Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Stochastic Modelling and Analysis of the Bitcoin Protocol in the Presence of Block Communication Delays

Stefano Bistarelli[1] | Rocco De Nicola[2] | Letterio Galletta[2] | Cosimo Laneve[3] | Ivan Mercanti*[2] | Adele Veschetti[3]

[1]University of Perugia, Perugia, Italy
[2]IMT School for advanced studies, Lucca, Italy
[3]University of Bologna, Bologna, Italy

Correspondence
*Email: ivan.mercanti@imtlucca.it

Abstract

We analyze the protocol of the Bitcoin blockchain by using the PRISM probabilistic model checker. In particular, we (i) extend PRISM with the ledger data type, (ii) model the behaviour of the key participants in the protocol – the miners – and (iii) describe the whole protocol as a parallel composition of processes. The probabilistic analysis of the model highlights how forks happen and how they depend on specific parameters of the protocol, such as the difficulty of the cryptopuzzle and the network communication delays. Our results confirm that considering transactions in blocks at depth larger than 5 as confirmed is reasonable because the majority of miners have consistent blockchains up-to that depth with probability of almost 1. We also study the behaviour of networks with churn miners, which may leave the network and rejoin afterwards, and with different topologies.

1 | INTRODUCTION

Blockchain is an emerging technology that implements a distributed ledger on peer-to-peer asynchronous networks. It is applied in many contexts, such as the management of cryptocurrencies (Bitcoin being the most famous one[1]) and of decentralized applications (e.g., Ethereum smart contracts[2]), the implementation of voting systems[3] and the support of other application specific protocols[4,5,6].

The implementation of a distributed ledger has to solve a well-known critical issue: the inconsistency of updates that are performed by different nodes. This problem, which has been demonstrated unsolvable in 1985[7], is overcome in the Bitcoin protocol by using a probabilistic approach where the probabilities depend on the rate of the addition of new blocks – called mining – and on the delays of broadcasts. Additionally, to further reduce the probabilities of inconsistencies in the copies of the ledger, Bitcoin guarantees the so-called eventual consistency whereby the various replicas may be temporarily inconsistent in at most the last m blocks. In particular, Bitcoin considers as "confirmed" (and therefore "can be paid") every block that is at depth greater than m ($\mathsf{m} = 5$ in the current protocol[8]).

The Bitcoin protocol is complex and many researchers are actively involved in studying its properties and its criticalities. Clearly, understanding the details of the protocol is of paramount importance because overlooking some of them might introduce vulnerabilities and pave the way to attacks. For example, inconsistencies of the ledger replicas, called forks, occurring when there are two or more blocks at the same height of the ledger, may be used for rewriting the transaction histories and for letting the blockchain evolve to a wrong state. A typical example of wrong state is a state where a transaction is paid twice – called a double spending attack.

In this paper, we analyze the probabilistic behaviour of the Bitcoin protocol by using formal methods. In particular, following the approach of Pass et al.[9] and of Gramoli[8], we use an abstract model that defines the network of nodes as the parallel composition of processes – the miners – and the time needed to mine a block and to broadcast a message as an exponential distribution with a rate parameter associated to process actions. As done by Biais et al.[10], by Eyal and Sirer[11] and by Zamyatin et al.[12], we use Continuous Time Markov Chains (CTMCs) for providing a probabilistic model of our processes. To examine the probability of reaching a state of

fork in different settings, we analyse our model with a probabilistic model checker. In our model, we consider every detail of the real implementation of the Bitcoin protocol that influences the probability of reaching a state of fork. The details of the protocol that do not impact on our analysis are ignored. The overall aim of our analysis is to study the resilience of the protocol by varying some parameters such as the network topology, and the presence of nodes that leave and join the network dynamically.

The formal definition of the abstract model of Bitcoin is described using PRISM [13], a process calculus with a probabilistic and stochastic semantics. This framework has been chosen because it includes an automatic model checker that enables us to perform our analyses. Actually, in order to deal with the complex data types used by Bitcoin, namely `ledger`, `block` and `set`, and with the operations upon them, we extend PRISM and introduce the richer variant PRISM+ that enables us to naturally model these concepts. Implementing PRISM+ has required a significant programming effort in order to make the foregoing data types native. With PRISM+, we can analyze several models of the Bitcoin protocol and measure the impact on probabilities obtained by adjusting the block mining rate and the average communication delay of broadcast blocks.

As an initial step, we assess the coherence of our model by verifying that the probabilities of mining a new block within a given amount of time and the probability of having a fork are both in full agreement with the values available from the literature [14]. In particular, we confirm that the probability of a fork strictly depends on the broadcast delay and on the difficulty of the cryptopuzzle, i.e. the difficulty of the computational problem that miners must solve in order to add a new block to the ledger. We also corroborate the statement [8] that waiting for at least 6 confirmations before considering a transaction as confirmed in the ledger is reasonable, because at that time the majority of miners has a consistent blockchain with probability $1 - 10^{-12}$.

After validating our model, we study the trade-off between the security guarantees and the difficulty of the cryptopuzzle. More precisely, we analyze the variability of the probability of reaching fork states when the speed of the mining process increases, that is when the difficulty of the cryptopuzzle decreases. We observe that the speed of mining can be boosted by decreasing slightly the difficulty level at the cost of an almost irrelevant increase of the probability of forking. Obviously, the easier is the cryptopuzzle, the faster the entire system mines a block. We also show that a good balance between speed and safety can be obtained with an average mining rate equal to $1/500$. In fact, with this rate, the process of mining a block is faster than in Bitcoin ($1/600$ [14,10]), but the probability of reaching a state of fork is not much higher.

We also study a network with churning nodes, i.e. nodes that can leave and rejoin the network. In particular, we analyze how the presence of this kind of node affects the probability of mining new blocks and, consequently, the probability of forks. Our analyses show that this probability is lower when there are churning nodes in the network. The same considerations stays valid for forks. This is due to the fact that, in our network model, each node is connected to every other node, thus the presence of churning nodes does not lead to message losses.

Finally, we consider several network topologies (linear topology, ring topology, tree topology and fully connected topology) and analyze them in order to estimate how the connections between nodes can affect the likelihood of a fork. Our simulations show that the probability of inconsistency increases when new blocks, instead of being forwarded to all the nodes of the network directly, are transmitted only to a subset of them. Thus, the smaller is the subset of the receiving nodes per miner, the higher is the probability of having a fork. The different topologies have instead no impact on the mining process because the rate of communications (of blocks) is much higher than the mining rate.

We conclude by remarking that our PRISM primitives (`ledger`, `block` and `set`) are generic, namely they are not tied to a particular protocol and can be used for modelling and analysing other blockchain protocols, such as proof of stake and its variants.

The rest of the paper is structured as follows. Section 2 contains an overview of Bitcoin and its (Proof of Work) consensus algorithm and recap the main constructs of the PRISM language. Section 3 presents PRISM+, our extension of PRISM with the new data types, `ledger`, `block` and `set`. Our model of Bitcoin is presented in Section 4. Section 5 contains our simulations and compares the results we obtain when considering different type of networks. Section 6 compares our proposal with other works in the literature and Section 7 draws some conclusions and discusses possible future work.

2 | BACKGROUND

In this section we present an overview of the Bitcoin consensus protocol and provide an introduction to the PRISM framework that will be used for the simulations and the analyses.

2.1 | The Bitcoin Protocol

Bitcoin is a peer-to-peer asynchronous network whose nodes host a ledger recording economic transactions that are grouped into blocks. The ledgers are trees of blocks with a pointer (handle) to a leaf block at maximal depth; the blockchain is the sequence of blocks from

the handle to the root block, (genesis block). Blocks are created by special nodes of the network – the miners – and contain a number of information, including transactions and a pointer to the current handle of miner's ledger.

Once a block has been mined, the miner (i) adds the block to its own ledger (therefore the depth of the ledger increases and the handle is updated); and (ii) broadcasts it to all the connected nodes of the network. Every node receiving the new block updates its local copy of the ledger by inserting the block in the right position and, if necessary, it also updates its own handle. If the block cannot be connected to the ledger (because, due to network delays, a previous block has not been delivered) then it is added to the local set of the miner and will be inserted afterwards (orphan blocks).

Because of asynchrony, it may happen that two nodes mine and broadcast a block almost concurrently, yielding different ledgers with different handles (and, therefore, with different blockchains). This phenomenon, called fork, is at the core of the inconsistencies of Bitcoin and, to overcome this problem, the protocol uses a probabilistic algorithm. In particular, Bitcoin has a technique to regulate the mining of blocks, called Proof of Work (PoW). According to PoW, miners can add a block only if they solve a computational problem. Technically, the problem consists of finding a number (a nonce) which is inserted into the block header. The block header is then hashed and if the numerical value of the hash is less than a predefined condition, which is called target, then the miner is said to have mined the block. The only way to find such a nonce is through an exhaustive search. The finding of suitable nonce values can be modelled as a Bernoulli trial with a probability of $\mathsf{L}/2^{256}$ of being successful, where L is the target. The time needed to mine a new block depends on the difficulty of the PoW and the hashing power of the miners. Clearly, the faster miners are, that is the more of computational power they own, the higher is the probability of forks and, thus, the more likely is the inconsistency between miners. For this reason, the Bitcoin PoW difficulty is determined by a moving average targeting a certain number of blocks per hour. If they are generated too (slowly) rapidly, the difficulty is (decreased) increased as shown by Nakamoto[1]. The current protocol modulates PoW in order to have 6 blocks per hour on average.

