress" : {
            "payee" : "B5HWQyy9pDzjU6zGwshqCXQBAiXwSuT9zw",
```

```
            "script" : "76a9140928aa8fcc6263e3997e1d0c4df62ca09af40f0388ac",
            "amount" : 450000000000
        },
        "EIASaddress" : {
            "payee" : "BPVuYwyeJiXExEmEXHfCwPtRXDRqxBxTNW",
            "script" : "76a914d0f33c697499951e3f416cdb366440432f5cccc088ac",
            "amount" : 450000000000
        }
    },
    "error" : null,
    "id" : 0
}
```


Some of these attributes are optional. Miners will typically require the following attributes.

coinbasetxn	The coinbase transaction in hexadecimal format.
previousblockhash	The hash of the previous block in the chain.
transactions	An array of all transactions to include in the block, in hexadecimal format.
expires	How many seconds miners should wait until asking for a new template.
target	The difficulty target for the block. Miners will submit when target is met.
height	The height of the current block in the chain.
version	The version of the protocol, usually left as 2.
curtime	The current UNIX timestamp.
mutable	An array of options for mutating the block, `coinbase/append` is often allowed so miners won't run out of nonces.

Table 2: Attributes for the GBT response.

submitblock

When miners have found a solution they can submit the whole block in hexadecimal format with the `submitblock` command, which takes the block hex as argument.

getwork

Before GBT was introduced, miners used the `getwork` command to know what to mine. It is simpler than `getblocktemplate` as it doesn't output all the block information, but only the header. The miners can then iterate through 2^{32} nonce values. As mining got more popular this became a limitation. Big mining rigs could solve gigahashes per second and quickly run out of nonce values. With GBT, miners can change other parts of the block so they won't run out of combinations.

Stratum

Stratum is another popular protocol for mining. Unlike the other two, it is not a part of the standard wallet. It was developed around the same time as GBT and was backed by a major mining pool. It is based on HTTP but unlike the other two protocols it keeps an open connection at all times, so that the pool can immediately inform miners of changes instead of clients polling the server at a regular interval (although GBT now also has this option).

3 The Process

This is a description of the process that miners and mining pools go through.

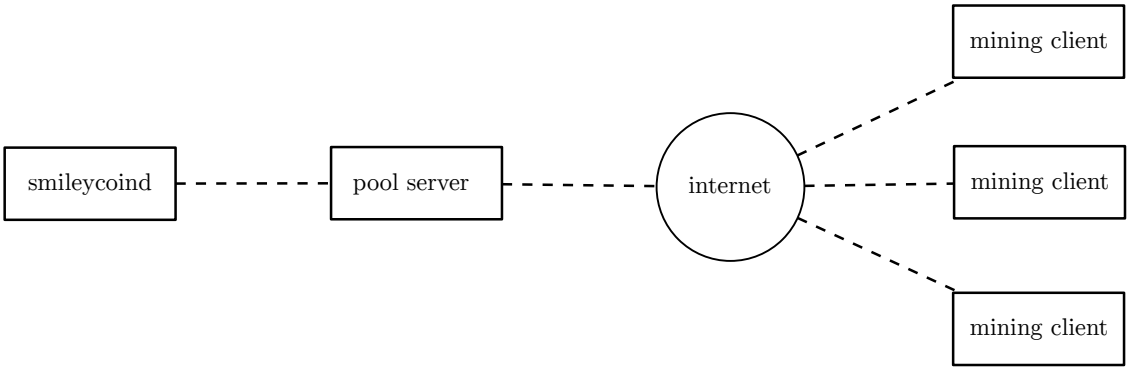

Figure 3: Network Diagram

3.1 The Mining Pool

The mining pool can be seen as a middleman. It has the core wallet on one side, which communicates with the SMLY peer-to-peer network, and maintains a copy of the blockchain and transactions that should go into the next block (the raw memory pool). The mining pool gets the template of the block that should be mined, and coordinates with mining clients to find the golden nonce.

3.2 Shares

When the wallet is asked for the block template, it responds with all the data that should go in the next block, along with the difficulty target. The actual difficulty target can be very high, making it impossible for smaller miners to find solutions.

Mining pools usually get the block template and target from the wallet and create their own templates with the same data but lower targets for mining clients. This way miners will regularly submit solutions and the mining pool can check if it also meets the network target, and in that case forward the block submission to the wallet. Even if most solutions submitted by miners do not meet the network target, the mining pool is getting feedback from the clients and can determine their hash rates.

3.3 Payout Systems

When a successful block is submitted, the reward is usually distributed between the clients in proportion to their efforts (determined by their hash rates). The majority of mining pools use either the PPS (Pay Per Share) or PPLNS (Pay Per Last N Shares) payout systems.

In the first one, the miner is guaranteed to be paid for each share that he submits in proportion to the total number of shares received by the pool. Usually the miner will be paid immediately after submitting the block. The second one is similar, except that when the pool finds a block only miners from the last N number of shares will be rewarded. This way the miner is only paid once every block and there is less risk for the pool operator.

4 Implementation

The mining pool is written in Django, a web framework for Python which allows for very rapid development. With Django we could focus on the practical side rather than solving unrelated technical difficulties.

The mining pool is very lightweight and only spans around 500 lines of code. It consists of two HTTP routes, one for `/` which miners use, and `/info` which is a web interface showing pool status. The routes are stored in `urls.py` and their corresponding endpoints in `views.py`. Furthermore we wrote two custom classes. The first is `Block` in `block.py` for constructing the block, templates and converting various hexadecimal values. The second is `RPC` in `rpc.py` for communicating with the wallet.

The mining pool uses a SQLite database to store the shares submitted by miners. Its structure is found in `models.py`, it stores the share timestamp, address of the miner, height of the block, difficulty of the submission and the actual difficulty required by the wallet.

The miner starts by sending a POST request to `/` with the `getblocktemplate` command. The server then asks the wallet for the current block template, uses that information to construct a `Block` instance with the same data but a custom coinbase and a much lower difficulty.

When the miner has found a solution it sends a POST request to `/` with the `submitblock` command. The server then calculates the difficulty of that submission and if it matches or exceeds the network difficulty, the server uses the `RPC` class to send a `submitblock` to the wallet, thus successfully mining the block. If the submission difficulty is too low nothing happens, but information about the submission is stored in the `Shares` SQL table.

4.1 Using the Mining Pool

More detailed installation instructions can be found on GitHub.

The server can be started with the following command.

