

# TinyCoin: Simulating mining strategies in a simplified Bitcoin Network

Final Project for the Peer to Peer course

Francesco Balzano

Master Degree in Computer Science and Networking

A.Y. 2016-2017

---

## Contents

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>Design choices</b>	<b>2</b>
2.1	Network Nodes . . . . .	2
2.2	Blockchain, Blocks and Transactions . . . . .	3
2.3	Forks . . . . .	3
2.4	Oracle . . . . .	3
2.5	Network Latencies . . . . .	3
<b>3</b>	<b>Implementation</b>	<b>5</b>
3.1	Protocols . . . . .	5
3.2	Initializer . . . . .	6
3.3	Control . . . . .	6
3.4	Other Classes . . . . .	6
<b>4</b>	<b>Test</b>	<b>7</b>
<b>5</b>	<b>Experimental Results</b>	<b>7</b>
5.1	Blocks mined by honest and fraudulent miners in the blockchain . . . . .	7
5.2	Forks in the blockchain . . . . .	7
5.3	Mined blocks per computational power . . . . .	8
5.4	Reward per computational power . . . . .	9
5.5	Number of blocks in the blockchain for different latencies of messages . . . . .	9
<b>6</b>	<b>Conclusions</b>	<b>9</b>
<b>7</b>	<b>Limitations</b>	<b>12</b>
<b>8</b>	<b>References</b>	<b>12</b>

# 1 Overview

In this project I have implemented a simplified version of Bitcoin, called TinyCoin, whose specifications are reported in [1]. In brief, TinyCoin distinguishing features are:

- each user has a single address that records the unspent amount at that node
- each transaction has a single input, a single output and does not include neither a digital signature nor scripts
- network nodes may be either normal nodes or miners. In turn, miners may be either honest or fraudulent (*i.e. Selfish Miners*). Each miner has a type that reflects its mining hardware: CPU, GPU, FPGA or ASIC
- there is a centralized oracle that decides which miner has created a new block of the blockchain at regular intervals of time. The decision is biased by the computational power of the miners. Each block is unique.

The goal of this project is to evaluate the selfish mining strategy defined in [2] in TinyCoin. This strategy is evaluated by taking into account different metrics and parameters. The results and the discussion of this simulation are reported in the *Experimental Results* section.

## 2 Design choices

In the following paragraphs I explain the design choices that I made. This is a high level description, indeed the classes of the project are described in the *Implementation* section.

### 2.1 Network Nodes

Any node in the TinyCoin network is either a (normal) node, or a (honest) miner or a selfish miner. According to the specifications, I have considered a static topology, which is generated at random at the beginning of the simulation.

- **node**: makes and receives transactions
- **miner**: makes and receives transactions and mines blocks of the blockchain. In particular, as soon as a new block is mined it is immediately advertized to all the nodes in the network.
- **selfish miner**: makes and receives transactions and mines blocks of the *private* blockchain. A selfish miner indeed holds both a copy of the public blockchain, which is the “official” blockchain, and a copy of the private blockchain, which is the blockchain created and maintained collectively by all the selfish miners. At any time the two blockchains may be equal or may differ for some block. If they differ, it is because the selfish miners have discovered new blocks which have been added to the private blockchain but have not been disclosed to the public yet. Indeed when a selfish miner mines a new block, it does not naively publish it and add to the public blockchain. Instead, it applies a strategy that allows it and the other selfish miners to get the maximum revenue from their computing power. One of the strategies that they can follow is explained in [2]. In this project, I chose to implement it. The pseudocode is reported in algorithms 1, 2 and 3.

**Initialization** Since we assume that all the nodes are present at the beginning of the simulation, at initialization (algorithm 1) a selfish miner has both the private and public blockchain empty, and the *privateBranchLength*, which tells how many blocks the two blockchains differ for, is set to zero.

**Selfish Miner found a block** Algorithm 2 describes what happens when one selfish miner creates a new block: the block is added to the private blockchain and the *privateBranchLength* is incremented accordingly. If there was a fork in the public blockchain (line 4), then the selfish miners are in competition with the honest miners to insert their own blocks in the final version of the blockchain. In this case the new block is advertized to all the nodes of the network. In so doing, the selfish miners are quite confident that their two blocks will be included in the blockchain, at the expense of the block mined by the honest miner. Figures 1, 2 and 3 illustrate this situation.

**Other miners found a block** Algorithm 3 describes the behaviour of the selfish miner when a (honest) miner creates a new block. If the private blockchain and the public one were equal (line 3), then the selfish

miner is fallen behind the other miners and thus accepts the new block and continues mining on top of it. If the selfish miner was one block ahead (line 6), then it advertises to all the nodes its private block, leading to a fork in the blockchain. If the selfish miner was two blocks ahead (line 8), then it publishes both its blocks. This way, it is quite confident that these two blocks will become part of the blockchain, because the branch that it creates is longer than the branch created by the honest miner. If the selfish miner had an advantage of more than 2 blocks (line 11), then it publishes only the first unpublished private block, leading the honest miners to spend clock cycles to mine blocks that at the end won't be part of the final blockchain.