To further reduce the probability of inconsistencies, Bitcoin also uses the so-called eventual consistency (also known as n-consistency[9]). This is a weak version of consistency, according to which the protocol considers consistent those ledgers with the corresponding blockchains equal up to the last few blocks.

In particular, Bitcoin considers both transactions and miner's rewards in blocks at depth greater than 5 as confirmed[15].

A goal of this paper is to study how the probabilities of inconsistencies do change after adjustments to the mining speed or to the broadcast delay that may vary according to the network topology. To this aim, we model delays by means of rates of actions and the process of finding a solution to the PoW problem by means of an exponential distribution[16]. In doing this, we abstract out those features of Bitcoin that are not relevant for our analysis, namely the actions performed by the miners to solve the cryptopuzzle, the usage of cryptographic primitives, and the rewards assigned to those who mine a new block. Moreover, we do not take into account the adjustments in the difficulty and changes in the running average of the global hashing rate since they do not affect the outcome of our analyses.

2.2 | The PRISM Language

PRISM[13] is a probabilistic model checker that inputs a formal description of a system and computes the likelihood of the occurrence of certain events. The formal description of systems in PRISM is given by means of a process algebra that allows one to specify interacting modules. A module contains a number of local variables whose values at any given time constitute its state. The behaviour of a module is described by a set of commands whose basic form is

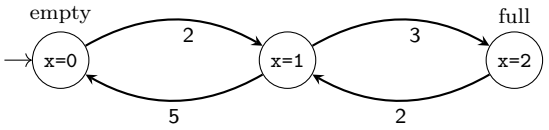
```
[a] g -> rho : update ;
```

In this command, the guard `g` is a predicate over all the variables in the model (including those belonging to other modules). The update `update` describes a transition that the module can make if the guard is true and it is specified by giving the new values of the variables in the module, possibly as an expression formed from other variables or constants. PRISM uses the prime notation to specify updates, e.g. `x' = x+1` means that, in the next state, `x` takes the value stored in it plus one. The expressions `rho` are used to assign probabilistic information to the transitions – in the following they are called rates because we will employ stochastic semantics. The name `a` is an action: when it is present in commands of different modules then the corresponding commands must be executed at the same time, i.e. they synchronize, and the overall rate is the product of the corresponding rates. Finally, a module may also contain labels that are a way of identifying sets of states that are of particular interest.

A PRISM system consists of modules and global variables; its global state is determined by the local state of all modules, together with the values of the global variables. We refer to Kwiatkowska et al.[17] for a full account of the formalism. The following listing reports a PRISM module defining a two-items queue. The variable `x` encodes the state, which has three possible values: `0` (the queue is empty), `1` (the queue has one element) and `2` (the queue is full, i.e. it has two elements). For example, from the state `x=1` the system can evolve either in `x=2` with rate `3` and in `x=1` with rate `5`.

```
1  module C
2      x : [0..2] init 0;
3
4      [] x=0 -> 2 : (x′ = 1);
5      [] x=1 -> 3 : (x′ = 2);
6      [] x=1 -> 5 : (x′ = 0);
7      [] x=2 -> 2 : (x′ = 1);
8
9  endmodule
10
11 label "full" = x = 2;
12 label "empty" = x = 0;
```

We observe that the above module `C` may be also described by a transition system where states are those of the system and transitions are labelled by rates. That is



PRISM supports different kinds of probabilistic formalism. In this paper we focus on Continuous Time Markov Chains (CTMC) models, which are transition systems (as the one above) whose semantics are defined by rates. In particular, if the rate of a transition from a state $\mathtt{s}$ to $\mathtt{s}'$ is $\mathtt{r}$ then the probability of moving from $\mathtt{s}$ to $\mathtt{s}'$ within $\mathtt{t} \geq 0$ time units is $1 - \mathtt{e}^{-\mathtt{r}\cdot\mathtt{t}}$, that is rates are used as parameters of an exponential distribution. Note that, the higher the rate, the higher the probability to leave $\mathtt{s}$ in a given time. This is the exact abstraction used by Nakamoto [1] to analyze the Bitcoin protocol, where the parameter $\mathtt{r}$ depends on the miner hashing power and the difficulty level of the crytopuzzle, and the abstraction discussed by Decker and Wattenhofer in [14] where an exponential distribution approximates the probability of delivering blocks across the Bitcoin network within $\mathtt{t}$ time units. When a CTMC state has several exiting transitions, e.g. the above state `x=1`, then the probability of choosing one transition depends on the rates of the corresponding transitions – this is known as race condition. For example, if $\mathtt{r}_1, \cdots, \mathtt{r}_n$ are the rates of transitions exiting from a state $\mathtt{s}$ (and entering on pairwise different states), then the probability of taking a transition with rate $\mathtt{r}_i$ is $\mathtt{r}_i/\mathtt{R}$, where $\mathtt{R} = \mathtt{r}_1 + \cdots + \mathtt{r}_n$. In this setting, the probability of moving from $\mathtt{s}$ in $\mathtt{t}$ time units is $1 - \mathtt{e}^{-\mathtt{R}\cdot\mathtt{t}}$. Since, in Markov chains, the events are independent from the previous events in the history (the Markov property), the probability of reaching a state in a given time $\mathtt{t}$ is a function of the product of the probabilities in the intermediate states (this function is not simple because one has to consider all the possible partitions of $\mathtt{t}$ and all the possible paths to reach the state from the initial step).

In the PRISM framework, properties of CTMC models are expressed in Continuous Stochastic Logic (CSL) [18,19,20], which is an extension of temporal logic with a probabilistic operator. In particular, in this contribution we will analyze formulas of the form

```
P=?[F<=t property]
```

that return the probability that the `property` is true in a state of the model within `t` time units (starting from the initial state). For example, if we want to express the probability that the two-items queue is "full" within `t` time units, we will write

```
P=?[F<=5 "full"]
```

To compute this probability, PRISM performs Statistical Model Checking. That is, since models may bear infinite sets of paths (in general and in this case, in particular), PRISM does not perform an exhaustive exploration of the state-space. Rather, it imposes a maximum path length to avoid the need to generate excessively long paths and returns the probability of the formula in the finite model. The core idea of the approach is to conduct some simulations of the system, monitor them, and then decide whether the system satisfies the property or not with some degree of confidence. All the simulations in this article have been driven with a Confidence Interval (CI) method for Statistical Model Checking that gives an approximate value for a `P=?` property, based on a confidence level and the number of samples generated. For instance, if we compute the probability that the two-items queue is "full" within 5 time units in PRISM with the CI method, we obtain that `P=0.957`. In the simulations presented in Section 5, the confidence level is set to 99% and the samples to be generated are 100000. In the caption of the figures we always show the `t` time units (bound time) considered for each property.

3 | THE EXTENDED PRISM LANGUAGE

To have a faithful abstraction of the Bitcoin protocol we extend the PRISM language with three dynamic data types: `block`, `ledger` and `set`. We call the extended language PRISM+ and, in this section, we overview its main features. Our extension has been implemented

in a prototype that the reader can find on-line at https://github.com/adeleveschetti/bitcoin-analysis together with all the data of our simulations, and the instructions for the installation and the use of the tool.[A]

3.1 | Blocks

As discussed in Section 2, a Bitcoin block records information items such as the transactions, the nonce, a timestamp, the Merkle root and a pointer to its parent. In our model, we ignore the block hash value, because we consider all blocks and all transactions valid. In particular, our blocks are either `genesis` or terms of the form $(\mathtt{m}^{\mathtt{n}};\rho)$, where $\mathtt{m}^{\mathtt{n}}$, called name, is such that `m` is a miner's name and `n` is a unique numeric label. The term ρ, called parent, is the name of another block to which $(\mathtt{m}^{\mathtt{n}};\rho)$ points. For instance, $(\mathtt{m3}^{0};\mathtt{m4}^{7})$ denotes a block named $\mathtt{m3}^{0}$, which is the first block created by the miner `m3`, and whose parent is the block named $\mathtt{m4}^{7}$, which is the seventh block created by the miner `m4`. The block `genesis`, called genesis block is the root of every ledger.

We introduce the following operation for creating blocks:

- `createBlock(m, n, L)`, which returns a block $(\mathtt{m}^{\mathtt{n}},\rho)$, where ρ is the handle of the ledger `L` (see below).

3.2 | Ledgers

The `ledger` data type, noted `L`, `L'`, $\cdots$, is a pair $\langle \mathtt{T};\mathtt{p}\rangle$ where `T` is a tree of blocks and `p` is the name of a leaf block at maximal height, called the handle of `L`.

