Stochastic Modelling and Analysis of Bitcoin

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

[1] University of Perugia
[2] IMT School for Advanced Studies, Lucca
[3] University of Bologna – INRIA Focus

Abstract. Nakamoto's Bitcoin blockchain protocol implements a distributed ledger on peer-to-peer asynchronous networks. In this paper we analyze Bitcoin by (*i*) extending PRISM, a probabilistic model checker, with a *ledger datatype*, (*ii*) modelling the behaviour of blockchain's key participants – the *miners* – and (*iii*) describing the whole protocol as a parallel composition of processes. The probabilistic analysis of the model highlights how forks happen and how they depend on some parameters of the protocol, such as the difficulty of the crypto-puzzle and the network communication delays. Our results confirms that considering transactions in blocks at depth larger than 5 as permanent is reasonable because the majority of miners have consistent blockchains up-to that depth with probability almost 1.

1 Introduction

Blockchain is an emerging technology that implements a distributed ledger on peer-to-peer asynchronous networks that are dynamic, in the sense that nodes may either join or leave. It is exploited in many contexts including the management of cryptocurrencies (Bitcoin being the most famous one [21]) and of decentralized applications (e.g., Ethereum smart contracts [7]), the implementation of voting systems [5] and the support of other application specific protocols [2, 25, 26].

When implementing a distributed ledger on a dynamic peer-to-peer asynchronous network, one has to address the problem of inconsistent updates of the ledger performed by different nodes. This problem is actually a variant of the distributed consensus, which is known to be unsolvable since 1985 [13]. To overcome this shortcoming, blockchain uses an ingenious approach: it guarantees a so-called *eventual consistency* whereby the various replicas of the ledger may be temporarily inconsistent in at most the last m blocks.

The blockchain consensus protocol is very complex and the current research is actively involved in studying its properties and its criticalities. Indeed, understanding the details of the behaviour of such protocol is of paramount importance because overseeing some details might introduce vulnerabilities and pave the way to attacks. For example, inconsistencies of the ledger replicas, which are 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 make the blockchain evolve to a wrong state.

In this paper we consider the consensus protocol used in Bitcoin and study its properties by taking advantage of formal methods. In particular, we focus on the key participants of the protocol, the *miners* and model their behaviour as PRISM [19] processes; then we describe the whole protocol as the parallel composition of the involved miners. Since the duration of mining a block and of broadcasting a message to all the nodes of the network can be modelled as an *exponential distribution* with a so-called *rate parameter* [4, 12, 28], we found it appropriate to rely on Continuous Time Markov Chains (CTMCs) for providing a probabilistic model of our processes. Remarkably, to the best of our knowledge this is one of the first paper formally studying the properties of Bitcoin through a probabilistic model checker.

In order to model and analyse Bitcoin, we use PRISM [19], an off-the-shelf tool suitable for modelling and analysing systems that exhibit random or probabilistic behaviours. However, since PRISM lacks linguistic primitives to model the complex datatypes used by the protocol underlying Bitcoin, we had to extend it with a library implementing the notion of block of transactions and of ledger in a native way: `ledger`, `block` and `set`. After this extension, we have been able to define a simple model of the Bitcoin protocol that details miners' behaviours as processes and analyse the resulting system. As an initial step, we assess the coherence of our model by verifying that the probability of mining a new block within a given amount of time and the probability of forking we compute are in full agreement with the values available from the literature.

In particular, it turns out that the probability of a fork strictly depends on the broadcast delay and on the difficulty of the cryptopuzzle, that is the difficulty of computational problem that miners must resolve in order to add a new block to the ledger. Our analysis also vindicates the statement that waiting for at least 6 confirmations before considering a transaction as permanent in the ledger is reasonable because at that time the majority of miners has a consistent blockchain with probability almost 1.

After the empirical validation of the model, we study the trade-off between security guarantees and hardness of the solution of the cryptopuzzle needed to add a new block to the ledger. More precisely, we study the probability of reaching forking states when we decrease the level of difficulty of the cryptopuzzle in order to increase the speed of the mining process. Our analysis shows that by slightly decreasing the difficulty level the speed of mining increases at the cost of an almost irrelevant increase of the probability of forking.

We would like to stress that our PRISM libraries (`ledger`, `block` and `set`) are generic, in the sense that they are not tied to the Bitcoin protocol. In fact, they can be used for modelling and analysing other blockchain protocols, which is one of the research topics in our future agenda.