## 2.2 Blockchain, Blocks and Transactions

- **transaction:** transactions are generated by nodes. They have an input node, an output node, an amount and a fee. For the sake of simplicity, I have assumed that each node pays the transaction at the moment of creation, and that for each transaction the fee is fixed in percentage.
- **block:** a block basically contains a set of transactions. For the sake of simplicity, I assumed that the reward for the block creation is immediately given to the miner. If such block is later removed from the blockchain, then also the reward given to the miner is given back. The block creation is arbitrary: it is not required to the miners to solve any Proof Of Work, instead it is the oracle that chooses at random the miner that has to create the new block of the blockchain.
- **blockchain:** a blockchain is a list of blocks. It does not include hash pointers, so it is not tamper-free. At any node, a block is added to the blockchain if it has been created by a miner and if its parent matches the current last block of the blockchain.

## 2.3 Forks

Forks have a serious impact on the selfish mining strategy. Indeed, selfish miners goal is basically to fork the blockchain, creating a branch that eventually becomes longer than the “official” one and thus gets included in the blockchain, to the detriment of the branch created by honest miners. There are two kinds of forks that are supported in this project:

- **selfish miner:** it supports forks of arbitrary length, that is the public and the private blockchain may differ for an arbitrary number of blocks from the point of view of the selfish miner. This fact is necessary in order to apply its strategy, as described in algorithms 1, 2 and 3.
- **node:** a normal node only supports a simplified kind of fork, that is one in which there are two branches made of only one block. The next block that is received by the node that extends one of the two branches actually resolves the fork. If at this point another blocks arrives willing to extend the abandoned branch, it will be simply discarded. There are two reasons for this assumption. The first is simplicity: at any time there are no orphan branches in the blockchain, but only a simple fork is admitted. The second is tamper resistance: since the blockchain is not tamper-free, avoiding to leave branches in the blockchain and allowing to fork only the last block of the blockchain means that fraudulent miners cannot mine on top of arbitrary, orphan branches.

## 2.4 Oracle

The Oracle is the entity that decides which miner has solved the proof of work and created a new block. In this project the oracle takes this decision periodically at fixed intervals of time. Each interval has the same duration.

## 2.5 Network Latencies

Different latencies for the message exchange are tested. A base latency is defined in the configuration file, then different delays are introduced with respect to this base latency.

---

**Algorithm 1** Selfish Miner initialization

---

- 1: public chain  $\leftarrow \emptyset$
  - 2: private chain  $\leftarrow \emptyset$
  - 3: privateBranchLength  $\leftarrow 0$
  - 4: Mine at the head of the private chain
- 

---

**Algorithm 2** Selfish miners found a block:

---

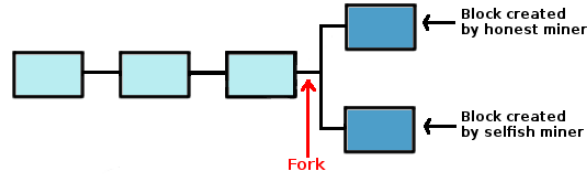
- 1:  $\Delta_{\text{prev}} \leftarrow \text{length}(\text{private chain}) - \text{length}(\text{public chain})$
  - 2: append new block to private chain
  - 3: privateBranchLength  $\leftarrow \text{privateBranchLength} + 1$
  - 4: **if** ( $\Delta_{\text{prev}} = 0$  AND privateBranchLength = 2) **then**
  - 5:     private chain  $\leftarrow$  public chain
  - 6:     privateBranchLength  $\leftarrow 0$
  - 7: Mine at the new head of the private chain
- 

---

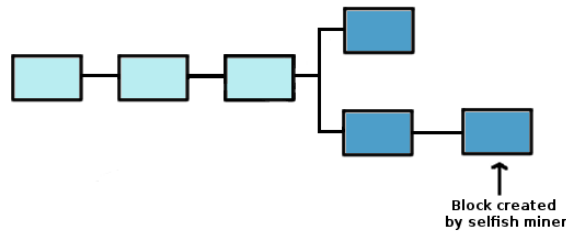
**Algorithm 3** Honest miners found a block:

---

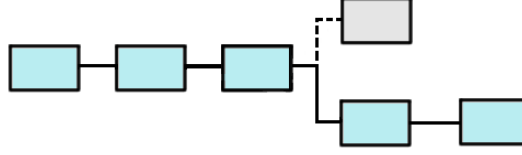
- 1:  $\Delta_{\text{prev}} \leftarrow \text{length}(\text{private chain}) - \text{length}(\text{public chain})$
  - 2: append new block to public chain
  - 3: **if** ( $\Delta_{\text{prev}} = 0$ ) **then**
  - 4:     private chain  $\leftarrow$  public chain
  - 5:     privateBranchLength  $\leftarrow 0$
  - 6: **else if** ( $\Delta_{\text{prev}} = 1$ ) **then**
  - 7:     publish last block of the private chain
  - 8: **else if** ( $\Delta_{\text{prev}} = 2$ ) **then**
  - 9:     publish all of the private chain
  - 10:     privateBranchLength  $\leftarrow 0$
  - 11: **else**
  - 12:     publish first unpublished block of the private blockchain
  - 13: Mine at the new head of the private chain
- 



**Figure 1:** When two different blocks are generated having the same parent (usually these two blocks are generated very close in time), a fork is created in the blockchain.



**Figure 2:** When a selfish miner creates a new block, it adds it immediately to the public blockchain if and only if there is a fork in the blockchain. This way, selfish miners are one block ahead with respect to other miners, and so the probability that their two blocks will be part of the final blockchain increases.



**Figure 3:** Selfish miners have reached their goal: their branch is the longest, and so all the miners that were mining on the shorter branch abandon it and start mining on the other, longer branch.

	Node	Miner	Selfish Miner
NodeProtocol	✓	✓	✓
MinerProtocol	✗	✓	✗
SelfishMiner Protocol	✗	✗	✓

**Figure 4:** Relationships between node types and node protocols.

### 3 Implementation

I set up the simulation using the event driven version of the Peersim simulator. The classes of this project are in some way dictated by the framework itself. For instance, I wished to have three different classes Node, Miner and SelfishMiner that extended the base class NetworkNode. Each network node would have been one among Node, Miner and SelfishMiner, so that it was immediately clear which was its role in the network. Anyway this was not possible because I had to follow the structure of classes defined by the framework. In this section, I describe the classes that I developed in reference to their role in the framework.

#### 3.1 Protocols

Every node in the network is a TinyCoinNode. If the node is a normal node, only the NodeProtocol is enabled; if it is a miner, both the NodeProtocol and the MinerProtocol are enabled; if it is a SelfishMiner, both the SelfishMinerProtocol and the NodeProtocol are enabled (figure 4).

- **NodeProtocol** The NodeProtocol is responsible for creating and advertising transactions and for publishing blocks of the public blockchain. The method *nextCycle()*, called at every cycle, creates a transaction with a fixed probability defined in the configuration file. Both the amount and the recipient of the transaction are random. Then the transaction is published on the network. The method *processEvent()* is invoked every time the NodeProtocol receives a message. A message may be either a transaction or a block. In the former case, if the node hasn't already seen it, it adds the transaction to its transaction pool and floods it to its neighbors. In the latter case the behaviour is different based on the type of node. If the node is a normal node or a honest miner, it checks whether the received block has to be added to the blockchain, checks the eventual presence of forks, makes some other operation and finally floods the block to all its neighbors. Else, if the node is a selfish miner it implements the algorithm 3. The fact that also the selfish miners run the NodeProtocol could seem weird, but actually there are two reasons for this choice. First: a selfish miner is also a node, able to make and receive transactions. Second: I use the NodeProtocol as a “public layer” to exchange public information among the nodes of the network, namely transactions and blocks of the public blockchain. Instead, I use the SelfishMinerProtocol as a “private layer” to exchange private information inside the selfish miners pool, namely the blocks of the private blockchain not disclosed to public yet.
- **MinerProtocol** The MinerProtocol is run by the miners. At each cycle the method *nextCycle()* is invoked, which is actually executed only if the oracle selected this miner for the current cycle. If this is the case, the

miner picks a set of transactions from its transaction pool and creates a new block. The block is immediately advertised to all its neighbors and the revenue for mining a new block is immediately assigned to the miner.

- **SelfishMinerProtocol** The SelfishMinerProtocol is run by the selfish miners. At each cycle the method *nextCycle()* is invoked, which is actually executed only if the oracle selected this selfish miner for the current cycle. This method creates a new block using the transactions in the transaction pool and then implements the algorithm 2, that is either shares the block with the selfish miners or with all the nodes based on convenience. The method *processEvent()* is invoked every time a “secret” block is received from a selfish miner, so it simply implements the algorithm 2.

### 3.2 Initializer

- **NodesInitializer** The NodesInitializer initializes the nodes in the network based on the probability values received from configuration file. Namely, each network node has a given probability of being either a normal node, a miner or a selfish miner. If it is a miner or a selfish miner, it has a given probability of mining using either CPU, GPU, FPGA or ASIC.