If there are several leaf blocks at maximal depth, the pointer is set to the first received leaf at maximal depth, precisely following the description of protocol presented by Nakamoto [1]. The height of a block `b` is the length of path that connects `b` to the genesis block. The root of `T` is the genesis block. Given a ledger `L`, the corresponding blockchain is the sequence of blocks which starts from the handle and reaches the genesis. When a miner with a ledger $\langle \mathtt{T};\mathtt{p}\rangle$ receives a block `b`, it adds `b` to `T`. Note that this is not always possible because the parent of `b` may be not in `T`; in this last case `b` is added to the local set of the miner and is inserted into the blockchain afterwards. In case the depth of `b` in `T` is higher than the one of the block pointed by `p`, the miner updates `p` to point to `b`. When a miner mines a new block, the corresponding parent is set to `p` and `p` is updated accordingly. We define the following operations for the new data types:

- `addBlockLedger(L,b)`: the first argument is a ledger $\mathtt{L}=\langle \mathtt{T};\mathtt{p}\rangle$ and the second is a block `b`. The operation adds `b` to `T` if the parent of `b` is in `T`. When `b` is at higher depth than the handle `p`, then the handle is updated with the name of `b`;
- `canBeInserted(L,b)`: checks if the block `b` can be inserted in the ledger `L`. Thus, if the parent of `b` is already in tree of blocks of `L`, the function returns true, false otherwise.

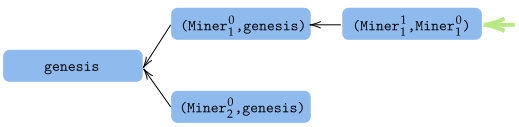


Figure 1 An example of Ledger.

In Figure 1, we depict the ledger

$$\mathtt{L}=\langle\{\mathtt{genesis},(\mathtt{Miner}_1^0;\mathtt{genesis}),(\mathtt{Miner}_1^1;\mathtt{Miner}_1^0),(\mathtt{Miner}_2^0;\mathtt{genesis})\};\mathtt{Miner}_1^1\rangle$$

that has two blocks created by $\mathtt{Miner}_1$ and one created by $\mathtt{Miner}_2$. The handle, which is represented by a green arrow, is the block whose name is $\mathtt{Miner}_1^1$. The blockchain of `L` is the sequence of blocks $(\mathtt{Miner}_1^1;\mathtt{Miner}_1^0)\cdot(\mathtt{Miner}_1^0;\mathtt{genesis}^0)\cdot(\mathtt{genesis}^0;\mathtt{genesis}^0)$. We notice that the block created by $\mathtt{Miner}_2$ is a stale block.

A basic notion of ledgers is that of fork:

Definition 1. Let $\mathtt{L}_1=\langle \mathtt{T}_1;\mathtt{p}_1\rangle$, …, $\mathtt{L}_\mathtt{n}=\langle \mathtt{T}_\mathtt{n};\mathtt{p}_\mathtt{n}\rangle$ be a set of ledgers and let `m` be the maximal height of the handles $\mathtt{p}_1,\cdots,\mathtt{p}_\mathtt{n}$. Let also $\mathtt{L}_{\mathtt{i}_1}$, …, $\mathtt{L}_{\mathtt{i}_\mathtt{k}}$ be the ledgers in the above set with the handle at height `m`. The set $\mathtt{L}_1$, …, $\mathtt{L}_\mathtt{n}$ has a fork of length $\mathtt{m}-\mathtt{h}$, where `h` is the length of the maximal common suffix of the blockchains of $\mathtt{L}_{\mathtt{i}_1}$, …, $\mathtt{L}_{\mathtt{i}_\mathtt{k}}$.

To better understand Definition 1, consider the two generic ledgers in Figure 2. These ledgers have a fork of length 1 because they differ the handles, i.e., the blocks pointed by the green arrow. PRISM+ will compute the probability that a fork of a certain length happens.

[A] A docker image with our prototype is also available at https://hub.docker.com/repository/docker/meivan/bitprism

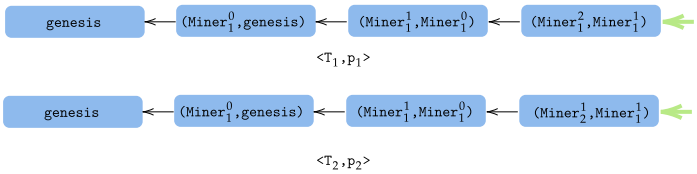


Figure 2 Two generic ledgers in a state of fork of length 1.

To this aim, we have implemented the operation

- `calculateFork`($\langle$`T`$_1$`;p`$_1\rangle$,...,$\langle$`T`$_n$`;p`$_n\rangle$): it takes as input a set of ledgers and returns the length of the fork; it returns 0 if there is no fork.

The function `calculateFork` compares the handles of the ledgers given as input and keeps the blockchains with different handles at maximal height. Then `calculateFork` uses an auxiliary function that recursively iterates over the resulting blockchains without the leading block: the length of the resulting fork is obtained by summing 1 to the length obtained from the previous recursive invocation.

3.3 | Sets

The `set` data type is implemented as a list of blocks without duplicates. In our Bitcoin model, this data type is used to maintain, for each miner, the collection of blocks to be added to the blockchain. The data type has the following operations:

- the extraction operation `extractBlockSet(S)` which returns a block randomly extracted from the set `S`;
- `addBlock(S,b)` which takes as input a set `S` and a block `b`; it returns `S` $\cup$ `b`. When `b` $\in$ `S`, `addBlockSet(S,b)` $=$ `S`;
- `removeBlock(S,b)` which returns the set `S` $\setminus$ `b`;
- `isEmpty(S)` which returns `true` if the set `S` is empty, `false` otherwise.

4 | THE BITCOIN MODEL

In our model, a Bitcoin system is the result of the parallel composition of `n` Miner processes, `n` Hasher processes and a process called Network. Hasher processes model the attempts of the miners to solve the cryptopuzzle, while the Network process model the broadcast communication among miners. At the beginning, we shall assume that miners are connected through a network guaranteeing broadcast. Later on, we shall consider other topologies (see Section 4.3); the actual architecture is illustrated in Figure 3. The abstraction also uses an auxiliary process, called Global, that computes the length of forks, see Section 5.

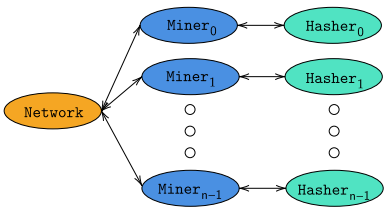


Figure 3 The Bitcoin model architecture.

As already said, in order to abstract out the solution of the cryptopuzzle and the broadcast of new blocks, we use rates.

4.1 | General Model

For the sake of clarity, we present a simplified version of the PRISM+ code implementing our processes. The actual abstraction, the analyzed properties with tests and the instructions for the use of the library are available on the online repository.

```
1  module Hasher_i
2     Hasher_i_STATE : 0;
3
4     [win_i] (Hasher_i_STATE=0) -> mR : Hasher_i_STATE'=0 ;
5     [lose_i] (Hasher_i_STATE=0) -> lR : Hasher_i_STATE'=0 ;
6  endmodule
7
8  module Miner_i
9     Miner_i_STATE : [Mine,Winner,Lost,Add,Move] init Mine;
10    b_i : block (Miner_i^0;genesis^0);
11    L_i : ledger ⟨{ genesis };genesis ⟩;
12    c_i : [0..100] init 0;
13    setMiner_i : set [];
14
15    [win_i] (Miner_i_STATE=Mine) ->
16                  hR_i : (b_i'=createBlock(Miner_i,c_i,L_i))&(c_i'=c_i+1)&(Miner_i_STATE'=Winner);
17    [lose_i] (Miner_i_STATE=Mine) -> hR_i : (Miner_i_STATE'=Lost);
18
19    [addBlock_i] (Miner_i_STATE=Winner) ->
20                  1 :(L_i'=addBlockLedger(L_i,b_i))&(Miner_i_STATE'=Mine);
21
22    [] (Miner_i_STATE=Lost) & !isEmpty(set_i) ->
23                  1 : (b_i'=extractBlock(set_i))&(Miner_i_STATE'=Move);
24    [] (Miner_i_STATE=Lost) & isEmpty(set_i) ->  1 : (Miner_i_STATE'=Mine);
25    [] (Miner_i_STATE=Lost) & !isEmpty(setMiner_i) ->
26                  1 : (b_i'=extractBlock(setMiner_i))&(Miner_i_STATE'=Add);
27    [] (Miner_i_STATE=Lost)& isEmpty(setMiner_i) -> 1 : (Miner_i_STATE'=Mine);
28
29    [removeBlock_i] (Miner_i_STATE=Move) ->
30                  1 : (setMiner_i' = addBlock(setMiner_i, b_i))&(Miner_i_STATE'=Mine);
31
32    [] (Miner_i_STATE=Add)&(canBeInserted(L_i,b_i)) ->
33                  1 : (L_i'=addBlockLedger(L_i,b_i)&(setMiner_i'=removeBlock(setMiner_i,b_i))
34                      &(Miner_i_STATE'=Mine);
35    [] (Miner_i_STATE=Add)&(!canBeInserted(L_i,b_i)) ->  1 : (Miner_i_STATE'=Mine);
36 endmodule
```

Listing 1: Simplified model of the Hasher and a miner.