```
$ python manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 14, 2019 - 02:51:36
Django version 2.2.7, using settings 'smlypool.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

The information web page is then available at `http://localhost:8000/info`.

We use an ASIC miner called FutureBit Moonlander 2 to test the mining pool. It comes with a custom build of `bfgminer`, a popular mining client. We start the miner with the following command, where we pass our payout address as username.

```
$ bfgminer --scrypt -o http://localhost:8000 -u BEppJqTLw5ByePPbZwm7hByqqwcmsCtVfK -p y -S ALL \
    --set MLD:clock=600 --no-getwork --no-stratum
```

The miner should start blinking and soon it outputs something like this.

```
[2019-12-14 02:50:26] Accepted 000ec771 MLD 0  Diff 67m/10m
[2019-12-14 02:50:34] Accepted 0041d12a MLD 0  Diff 15m/10m
[2019-12-14 02:50:37] Accepted 0038248a MLD 0  Diff 17m/10m
[2019-12-14 02:50:37] Accepted 003ffc0a MLD 0  Diff 15m/10m
```

This means that it has found solutions to its shares and is submitting them to the server. We can see these shares in the web interface as well.

Time	Height	Miner	Submitted	Actual
Dec. 14, 2019, 2:50 a.m.	608802	BEppJqTLw5ByePPbZwm7hByqqwcmsCtVfK	0.0156	30.6154
Dec. 14, 2019, 2:50 a.m.	608802	BEppJqTLw5ByePPbZwm7hByqqwcmsCtVfK	0.0178	30.6154
Dec. 14, 2019, 2:50 a.m.	608802	BEppJqTLw5ByePPbZwm7hByqqwcmsCtVfK	0.0152	30.6154
Dec. 14, 2019, 2:50 a.m.	608802	BEppJqTLw5ByePPbZwm7hByqqwcmsCtVfK	0.0677	30.6154

Table 3: Shares as they appear on the information web page.

As we can see the submitted difficulty is much lower than what is needed for the network.

4.2 Results

4.3 Code Reference

class block.Block()

This class contains the structure of the current block and includes methods to construct block templates.

`create_coinbase(outputs)`

Builds the coinbase for the block, outputs is a dictionary of addresses and corresponding shares, the shares will be normalized such that the total outputs equal 1000 SMLY.

`create_gbt(difficulty)`

Constructs the `getblocktemplate` response for miners with the requested difficulty.

`get_submission_difficulty(block_hex)`

Calculates the difficulty of the hash of a raw block submission. When miners use submitblock this is used to calculate the difficulty.

`target_to_difficulty(target)`

Convert the header target field into a difficulty integer.

`difficulty_to_target_hash(difficulty)`

Convert difficulty int to a hash target for block template.

class rpc.RPC()

This class is used to communicate with the SmileyCoin wallet.

`call(cmd, [params])`

Sends a command to the wallet with optional parameters and returns the response.