### 3.3 Control

- **TinyObserver** The TinyObserver monitors the simulation, logging on disk statistics that will be later plotted. These statistics include the number of forks occurred in the simulation, the number of blocks in the blockchain that have been mined by honest and fraudulent miners, the total number of blocks mined on varying the message latency and the number of blocks mined per unit of hash rate by honest and selfish miners on varying the number of selfish miners in the network.
- **Oracle** The method *execute()* of the Oracle is invoked once per cycle. This method selects at random the miner that has created the new block. In some (rare) case, the oracle may decide that two miners have concurrently solved the Proof of Work, and so two different blocks will be mined such that they will result in a fork of the blockchain. This may happen in practice, so I decided to model this case. At each cycle, each miner has a probability of being selected that depends on its computing power: the more the power, the more the probability that such miner has solved the Proof of Work. During the first invocation, the oracle initializes the probabilities of selecting a miner of a given type, taking into account the hash rate of each miner type and the total hash rate of the network. The hash rate of each miner type is a parameter that is defined in the configuration file. The total hash rate in the network depends on the number of miners in the network and on their type. The random selection of a miner is done in two steps. In the first, it is randomly chosen the type of the miner. In the second, the nodes in the network are shuffled and then they are scanned in this new order, taking the first miner with appropriate type.

### 3.4 Other Classes

In this paragraph I describe the classes of the project that don't belong to the previous categories:

- **Transaction** A Transaction is a simple class having one input node, one recipient node, a reward to be paid to the recipient and a fee to be paid to the miner that includes the transaction in the block.
- **Block** A Block contains a set of transaction. It is identified by a unique identifier and specifies the parent block of the blockchain, the miner that created the block and the reward that such miner has earned for creating the block.
- **TinyCoinNode** Every node in the network is a TinyCoinNode. A TinyCoinNode has a type (NodeType) and, in case it is a miner, a MinerType. Then it maintains a copy of the public blockchain, a transaction pool holding the unconfirmed transactions (the ones not yet inserted in a blockchain block) and the current balance of the node.
- **NodeType** It is a enum that indicates the type of each node. The type can be either Node, Miner or SelfishMiner.
- **MinerType** It is a enum that indicates the type of a miner. It can be either CPU, GPU, FPGA or ASIC.

## 4 Test

It could happen that a node receives a block  $b_i$  that can't be attached to the head block of the blockchain  $b_{head}$  because it has to be attached to a block  $b_{i-1}$  that the node has not received yet. In this case, the block  $b_i$  is not discarded but instead it is inserted into a pool of “missed” blocks. Every time a new block is inserted in the blockchain the pool is scanned to determine whether it contains some blocks that now can be added to the blockchain. This means that when the node will receive the block  $b_{i-1}$  it will add that block to the head of the blockchain and then will scan the missed blocks pool. It will thus discover that the block  $b_i$  in the pool now has to be added to the blockchain, and so it will insert it.

I have written two classes, *TestRunner* and *MissedBlocksTest*, to test the methods of *NodeProtocol* that implement this functionality. Actually a simplified version of those methods is tested, because I was interested in testing only the parts that deal with the pool of missed blocks. I have written two test methods inside the class *MissedBlocksTest*:

- *testWrongArrivalOrder()*: Verifies that each node is able to add to the blockchain, in the correct order, blocks that arrive in wrong order;
- *testForkResolutionWithMissedBlocksPool()*: Demonstrates that the blocks inside the missed block pool can be also used to resolve a fork

## 5 Experimental Results

Let us define *experiment* the single execution of the Peersim Simulator, for a given number of cycles, with the classes described above. An experiment logs statistics about its execution on files. These statistics depend on the values of the parameters defined in the (Peersim) configuration file.

Let us define *simulation* the repeated execution of the experiment. Namely, the parameters in the configuration file are assigned more values and for each configuration of values the experiment is carried out several times and the statistics produced are averaged and plotted. It is possible to run a simulation in a fully automatized way thanks to the use of scripts. These scripts, together with all the code of the project classes, are reported in the *Implementation* report.

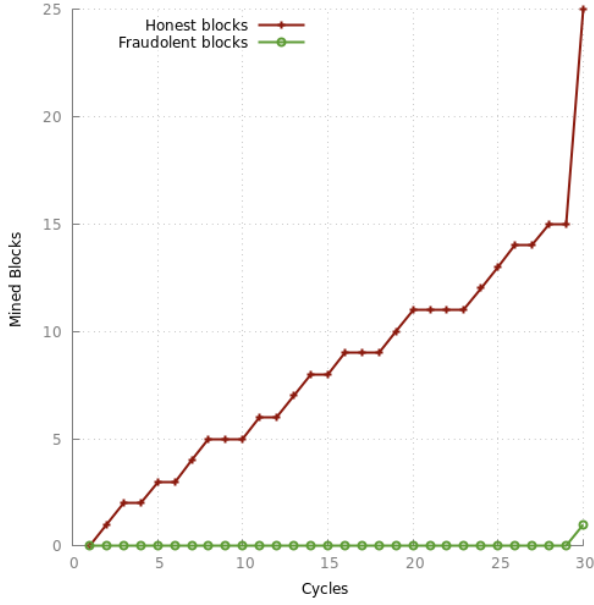
The value of the parameters used to run the simulation are specified in the configuration file. In this simulation, each experiment has a maximum duration of 30 cycles. For each configuration of the parameters, the experiment has been repeated 10 times. Finally the collected statistics have been averaged and plotted. In the following paragraphs, I report and comment these statistics.