The rest of the paper is structured as follows. Section 2 contains a short account of Bitcoin and its (*Proof of Work*) consensus algorithm and recap the main constructs of the PRISM language. Section 3 presents our extension of the

PRISM language with the new data types, `ledger`, `block` and `set`. Our model of Bitcoin is presented in Section 4. Section 5 contains our simulations of the Bitcoin protocol and the comparison with results obtained when the difficulty parameter is changed. Section 6 compares our proposal with the literature and Section 7 draws some conclusions and discusses possible future work.

2 Background

2.1 Bitcoin and Proof of work (PoW)

Bitcoin is a peer-to-peer asynchronous global network of nodes that hosts a ledger logging economic transactions. Transactions are created by participants to record cryptocurrency transfers and are broadcasted to all the nodes of the network. Suitable nodes of the network – the *miners* – gather these transactions, collect them in blocks and update the ledger. The integrity and the consistency of the ledgers are ensured by a mix of cryptographic primitives, i.e. secure hash functions and digital signatures, and by the consensus protocol. This protocol is run by every miner of the network and ensures that the updates performed on the ledger by the different miners will be eventually consistent. In particular, the protocol is byzantine fault tolerant [9, 20] and guarantees that the (eventual) consensus is always reached found by the majority of the nodes.

A key parameter of the Bitcoin consensus protocol is the mining speed. Clearly, the faster miners are, the higher is the probability of conflicts and thus of inconsistencies likely is the inconsistency between miners. Bitcoin regulates mining by relying on the *Proof of work* (PoW) [21] technique, which requires miners to solve a computational hard problem before being allowed to add a new block to the ledger[1]. By regulating the difficulty of the PoW, Bitcoin keeps the mining speed at around 10 minutes. Once a miner solves the PoW, it broadcasts the block to all the other nodes of the network to be added to their own ledger, and starts working on the next block.

Since the Bitcoin network is asynchronous it may happen that two nodes mine and broadcast a block concurrently. This phenomenon, called *fork*, is at the core of the inconsistency between ledgers of the network. To overcome this problem, the Bitcoin protocol guarantees the so-called *eventual consistency* that we shall shortly explain. Every miner has *a view* of the ledger, which is the longest chain starting from the root (the genesis block) – this chain is called *blockchain* – (there may be several such chains, a miner just commits to one). When a miner creates a new block, the block is added to its ledger in the leaf position of his blockchain and sends the block to the other nodes with the pointer to his parent. When a miner receives new blocks, it connects them to his ledger and updates its

[1] Technically, the problem consists of finding a number, dubbed *nonce*, that, once concatenated and hashed together with the (header of the) block, produces a target that starts with a certain number of zeros. The only solution to find such a nonce is through an exhaustive search and the difficulty is proportional to the size of the nonce search space and to the number of zeros required to the target.

"view" if another chain has become longer than its own blockchain, generating *stale* blocks[2]. Notice that due to network delays it might happen that a miner receives a block that cannot be connected to its ledger because a previous block is missing (they are called *orphan*). Eventual consistency means that all the blockchains of the network are consistent up-to the last few blocks. For this reason, Bitcoin considers both transactions and miner's rewards in blocks at depth greater than 6 as permanent [1].

In this paper we will overlook a number of features of Bitcoin that are either not relevant for the analysis of the protocol or that can be abstracted out. Such as the Bitcoin rewards that are reserved for those who solve the problem[3] or the computational complexity of the cryptopuzzle (that can be abstracted by choosing ad-hoc rates of the mining actions).

2.2 PRISM language

PRISM [19] is a probabilistic model checker that inputs a formal probabilistic description of a system and computes the likelihood of the occurrence of certain events. PRISM supports different kinds of probabilistic formalisms; in this contribution we focus on CTMCs models, which are transition system where each transition from a state s to a state s' is labeled by a positive real number ρ, dubbed the *rate* of the transition. Then, the probability of performing a transition from state s to state s' within t time units is given by the exponential distribution $1 - e^{-\rho t}$ [18]. We refer to [17] for a full account of the formalisms supported by PRISM.

In order to specify systems, PRISM uses a concurrent language, the *PRISM language*. A system is the parallel composition of a set of modules, representing processes, that can interact with each other; every module represents a (sequential) agent and has an internal state represented by a tuple of local variables. The overall state of a system is determined by the state of all the modules.

The behaviour of a module is defined by a set of commands that specify how and under which conditions a module performs a transition and updates its internal state. Modules have commands of the form

```
[] guard -> rho_1:update_1+...+rho_n:update_n;
```

where `guard` is a predicate over the variables in the module (even those of other modules); and `update_i` corresponds to a transition that the system can make. A transition is defined as an update that assign a new value to some variables of the module. When `guard` is true, the module chooses a transition specified in the command (the operator + denotes a nondeterministic choice) according to the rate `rho_i` associated to that update. That is, the next state is computed according to a race condition [18]. In fact, typically from one state there is more

[2] Blocks which were successfully mined but which are not included on the current best blockchain, likely because another block at the same height had its chain extended first.