A Hasher process is defined in Listing 1 (lines 1 to 6). It represents the PoW algorithm performed by miners: those miners who want to solve the cryptopuzzle synchronize with the Hasher which "answers" telling them if they succeeded or not. The Hasher consists of two transitions: the first one with action `[win_i]` and rate `mR` is triggered when the synchronizing miner finds a solution for the PoW (`mR` is taken such that $0 < \texttt{mR} < 1$); the second one with action `[lose_i]` and rate `lR` ($\texttt{lR} = 1 - \texttt{mR}$) is triggered when the synchronizing miner does not find a solution to PoW. In both cases the Hasher process makes a silent action.

A Miner, described in Listing 1 lines 8 to 36, has a ledger, called `L_i`, a set containing the blocks to be added to the ledger, called `setMiner_i`, a block `b_i` and a integer `c_i`. The variable `b_i` is used to store the block the Miner creates and to store the newly extracted block from the set. The integer `c_i` is a counter whose value ranges between 0 and 100 (initially is zero) and which keeps track of the number of blocks created by the Miner, so that we can assign unique names to blocks.

The Network process is defined in Listing 2. It contains a set of blocks `set_i` for each `Miner_i` that represents the messages to be delivered to the miner. The set `N_i`, in the case of the broadcast topology, is equal to $\{0, \cdots, i-1, i+1, \cdots, n-1\}$, e.g. the indexes of the miners to

whom a block must be sent (line 9). Also, the Network process contains a transition for each Miner to model sending and receiving messages.

```
1  module Network
2      n : numberOfMiners
3
4      for i from 0 to n-1:
5              set_i : set [];
6              N_i : set [];
7
8      for i from 0 to n-1:
9          [addBlock_i] ->  r_b : foreach k in N_i { set_k'=addBlock(set_k,b_i); };
10
11     for i from 0 to n-1:
12         [removeBlock_i] -> 1 : set_i' = removeBlock(set_i,b_i);
13 endmodule
```

Listing 2: Simplified model of the Network.

More precisely, the Network synchronizes with the Miner who won the PoW (in the winner state) using the `addBlock_i` action. As a result of this synchronization, Network updates the sets of blocks of the miners contained in the set `N_i`. [B]

Below we describe in detail how our processes abstract the Bitcoin protocol. As the reader can observe from the code in Listing 1, `Miner`'s state is initially set to `Mine`. In this state it can synchronize with `Hasher_i` using either the `win_i` or `lose_i` actions. As already said, this synchronization abstracts the cryptopuzzle solution. Note that the time needed for the creation of a new block at miner i is a random number sampled from an exponential distribution with rate P proportional to the ratio of the difficulty of the PoW problem and the hashing power of the miner. Therefore, the chosen action, `win_i` or `lose_i`, depends on the difficulty of the problem (represented by the hasher values `mR` and `lR`) and on the hashing power of the miner (represented by `hR_i`). Since the rate of a synchronization is equal to the product of the rates of the two actions, the rate of mining a new block is `mR`×`hR_i`, which corresponds to the parameter λ_{m_i} introduced in the previous section. Whereas the rate of loosing the competition is `lR`×`hR_i`. If the miner wins, it changes its status in `Winner` (lines 15 and 16), updates its ledger and sends the new block to the Network (action `addBlock_i` at lines 19 and 20) in order to forward it to the other miners (i.e. updating other miners' sets with the new block). If the miner loses, its status becomes `Lost` (line 17) and it checks for new blocks in the Network process with a certain rate $\mathtt{r_b}$, which simulates the latency of the network and corresponds to the product between the rate 1 (for the action of the Miner) and the rate $\mathtt{r_b}$ of the Network action (lines 22 and 23 and line 9 of Listing 2). If there are new blocks, the miner chooses randomly one of them (with the operation `extractBlock()`). This random choice simulates the delay due to the topology of the network. In our model, the rate $\mathtt{r_b}$ and the random selection of blocks from the sets simulates the communication delay of messages in the Bitcoin network. Then, the state of the Miner becomes `Move` and the Miner adds to its local set `setMiner_i` the block `b_i` (lines 29 and 30). Moreover, the Miner synchronizes with the process Network with action `removeBlock_i`. The Network removes the block `b_i` from the set of the Miner `set_i`. Then the state of the Miner becomes `Mine`. Otherwise, the Miner can try to take a block from its local set (lines 25-26). A block is randomly extracted from the local set `setMiner_i`. If the block taken from the local set can be added in the ledger (which means that the function `canBeInserted(L_i,b_i)` returns `True`), the Miner adds the block to its ledger and removes it from the local set (lines 32-34). Finally, its status is set to `Mine` and the process starts again. Otherwise, the block is not removed from the local set and the process starts again (line 35). If both the local set and the set stored in the Network process are empty, the Miner does nothing and its status returns `Mine` (lines 24 and 27). The time spent in performing these actions is simulated by the rate 1. This rate is much higher than the other rates (which are numbers in the $[0, 1]$ interval) because it corresponds to local management operations of the Miner. Therefore, the probability that a Miner tries to add a received block in its ledger is higher than the probability of receiving or mining a new block. It is worth noticing that a block is added in the correct position of the ledger, even if it is a stale block. In our model stale blocks are represented as valid blocks which are not part of the blockchain. In contrast, an orphan block is modeled as a block received by a miner, but that does not have its entire ancestry (yet) in the local ledger and thus cannot be added. So an orphan block is not added to the ledger and is left in the local set `setMiner_i`.

[B] Actually, the set `N_i` is not present in the PRISM+ model. The actions are replicated for all the Miners without the for loop.

4.2 | Churn Nodes

Nodes that may leave the Bitcoin network and rejoin after some time are called churn nodes. As described by Motlagh et al.[21,22], while a node is away from the network, other active nodes continue processing transactions, mining and adding blocks to their respective blockchains. When a node rejoins the network, its ledger is out of date and needs to be updated before the node can take part in network activities. Therefore, the first action to be taken after rejoining is to download all blocks that were added to the set of the Network during its sleep. When the blocks have been downloaded, the miner can start adding them to the ledger. In the model of Listing 1, this is performed by transiting to the state `Mine`. In order to model churn miners, we define a controller process that awakes and shuts down miners following a given policy explained below. The uncertainty is modelled by rates and the controller consists of a sequence of states that alternate `awake` and `sleep` synchronizations with the corresponding miner. Listing 3 shows a controller `Controller_i` for a miner `Miner_i` with with 11 states.

```
1  module Controller_i
2    Controller_i_STATE : [s0,s1,. . . , s10] init s0;
3
4    [sleep_i] (Controller_i_STATE = s0) -> r_i0 : Controller_i_STATE' = s1 ;
5    [awake_i] (Controller_i_STATE = s1) -> r_i1 : Controller_i_STATE' = s2 ;
     ...
12   [sleep_i] (Controller_i_STATE = s8) -> r_i8 : Controller_i_STATE' = s9 ;
13   [awake_i] (Controller_i_STATE = s9) -> r_i9 : Controller_i_STATE' = s10 ;
14 endmodule
```

Listing 3: Model of a controller with 11 states.

Note that the controller is a finite state system that, when the number of states are even, will leave the corresponding miner active forever; when the number is odd, it will leave the miner inactive forever. It is easy to define alternative controller with cyclic behaviours.

```
1  module Miner_i
2     Miner_i_STATE : [Mine,Winner,Lost,Add,Move,Update,MoveUpdate,Sleep] init Mine;
3     b_i : block (Miner_i^0;genesis);
4     L_i : ledger ⟨{ genesis };genesis ⟩;
5     c_i : [0..100] init 0;
6     setMiner_i : set [];
7
8     [sleep_i] (Miner_i_STATE = Mine) -> 1 : (Miner_i_STATE' = Sleep);
      ...
32    [addBlock_i] (Miner_i_STATE=Update) & !isEmpty(set_i) ->
33                  1 : (b_i'=extractBlock(set_i))&(Miner_i_STATE'=MoveUpdate);
34
35    [removeBlock_i] (Miner_i_STATE=MoveUpdate) ->
36                  1 : (setMiner_i'=addBlockSet(setMiner_i, b_i))&(Miner_i_STATE'=Update);
37
38    [] (Miner_i_STATE=Update) & isEmpty(set_i) -> 1 : Miner_i_STATE'= Mine ;
39
40    [awake_i] (Miner_i_STATE = Sleep) -> 1 : (Miner_i_STATE' = Update);
41 endmodule
```

Listing 4: Simplified model of a dynamic miner.

The churn miner of Listing 4 extends the one of Listing 1 with three additional states: `Sleep`, `MoveUpdate`, and `Update`.

As before, the initial state of this Miner is `Mine` where, in addition, it may synchronize with the controller (action `sleep_i`) and, after a certain amount of time, modeled by the exponential parameter `r_i0`, the Miner state becomes `Sleep` (line 8). In this state, the miner may synchronize with the controller again (action `awake_i`) and its state becomes `Update` (line 32). In the `Update` state, the Miner synchronizes with the `Network` process and extracts all the blocks from the corresponding set in `Network` by moving them into its local set `setMiner_i`

(lines 35-36). When `set_i` becomes empty (line 38) the Miner state is set to `Mine` and the Miner can resume its standard behaviour, which is the one defined in Section 4.1 in Listing 1 (lines 15-35).