### 5.1 Blocks mined by honest and fraudulent miners in the blockchain

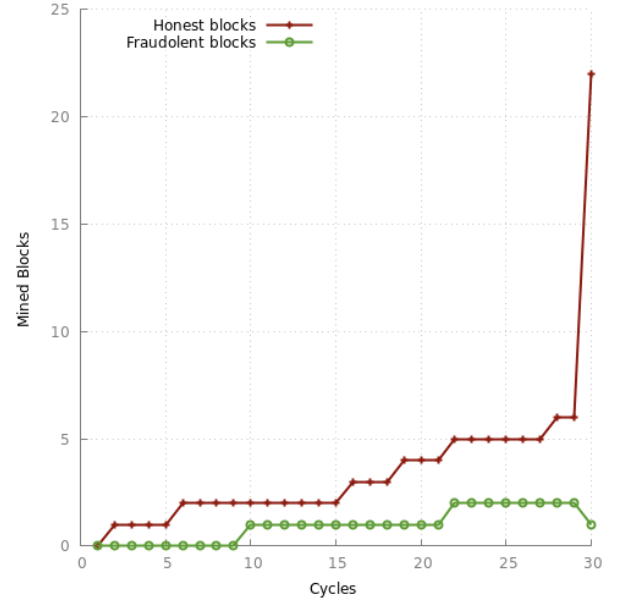
Figures from 5 to 10 show how many blocks have been mined by honest miners and how many from selfish miners, on varying the probability that a miner is a selfish miner. This probability is a parameter in the configuration file that is passed to the *NodesInitializer* class, which decides whether a network node is a normal node, a miner or a selfish miner. The graphs show that the system behaves as expected: if the probability of having selfish miners is small, and thus the number of selfish miners is small (making several experiments and taking the average reduces the variance of the number of selfish miners), the number of blocks mined by selfish miners is small, whereas the number of blocks mined by honest miners is big. If the number of selfish miners increase the two lines get closer (figure 6) but when the number of selfish miners is more or less equal to the number of honest miners (figure 7), the number of blocks mined by selfish miners is significantly higher than the number of blocks mined by honest miners. This is a demonstration that in this system the selfish mining strategy is effective. If the number of selfish miners further increase, the honest miners are not able to mine any block that will end up in the blockchain.

### 5.2 Forks in the blockchain

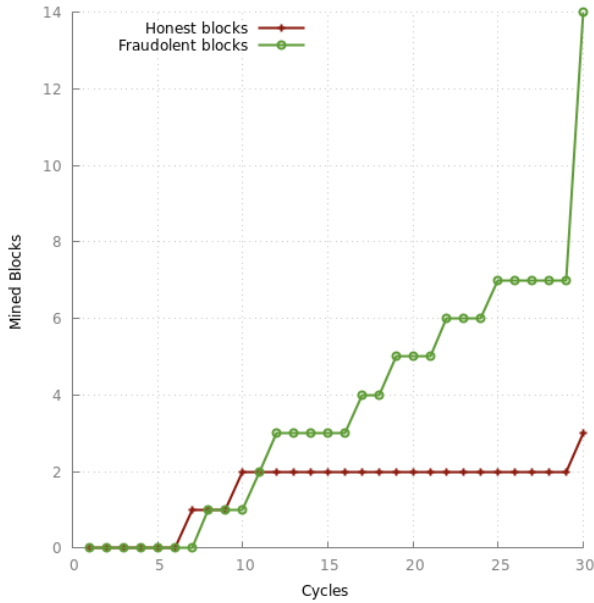
Figure 11 shows the number of forks in the blockchain on varying the probability that a miner is a selfish miner. Several lines are plotted on the same graph, each one parameterized with a different value of probability. The maximum number of forks is when the probability is 0.50, and this is probably due to the fact that in this case the struggle between honest miners and selfish miners to put their blocks in the blockchain is maximum because their hash rate is balanced. Instead, when the number of selfish miners is much bigger than the number of honest