In our simulations we consider three controllers: one with four states (so sleep-awake-sleep synchronizations, the first sleep has a very high rate, therefore the corresponding miner goes asleep immediately), the second with two states (one sleep synchronization only: when the miner shuts down, it will be down forever) and the third one with five states (sleep-awake-sleep-awake synchronizations). Additional experiments with larger number of churn miners and with cyclic behaviour are left to future work.

4.3 | Network Topologies

We modeled several kinds of network topologies and analyzed how they affect the likelihood of a fork. Network topology refers to how nodes are connected with each other and transmit new blocks. We study three network topologies: the ring topology, the tree topology and the linear topology. The three topologies have been modeled by changing only the Network process; Hasher and Miner are those defined in Section 4.1.

```
1  module Network
2      n : numberOfMiners
3
4      for i from 0 to n-1:
5              set_i : set [];
6              N_i : set [];
7
8      for i from 0 to n-1:
9          [addBlock_i] ->  r_b : foreach k in N_i { set_k'=addBlock(set_k,b_i); };
10
11     for i from 0 to n-1:
12         [removeBlock_i] -> 1 : set_i' = removeBlock(set_i,b_i)
13                               & foreach k in N_i { set_k'=addBlock(set_k,b_i); };
14 endmodule
```

Listing 5: Simplified model of the Network.

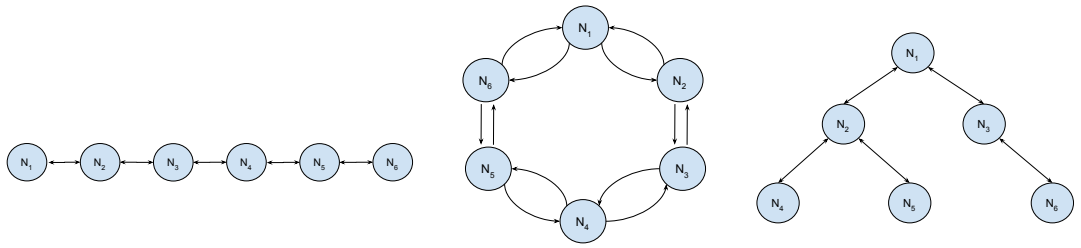
Listing 5 shows the modified code for the Network process. As the reader can observe, the code is the same as of the one presented in the Section 4.1, except for the line 13. In particular, we modified the set `N_i` containing, for each node, the set of nodes to whom the new blocks have to be forwarded. When a Miner extracts a block received by the Network, the block is forwarded to the nodes contained in the set `N_i`. This set is defined according to the topology as follows:

- Linear topology (Figure 4a): `N_i` contains the previous node and the next one (except for the terminal nodes where `N_i` are singletons). For instance, for miner `Miner_i`, assuming $i \neq 0,n$, `N_i = { Miner_i-1, Miner_i+1 }`, while `N_0 = { Miner_1) }` and `N_n = { Miner_n-1 }`.
- Ring (Figure 4b): the set `N_i` contains the previous and the next node for each miner. Thus, for every miner `N_i = { Miner_((i-1)%n), Miner_((i+1)%n)}` .
- Tree topology (Figure 4c): every `N_i` contains the parent node and the children node, except for the root of the tree and the leaves. Roots have only children nodes; leaves have only the parent.

The above topologies have been selected because they are the simplest to realize in practice. The goal is to compare the resilience to forks of these topologies with respect to the broadcast topology, (where every miner forwards the block to all the other miners). Henceforth, one can choose the best topology according to the preferred trade-off between risks of forks and connection costs.

5 | STOCHASTIC ANALYSIS

In this section, we report the results of the simulations of our model. We used PRISM+, presented in Section 3, to analyze the behavior of Bitcoin in different settings. The first analysis validates our model with respect to the real Bitcoin network; the second one studies the trade-off between the security and the efficiency/scalability of the network, i.e. we study the interdependence between the probability of



(a) Linear topology. (b) Ring topology. (c) Tree topology.

Figure 4 Network topologies.

reaching a fork of length `k` and the difficulty of a cryptopuzzle. The last two analyses study the time needed to mine a new block and the probability of reaching a fork of length `k` for a network with churn miners and in networks using different kind of topologies, respectively.

We always assume that all miners work honestly, for example, they never try to mine new blocks and attach them in internal nodes of the ledger, as would occur in a double spending attack or in a block withholding attack [11]. All the simulations have been carried out on a Virtual Machine with 8 VCPU and 64 GB RAM.

Following Section 2, we define the properties of interest in the stochastic temporal logic CTL. For example, the formula

```
P=?[F<=T "winner"]
```

defines the probability that some miner mines a new block within the first `T` time units. Thus, when checking the above property, PRISM must check whether `F<=T "winner"` is true for each path. This is the formula we use to assess the coherence of our model with respect to Bitcoin – see Section 5.1. Another example is the formula that checks the occurrence of forks in ledgers. To formalize this formula we introduced in our model a suitable module to compute forks, called `Global`, whose code is reported in Listing 6.

```
1  module Global
2      difference : [0..100] init 0;
3
4      for i from 0 to n:
5          [] (Miner_i_STATE = Add) -> 1 : (difference' = calculateFork(L_1, ..., L_n));
6  endmodule
```

Listing 6: The Global process.

The process `Global` computes the difference between the ledgers of the system every time a ledger is modified, e.g. when the `Miner_i` changes its state to `Add`, using the PRISM+ operation `calculateFork` (see Section 3). The value returned by `calculateFork` is stored in the state variable `difference`. Therefore, the probability of reaching a state of fork of length `k` within the first `T` time units is defined as:

```
P=?[F<=T difference = k]
```

The complete definition of the considered properties can be accessed in the on-line repository.

5.1 | Validation of the abstract model

To validate our model with respect to Bitcoin, we take some well-known values of the protocol from the literature and we compare them with the result of our simulations.

We begin by studying mining rates. According to the hashrate distribution of Bitcoin mining pools on May 2020, the probability that a block is mined in Bitcoin within 600 seconds is about 63% (or $1 - e^{-1}$). In 30 minutes (1800 seconds) a block has about 95% chance of being found and in 3000 seconds the probability that someone has found the block is close to 1 [C]. If we model a system with 16 miners whose hashing power (the rate) corresponds to the hashing power distribution of (the main) Bitcoin pools as illustrated in Figure 5, we obtain a probability of mining a new block which varies over time as shown in Figure 6.

Figure 6 displays the probability that some miner in the system has mined a new block.

[C] https://en.bitcoin.it/wiki/Confirmation

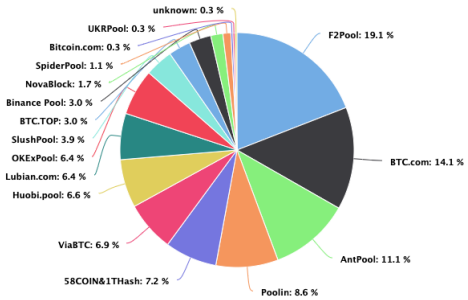


Figure 5 Hashrate distribution of Bitcoin mining pools on May 2020. Source: https://www.blockchain.com

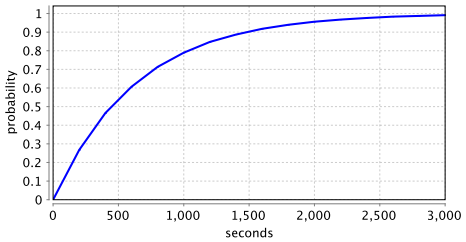


Figure 6 Probability of mining a block.

This analysis is a sanity check to verify that our model is coherent with Bitcoin. The plot shows two important facts: (i) the probability of mining a new block has an exponential behaviour as expected from the literature; (ii) the timings are in line with those of the protocol: indeed, the probability that a block is mined in 3000 seconds is almost 1. Similarly, the probabilities at 600 and 1800 seconds are coherent with the protocol. If the first fact is not surprising for how we built the model, the second one makes us confident that the results obtained by our simulations below are meaningful.

In another analysis, we study the probability of reaching a state with a fork of length 1 by varying the communication delay. The expected output is that the higher is the rate for the communications, the smaller is the time for the transitions to occur. In fact, this is

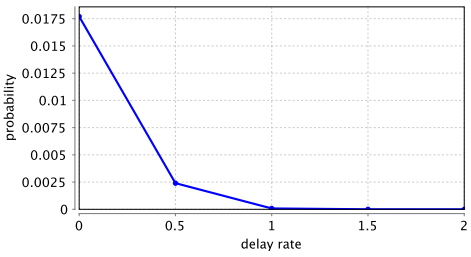


Figure 7 Probability of reaching a fork of length 1 by varying the broadcast delay; the bound time $\mathtt{T}$ is set to 600 seconds.

what Figure 7 highlights. In particular, when the communication rate is $\mathsf{r_b} = 0.08$, we obtain results in line with Bitcoin, as presented by Decker and Wattenhofer [14].

As a last analysis for validating our modelling, we study the probability of having forks of increasing length when the broadcast rate is fixed to $\mathsf{r_b} = 0.08$. Our choice derives from the observation that the average communication delay in the Bitcoin network is 12.6 seconds [14] and that can be approximated by an exponential distribution with mean λ. Thus, taking λ equal to 12.6, we have that $\mathsf{r_b} = 1/12.6 = 0.08$.

The results of our simulations are in Figure 8. The reader can observe that the probability to have a fork of length 5 is of the order of 10^{-8}, whereas it is approximately zero when the length of the fork reaches 6. This is a key result because blocks at depth 6 are considered as confirmed in Bitcoin and therefore paid.

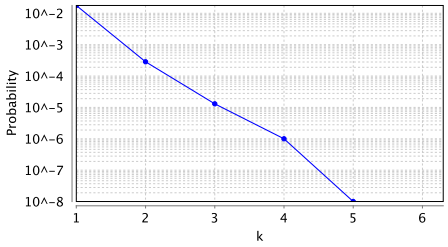


Figure 8 Probability of a fork of length `k`; the bound time `T` is set to `k*600` seconds.

5.2 | Variation of Cryptopuzzle Difficulty

We start our study of the resilience of the Bitcoin protocol to relevant changes of the rates. We begin by analyzing the probability of having a fork while varying the difficulty of the cryptopuzzle (in Bitcoin this difficulty is adjusted with respect to the computational power of the miners, in order to have a new block on average every 10 minutes). Figure 9 highlights the relationship between the probabilities of mining a block within a specific amount of time with two two different average mining rates, denoted with $1/\tau$ in the figure.

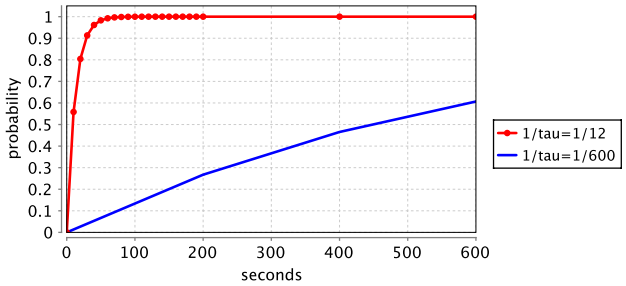


Figure 9 Probability of mining a block within 600 seconds.

The comparison is between a system with Bitcoin average mining rate (1/600) and a system where a new block is produced every 12 seconds (1/12). Of course, the probability that a miner finds a new block in the second system is much higher than Bitcoin. In particular, after 100 seconds the probability that a miner mines a new block is 1 when the average mining rate is 1/12; on the contrary, with the Bitcoin rate, the probability is less than 0.2.

Figure 10 shows the relationship between the length of the forks and the average mining rates. In this case, the probability of reaching a fork of length 6 with average mining rate 1/12 is greater than 0, whereas it becomes zero with the average mining rate of Bitcoin.

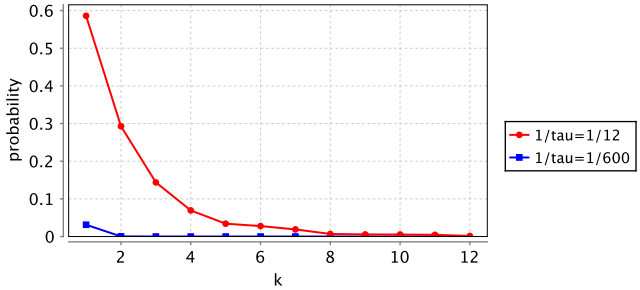


Figure 10 Probability of a fork of length k; the bound time T is set to 600 seconds.

Finally, we study how the time required to mine a block varies when we consider different cryptopuzzle difficulties. Since the difficulty of the cryptopuzzle is inversely proportional to the rate of the mining process, one might be interested in studying the trade-off between speed and security.

The results of the simulations in Figure 11 confirm that the easier the cryptopuzzle is, the faster the entire system mines a new block. This follows from the fact that the average time τ required to mine a new block is given by $\tau = \frac{2^{32}\mathsf{D}}{\mathsf{H}}$ where D is the cryptopuzzle difficulty and H is the global hash rate. Figure 12 displays how the probability of reaching a fork of length 1 varies depending on different average mining intervals. Our results show that a good balance between speed and safety can be obtained with a mining rate equal to 1/500 per

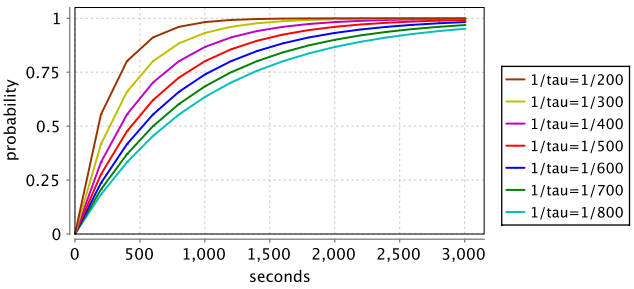


Figure 11 Probability of mining a block within 3000 seconds.

second. Indeed, with this rate, the process of mining a block is faster than in Bitcoin (1/600), but the probability of reaching a state of fork is not much higher. Even if this is a theoretical result, it shows that a better trade-off between the rate of the mining process and the

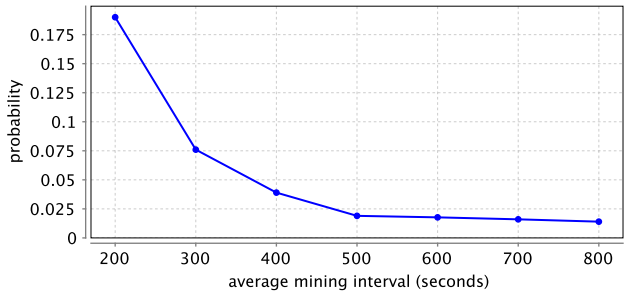


Figure 12 Probability of a fork of length 1 versus the average mining interval; the bound time `T` is set to 600 seconds.

security of the network can be obtained and can be measured. Of course, changing this trade-off may impact the behaviour of Bitcoin in different aspects. Since mining is a energy consuming task, one may expect that speeding up the process may lead to some energy savings in practice. Although this seems reasonable in theory, actually, it also depends on the behaviour and strategies of miners, e.g., they may decide to invest more in mining since it is now "easier" to mine new Bitcoins. It is not easy to predict with certainty how this change impacts miners' strategies. We leave studying this problem as a future work. A related aspect concerns understanding how the value of Bitcoins in the market varies, when the cryptopuzzle difficulty changes. Actually, a recent paper by Fantazzini and Kolodin [23] seems to suggest that the hashrate is not useful in predicting the Bitcoin price on its own. However, we believe that the change could affect the fees that miners receive for their work, so impacting their strategies. Also, studying this problem is left as a future work.

5.3 | Churn Nodes

In this subsection, we focus on the simulations of a system using the broadcast topology (see the `Network` process in Listing 2) but with three churn nodes. As anticipated in Section 4.2, our model consists of 13 "static" miners and three churn miners. The first miner goes asleep as soon as the process starts and then awakes after a while. The other two miners, at the beginning, participate at the mining process, but shut down after a given amount of time. The difference between the two is the fact that one of them, when it shuts down, does it forever, whereas the other awakes again after a while. A churn miner impacts on the Bitcoin protocol because, when a node leaves or joins the network, the overall hashing power changes [24,22,21].

This remark is confirmed by Figure 13, which compares the time needed to mine a new block with the presence of churn nodes to the time needed in the Bitcoin system with only static nodes. Since in the dynamic system, there are fewer nodes trying to solve the cryptopuzzle, the probability that some miner wins is lower. Also the probability of reaching a state of fork is lower as reported in Figure 14. This is due to the fact that there are fewer miners and, in this setting, each miner is connected to every other, thus the presence of churning nodes does not lead to message losses.

Finally, we study the latency of a churn miner when it rejoins the network because it has to download all the blocks that were mined during its absence. In particular, we assume that the mean node sleep rate ranges from 2 to 10 hours, as suggested by Motlagh at el. [21,22].

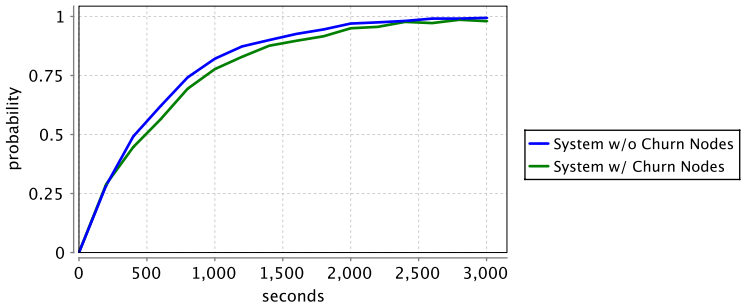


Figure 13 Probability of mining a block within 3000 seconds.

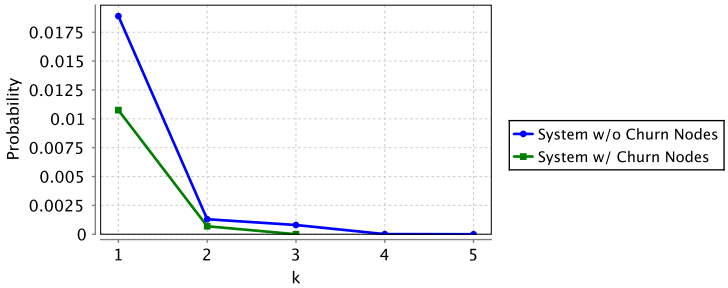


Figure 14 Probability of a fork of length k, the bound time T is set to $k * 600$ seconds.