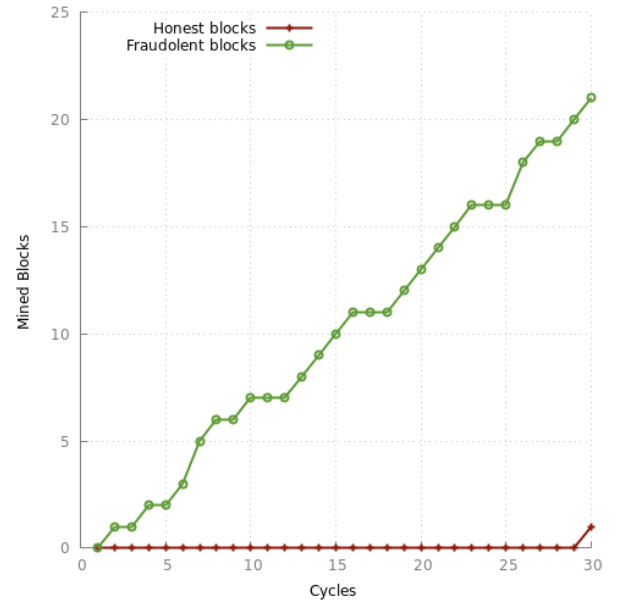
**Figure 5:** Probability of selfish miner = 0.10



**Figure 6:** Probability of selfish miner = 0.30



**Figure 7:** Probability of selfish miner = 0.50



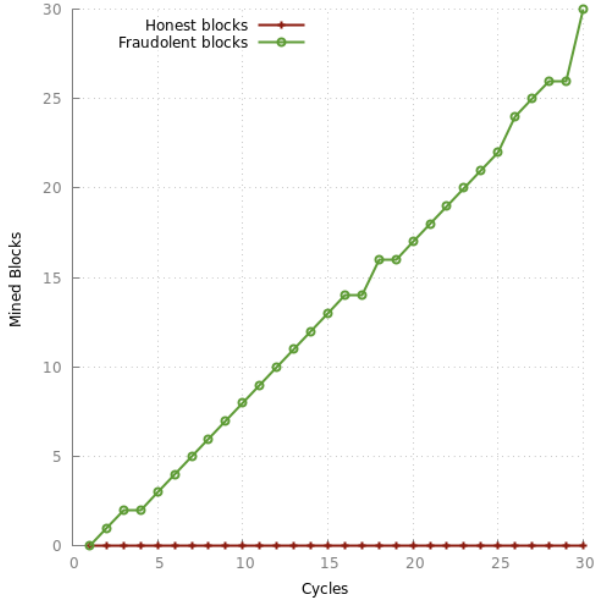
**Figure 8:** Probability of selfish miner = 0.70

miners the number of forks is small, because on average honest miners take a lot of time to mine a block. The same situation happens when the number of honest miners is much bigger than the number of selfish miners.

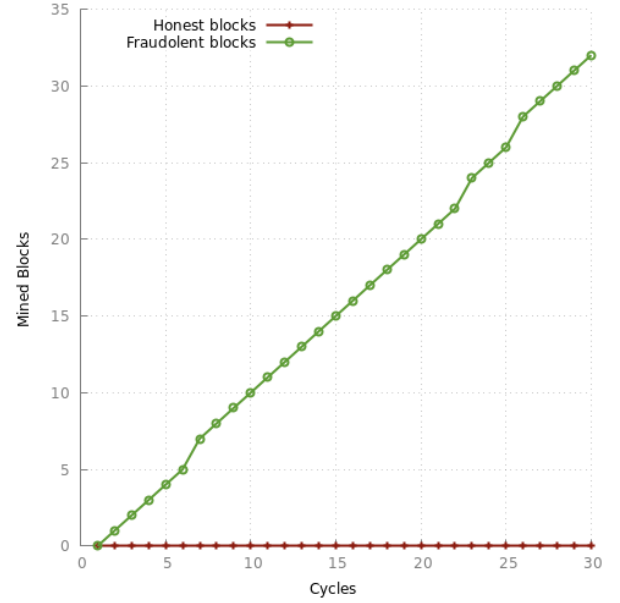
### 5.3 Mined blocks per computational power

Each type of miner (CPU, GPU, FPGA or ASIC) is assigned a different hash rate, defined as the number of hash functions computed per second. For each value of probability (of being a selfish miner), I have computed the total hash rate of both honest miners and selfish miners. Then, I computed how many blocks in the blockchain were mined by selfish miners and how many by honest miners and finally I have computed the ratio Mined Blocks / Hash Rate, where the hash rate is in billion hashes per second. The results are plotted in figure 12: the effectiveness of the hash rate of both honest and selfish miners is not constant, but varies with the number of selfish miners in





**Figure 9:** Probability of selfish miner = 0.90



**Figure 10:** Probability of selfish miner = 1.00

the network. If they are few (probability = 0.10), then the majority of blocks are mined by honest miners and so their ratio mined blocks / hash rate is high, while the same ratio for the selfish miners is close to zero because even if selfish miners have a low hash rate, they actually mine very few blocks. The selfish miners start mining more blocks per hash rate than the honest miners for a value of probability between 0.30 and 0.50. This fact is another proof of the effectiveness of the selfish mining strategy. If the value of probability increases, the ratio increases for the selfish miners while it decreases for the honest miners. Another thing to note is that the value of such ratio is maximum for the selfish miners, meaning that the adoption of selfish mining strategy leads to a better exploitation of the computational power.