Our experiments highlight that the synchronization process requires little time, as the reader can observe from Figure 15 which shows the probability that a node (with different sleep rates) synchronizes the missing blocks within a minute after rejoining the network.

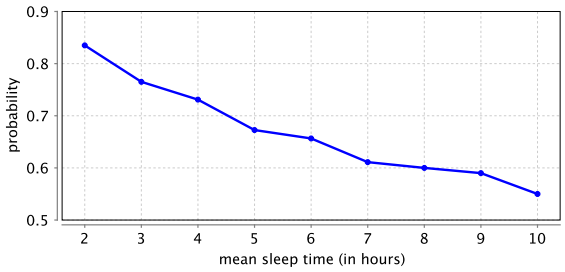


Figure 15 Probability that the node can synchronize in a minute with mean sleep time ranging from 2 to 10 hours.

Clearly, the longer the nodes are down the lower the probability that it quickly synchronizes because the number of blocks mined during its sleeping period increases.

5.4 | Different Topologies

After having described the models of different kinds of topologies in Section 4.3, we show the results of our simulations. The rates $\mathtt{r_b}$ presented in Section 4.3 are set to 1 in the simulations of linear, tree and ring topologies,

In fact, in this setting, we do not model the latency of the network by means of a rate, but we use the set of connected nodes. In particular, we choose to set $\mathtt{r_b}$ exactly to 1 so that it can be considered as an instantaneous action (the other actions have lower rates).

In Figure 16 we show that the probability of mining a block is not affected by changing the network topology. This outcome is trivial because we are changing how the nodes receive and send the new blocks. This has very high rate with respect to the mining process, which remains unchanged. It turns out that the main difference between topologies is the probability of reaching a fork of length k. As the reader would expect, when a new block is mined, if it is not forwarded to all the other nodes but only to a subset of the total, the

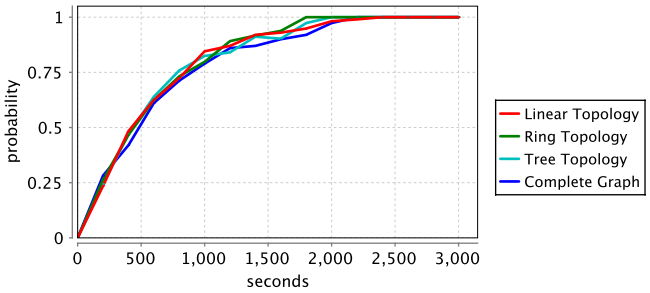


Figure 16 Probability of mining a block within 3000 seconds.

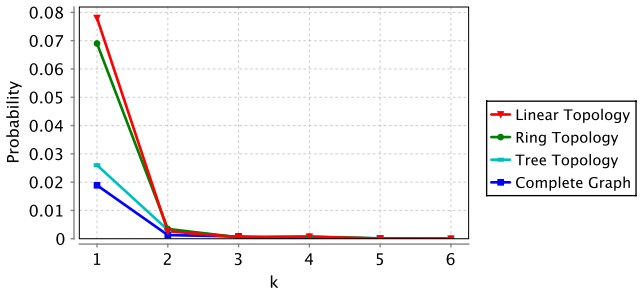


Figure 17 Probability of a fork of length k, the bound time T is set to $k * 600$ seconds.

probability of inconsistency increases. In Figure 17 we compare the four kinds of topologies: broadcast, linear, ring and tree topologies. Our result is that the smaller is the subset of the receiving nodes per miner, the higher is the probability of a fork.

6 | RELATED WORK

The blockchain protocol was introduced by Haber and Stornetta[25] and only in the last few years, because of Bitcoin, the problem of analyzing the consistency of the ledgers has attracted the interest of several researchers. The discussion of the mainstream blockchain consensus algorithms and the way the classic Byzantine consensus can be revisited for the blockchain context is presented in a paper by Gramoli[8], where the Bitcoin and Ethereum consensus algorithms are described and the behaviour of each process involved in the system is illustrated through pseudo-code. Garay et al.[26] prove the correctness of the protocol when the network communications are synchronous and focusing on persistence and liveness. The extension of this analysis to dynamic asynchronous networks with bounded delays can be found in a paper by Pass et al.[9]. There, the authors also provide an abstract model of the Bitcoin protocol that ignores all irrelevant implementation details. The abstract model enables them to formally study the behaviour of the protocol and to detect where there is room for improvement. Pîrlea and Sergey[27], instead, propose a formalization of blockchain consensus with a proof of its consistency mechanised in a proof assistant. They present an operational model that provides an executable semantics of the system and prove a form of eventual consistency focusing on the notion of global system safety. Similarly to these papers, we propose an abstract model of Bitcoin where we ignore all the implementation details which do not affect the properties of interest. The main difference between these contributions and our work is that we formalize the blockchain protocol as a stochastic system (with exponential distribution of duration) and prove its properties by simulating the model through the PRISM model checker. There are few works in the literature that have followed a research line similar to ours, i.e. studying the properties of the Bitcoin protocol using a probabilistic model checker. DiGiacomo-Castillo et al.[28] and Chaudhary et al.[29] use UPPAAL[30,31]. The former studies the security of the proof of work consensus when the network has an adversary miner that leverages the selfish mining strategy introduced by Eyal and Sirer[11]. In particular, their experiments show the effectiveness of selfish mining against various deployment parameters. Chaudhary et al.[29] analyze the probability of success of a double spending attack in the Bitcoin protocol. They show that double spending can be achieved if the parties in the Bitcoin protocol behave maliciously. In these two works, the main goal is to verify the resilience of Bitcoin by analyzing the probability of a successful attack. In contrast, we do not consider malicious miners in the network, but study the properties of the protocol under different circumstances. Another difference is that the foregoing works do not model churning nodes and only consider the case in which

a block is broadcast to all the other nodes in the system. Bastiaan[32] uses PRISM to analyze the so-called 51%-attacks (a pool can attack the network as soon as it reaches a substantial percentage of hash power) in an extension of the Bitcoin protocol (the two phase proof of work). The author proves that the extension of Bitcoin is effective at preventing the 51%-attacks. As in our work, each miner is modeled as a module in the PRISM language; however the work focuses on the actual cryptographic problem and does not implement blocks and the blockchain data structures. In particular, in our study, every cryptographic detail that does not affect our analysis is ignored.

Some recent papers propose stochastic models to analyze specific parts of blockchain systems. Biais et al.[10] focus on miners and propose a game-theoretic approach to analyze the strategies miners can adopt and the kind of equilibrium these strategies can lead to in blockchain dynamics. A similar approach is adopted by Zhang et al.[33] which propose a formal mathematical framework, to model the core concepts in blockchain-enabled economies. They illustrate the dynamics of the blockchain economies simulating and testing two different block reward strategies. The main difference with respect to our work is that their analyses focus on economic aspects. In fact, their main goal is to understand what the economic forces at the root of forks are; our analysis instead focuses on the security and the integrity aspects of the system. Moreover, Piriou and Dumas[34] propose a basic stochastic model for the blockchain protocol to capture the block creation and broadcasting process. They model blocks as abstract objects with just the necessary information to analyze the ledger evolution. They also propose a framework to ease the tuning of the model and exploit Monte Carlo simulations to obtain probabilistic results on consistency of the ledgers. In contrast with our purpose, their main goal is to check the ability to detect and prevent double-spending attacks of blockchain protocols.

7 | CONCLUSIONS AND FUTURE WORK

In this paper we analyzed the consensus protocol of Bitcoin by using an extension of the probabilistic model checker PRISM. In particular, we extended PRISM with a library implementing the notion of block of transactions and ledger natively adding the new data types `ledger`, `block` and `set`, and the operations over them. Using this extension, named PRISM+, we defined an abstract model of the Bitcoin protocol where the behaviour of the miners is described in terms of processes and the whole protocol as the parallel composition of miners. Our model is a faithful abstraction of the Bitcoin PoW protocol. Indeed, in our model each blockchain is selected as the sequence of blocks starting from the pointer to a leaf at maximal depth with root the genesis block.

If there are several leaf blocks at maximal depth, the pointer is set to the first received leaf at maximal depth.

After defining the model, we performed some probabilistic analyses covering different features of the protocol. The first analysis was instrumental to assess the coherence of our model by verifying that the probability of mining a new block within a given amount of time and that of reaching a fork correspond to those of the real Bitcoin system and coincide with the values available in the literature. The second analysis was concerned with the trade-off between security and the difficulty of the cryptopuzzle. It has been observed that a slight decrease of the difficulty level of the cryptopuzzle leads to a significant increase of the speed of mining at the cost of an almost irrelevant increase of the probability of a fork. Those results are consistent with the ones of Laneve and Veschetti[35], which formally demonstrate the probability of a fork in Bitcoin.

We also modelled and analysed networks with churn nodes, which provide a more realistic account of the behaviour of this complex platform. In particular, we pointed out that in the scenarios that we investigated, churn nodes have a strong impact on the way the mining intervals vary with time: indeed, when a node leaves the network frequently, there is an immediate consequence on how the hashing power is distributed in the network.

Finally, we simulated the Bitcoin protocol taking into account different kinds of network topologies. The driving question was checking whether the considered alternative topologies have a resistance to forks equal to or greater than the original one of Bitcoin. The results of our simulations clearly pointed out that the less the nodes are connected, the higher is the probability of reaching a state of fork. Moreover, our result made evident that dynamic participation of nodes affects the process of data propagation in the network. Namely, when a node disconnects, all of its connections are deactivated and network connectivity is reduced and this leads to an increase of the mean number of hops required for a block of transaction to propagate across the network. An interesting topic of future research could be the combination of the two different models to analyze how this could affect the likelihood of a fork.

The security of blockchains has received much attention in the last few years and various types of attacks were investigated. In this work we restricted our analysis to systems where all miners are honest and work to extend the blockchain. In future research, we plan to study these security issues by considering both peer-to-peer network based attacks and mining-based attacks. The first type of attacks, e.g. Eclipse attack[36] and Sybil attack[37], can be modeled by changing the behavior of the Network process, whereas the second one, e.g. 51% attack, can be analyzed by introducing malicious Miner processes. In this context, we could extend the analysis of Section 5.

As further future work, we plan to extend our PRISM model to faithfully simulate Ethereum. The current Ethereum's consensus algorithm is based on a PoW that is different from Bitcoin: Ethereum adopts a chain selection rule to include blocks in the main branch that the Bitcoin consensus algorithm would have excluded [38].

The analysis of other types of consensus protocols used in blockchains is also an interesting future work. One of these protocols is the Proof of Stake [39] that would require revisions of the Hasher process, as well as the introduction of a new module that record stakes.

References

1. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf; 2008.

2. Buterin V. Ethereum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper; 2013.

3. Bistarelli S, Mercanti I, Santancini P, Santini F. End-to-End Voting with Non-Permissioned and Permissioned Ledgers. J. Grid Comput. 2019; 17(1): 97–118. doi: 10.1007/s10723-019-09478-y

4. Battista GD, Donato VD, Patrignani M, Pizzonia M, Roselli V, Tamassia R. Bitconeview: visualization of flows in the bitcoin transaction graph. In: IEEE Computer Society. ; 2015: 1–8.

5. Sward A, Vecna I, Stonedahl F. Data Insertion in Bitcoin's Blockchain. Ledger 2018; 3.

6. Wong PC, Thomas J. Visual Analytics. IEEE Computer Graphics and Applications 2004; 24(5): 20-21.

7. Fischer MJ, Lynch NA, Paterson M. Impossibility of Distributed Consensus with One Faulty Process. J. ACM 1985; 32(2): 374–382. doi: 10.1145/3149.214121

8. Gramoli V. From blockchain consensus back to Byzantine consensus. Future Gener. Comput. Syst. 2020; 107: 760–769. doi: 10.1016/j.future.2017.09.023

9. Pass R, Seeman L, Shelat A. Analysis of the Blockchain Protocol in Asynchronous Networks. In: . 10211 of Lecture Notes in Computer Science. Springer. EUROCRYPT. ; 2017: 643–673

10. Biais B, Bisiere C, Bouvard M, Casamatta C. The blockchain folk theorem. The Review of Financial Studies 2019; 32(5): 1662–1715.

11. Eyal I, Sirer EG. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In: . 8437 of Lecture Notes in Computer Science. Springer. Financial Cryptography and Data Security, FC. ; 2014: 436–454

12. Zamyatin A, Stifter N, Schindler P, Weippl ER, Knottenbelt WJ. Flux: Revisiting Near Blocks for Proof-of-Work Blockchains. IACR Cryptol. ePrint Arch. 2018; 2018: 415.

13. Kwiatkowska MZ, Norman G, Parker D. Probabilistic symbolic model checking with PRISM: a hybrid approach. Int. J. Softw. Tools Technol. Transf. 2004; 6(2): 128–142. doi: 10.1007/s10009-004-0140-2

14. Decker C, Wattenhofer R. Information propagation in the Bitcoin network. In: IEEE, P2P 2013. ; 2013: 1–10

15. Antonopoulos AM, Wood G. Mastering ethereum: building smart contracts and dapps. O'reilly Media . 2018.

16. Bowden R, Keeler HP, Krzesinski AE, Taylor PG. Modeling and analysis of block arrival times in the Bitcoin blockchain. Stochastic Models 2020; 36(4): 602-637. doi: 10.1080/15326349.2020.1786404

17. Kwiatkowska MZ, Norman G, Parker D. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: . 6806 of Lecture Notes in Computer Science. Springer. Computer Aided Verification - 23rd International Conference, CAV 2011. ; 2011: 585–591

18. Aziz A, Sanwal K, Singhal V, Brayton RK. Verifying Continuous Time Markov Chains. In: . 1102 of Lecture Notes in Computer Science. Springer. Computer Aided Verification, 8th International Conference, CAV '96. ; 1996: 269–276

19. Baier C, Haverkort B, Hermanns H, Katoen JP. Model Checking Continuous-Time Markov Chains by Transient Analysis. In: Emerson EA, Sistla AP., eds. Computer Aided VerificationSpringer Berlin Heidelberg. ; 2000; Berlin, Heidelberg: 358–372.

20. Baier C, Haverkort B, Hermanns H, Katoen JP. Model-checking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering 2003; 29(6): 524-541. doi: 10.1109/TSE.2003.1205180

21. Motlagh SG, Misic JV, Misic VB. Modeling of Churn Process in Bitcoin Network. In: IEEE. ; 2020: 686–691

22. Motlagh SG, Misic JV, Misic VB. Impact of Node Churn in the Bitcoin Network. IEEE Trans. Netw. Sci. Eng. 2020; 7(3): 2104–2113. doi: 10.1109/TNSE.2020.2974739

23. Fantazzini D, Kolodin N. Does the Hashrate Affect the Bitcoin Price?. Journal of Risk and Financial Management 2020; 13(11). doi: 10.3390/jrfm13110263

24. Motlagh SG, Misic JV, Misic VB. An analytical model for churn process in Bitcoin network with ordinary and relay nodes. Peer-to-Peer Networking and Applications 2020; 13(6): 1931–1942. doi: 10.1007/s12083-020-00953-y

25. Haber S, Stornetta WS. How to Time-Stamp a Digital Document. In: . 537 of Lecture Notes in Computer Science. Springer. CRYPTO. ; 1990: 437–455

26. Garay JA, Kiayias A, Leonardos N. The Bitcoin Backbone Protocol: Analysis and Applications. In: . 9057 of Lecture Notes in Computer Science. Springer. Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques. ; 2015: 281–310

27. Pîrlea G, Sergey I. Mechanising blockchain consensus. In: CPP 2018, Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, Los Angeles, CA, USA, January 8-9, 2018. ; 2018: 78–90

28. DiGiacomo-Castillo M, Liang Y, Pal A, Mitchell JC. Model Checking Bitcoin and other Proof-of-Work Consensus Protocols. In: 2020 IEEE International Conference on Blockchain (Blockchain). ; 2020: 351-358

29. Chaudhary K, Fehnker A, Pol v. dJ, Stoelinga M. Modeling and Verification of the Bitcoin Protocol. In: . 196 of EPTCS. MARS. ; 2015: 46–60.

30. Bengtsson J, Larsen K, Larsson F, Pettersson P, Yi W. UPPAAL—a Tool Suite for Automatic Verification of Real-Time Systems. In: Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control: Verification and Control. Springer-Verlag; 1996; Berlin, Heidelberg: 232–243.

31. David A, Larsen KG, Legay A, Mikuăionis M, Poulsen DB. Uppaal SMC Tutorial. Int. J. Softw. Tools Technol. Transf. 2015; 17(4): 397–415.

32. Bastiaan M. Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin. In: ; 2015.

33. Zhang Z, Zargham M, Preciado VM. On modeling blockchain-enabled economic networks as stochastic dynamical systems. Appl. Netw. Sci. 2020; 5(1): 19. doi: 10.1007/s41109-020-0254-9

34. Piriou P, Dumas J. Simulation of Stochastic Blockchain Models. In: IEEE Computer Society, EDCC 2018. ; 2018: 150–157

35. Laneve C, Veschetti A. A Formal Analysis of the Bitcoin Protocol. In: . 86 of OASIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. Recent Developments in the Design and Implementation of Programming Languages; 2020: 2:1–2:17.

36. Heilman E, Kendler A, Zohar A, Goldberg S. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In: 24th USENIX Security Symposium, Washington, D.C., USA, August 12-14, 2015. ; 2015: 129–144.

37. Douceur JR. The Sybil Attack. In: . 2429 of Lecture Notes in Computer Science. Springer, IPTPS 2002. Peer-to-Peer Systems, First International Workshop, IPTPS 2002. ; 2002: 251–260

38. Wood G. Ethereum: a secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf; 2014.

39. Bentov I, Gabizon A, Mizrahi A. Cryptocurrencies Without Proof of Work. In: . 9604 of Lecture Notes in Computer Science. Springer. Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC. ; 2016: 142–157