## 5.4 Reward per computational power

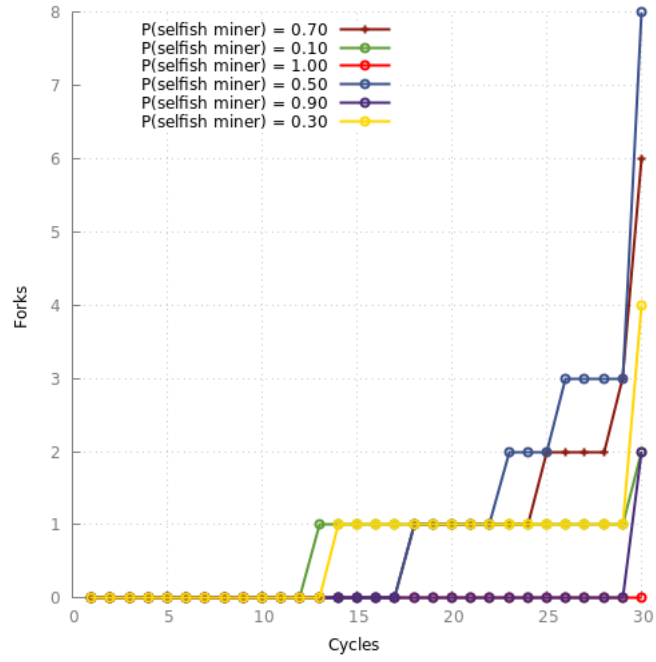
Figure 13 keeps the hypothesis of the previous paragraph to show how the ratio Reward / Hash Rate changes, on varying the probability that a miner is a selfish miner. The figure shows that for small values of such probability the honest miners earn more than the selfish miners, while if the value of such probability increases the selfish miners earn (much) more than the honest miners.

## 5.5 Number of blocks in the blockchain for different latencies of messages

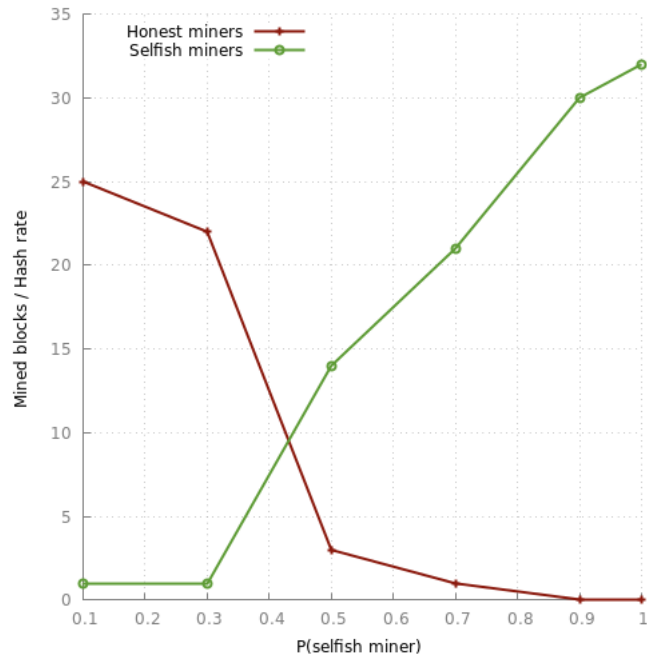
Figure 14 shows the number of blocks in the blockchain on varying the latency of messages. The base duration of each message exchange is defined with respect to the line with delay 0. This means that the line parameterized with a delay equal to 0.50 is characterized by a latency for each message which is 50% greater than the base case. This way we are for instance able to model congestion in the network. The results are quite straightforward: the blockchain is longer when the latency of each message exchange is smaller and shorter when the latency is bigger. In this simulation I don't have considered the case in which some message is lost, although I think the protocol should be reliable to message losses because of the flooding of blocks and transactions.

## 6 Conclusions

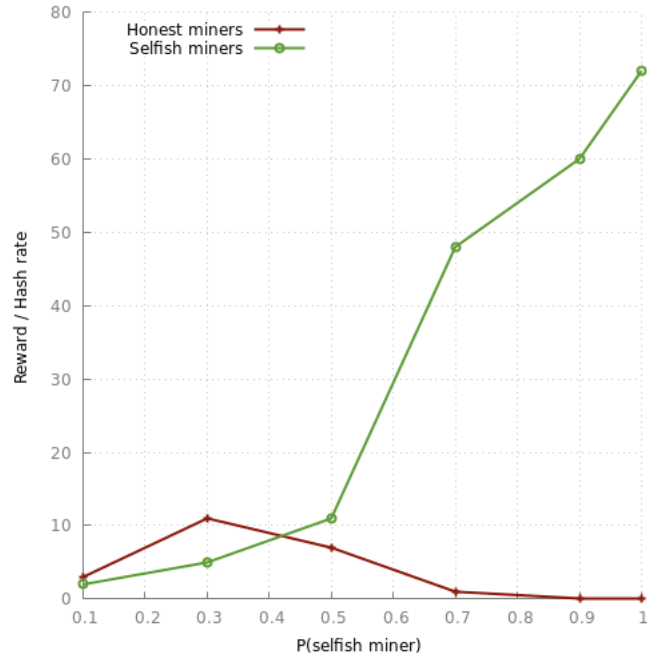
In this project I have implemented a simplified version of the Bitcoin network to evaluate the effects of the selfish mining strategy, a fraudulent mining strategy that should guarantee to the miners that adopt it a revenue, in terms of mined blocks, bigger than the one that they would expect if they had adopted the classical mining strategy.



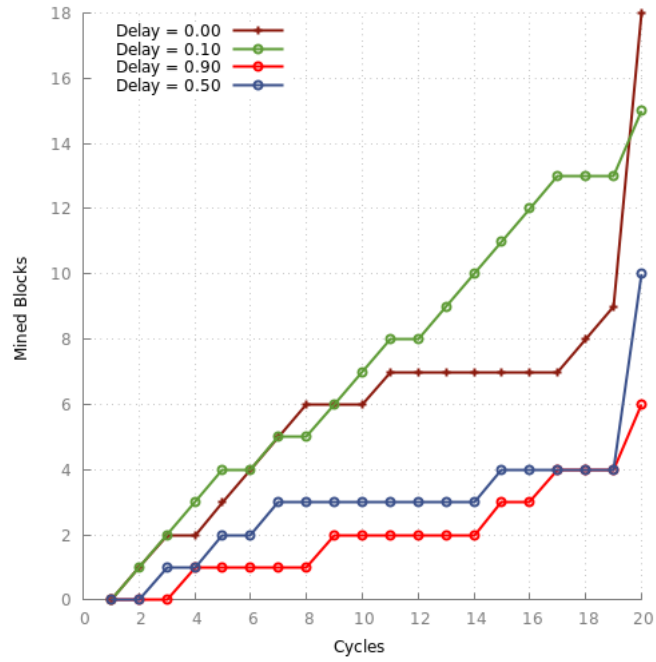
**Figure 11:** Forks in the blockchain on varying the probability that a miner is a selfish miner



**Figure 12:** Effectiveness of the hash rate of honest and selfish miners on varying the probability that a miner is a selfish miner



**Figure 13:** Reward earned by honest and selfish miners per unit of hash rate, on varying the probability that a miner is a selfish miner



**Figure 14:** Number of blocks in the blockchain on varying the latency of messages. A delay =  $0.X$  means that all the messages are delayed of  $X\%$  with respect to the simulation with zero delay.

I have defined *experiment* the single run of the program and *simulation* the repeated execution of an experiment with different configurations of parameters' values. In this document I have reported and analyzed the statistics obtained by running such a simulation.

The experimental results obtained in this setting suggest that the adoption of selfish mining strategy is effective in practice, and its effectiveness becomes clear when the number of selfish miners in the network is at least the 30% out of the total number of miners. Increasing the latency of message exchange results in a diminution of the number of blocks in the blockchain.

## 7 Limitations

There are some parts of the project that could be enhanced. I think they are the following:

- **Forks** Network nodes which are not selfish miners only support a simplified kind of fork, as already stated above. This results in a better tamper resistance and simplicity in this setting, but in a real Bitcoin network blockchain branches of arbitrary length are allowed.
- **Numbers and not probabilities** Except the last, all the plotted graphs are parameterized by the probability that a miner is a selfish miner. If this probability value is  $X$ , it means that we expect that the  $X$  percent of miners in the network are selfish miners. Since each experiment is repeated several times with the same parameters values we can really expect that the average number of selfish miners among the repeated experiments is more or less equal to the expected number. Anyway if we wish to remove any uncertainty the only way is to scan the network nodes, compute the number of miners and selfish miners and finally parameterize the graphs with this number and not with a probability value.

## 8 References

1. TinyCoin: Simulating mining strategies in a simplified Bitcoin Network
2. Majority is not Enough: Bitcoin Mining is Vulnerable