[3] These are 6.25 Bitcoin in May 26th 2020.

than one reachable state and the first transition to be triggered determines the next state of the CTMC.

Commands can also specify synchronizations between modules through actions, which are placed inside square brackets. To clarify, we propose a simple example from PRISM documentation[4] that models an N-place queue of jobs and a server which removes jobs from the queue and processes them:

```
1  ctmc
2
3  const int N = 10;
4  const double mu = 1/10;
5  const double lambda = 1/2;
6  const double gamma = 1/3;
7
8  module queue
9       q : [0..N];
10
11      [] q<N -> mu:(q'=q+1);
12      [] q=N -> mu:(q'=q);
13      [serve] q>0 -> lambda:(q'=q-1);
14 endmodule
15
16 module server
17      s : [0..1];
18
19      [serve] s=0 -> 1:(s'=1);
20      [] s=1 -> gamma:(s'=0);
21 endmodule
```

In the above code we define two modules, `queue` (row 8 to 14) and `server` (row 16 to 21) that synchronize on the action `serve` (in square parenthesis). Module `queue` has an integer variable `q` (defined in row 9) representing its size (the constant `N` - defined in row 3 - denotes its capacity). Transitions describe the operations on the queue. The first one (row 11) inserts a new element with rate `mu`, if the queue is not full (`q<N`); the insertion is rendered by incrementing the value of `q`. Note that the PRISM language uses the prime notation to denote the new value of a variable, in our case `q' = q + 1`. The second transition (row 12) says that no new element is inserted when the queue is full. The last transition (row 13) removes an element; it is triggered when the queue is not empty and there is a synchronization with module `server` on the variable `serve`.

In the module `server`, the boolean variable `s` (row 17) defines whether the server is busy or not. The first transition (row 19) allows the server to synchronize with `queue` on variable `serve` when the server is idle. After this synchronization the server updates its state to busy (`s'=1`). Note that the rate of this transition is equal to the product of the two individual rates (in this case, `lambda`*1). The second transition (row 20) of module `server` states that a busy server (`s=1`) may complete its task with rate `gamma`.

[4] `http://www.prismmodelchecker.org/manual/ThePRISMLanguage/Example2`

3 The Extended PRISM Language

To have a faithful implementation of the Bitcoin protocol we have extended the PRISM model checker with three dynamic data types: `block`, `ledger` and `list`. In this section we overview our extension.

3.1 The Block

As discussed in Section 2, Bitcoin blocks record the transactions that are going to be certified and also contain additional information, such as its hash value and a pointer to its parent. In our model, a block consists of two pieces of information: a *name* that uniquely identify the block and the name of the previous block (call it *father*) to which it is connected. Therefore the pair {`name;father`} uniquely identifies a block. In turn, `name` is a pair whose first element is the identity of the miner which mined the block and the second element is a unique numeric label. For instance, `m4,7` represents the name of a block created by the miner `m4` with label `7`. The numeric label counts the number of blocks minted by the miner. The `father` is encoded by the name of the corresponding block. Henceforth, {`m3,0; m4,7`} denotes a block named `m3,0` whose previous block is `m4,7`. We also consider all blocks of this model as valid [5]

3.2 The ledger

The `ledger` data type is a pair {`T;p`}, where `T` is a tree of blocks with each block pointing to its own parent, and `p` is a pointer to a leaf at maximal depth (therefore the name of the corresponding block). The root of the tree is the *genesis block* that is denoted by {`genesis,0; genesis,0`}. The *blockchain* of {`T;p`} is the sequence of blocks starting from `p` (therefore the last block is the genesis one). When a miner with a ledger {`T;p`} receives a block, he connects the block at his own ledger and, in case the corresponding depth is higher than the one pointed by `p`, his updates `p`. (This is not always possible because the father of the block maybe not in `T`.) When a (honest) miner mines a new block, the corresponding father is set to `p` and `p` is updated. In order to define these operations we use an operator `+` that has different implementations depending on its argument types. In particular:

- `ledger+block`: the first term is a ledger {`T;p`} and the second is a block `b`. The operator adds `b` to `T` if the father of the block is already in the chain. If the block is not present in the chain, the operator does nothing. In case the chain of `b` becomes longer than `p`, `p` is updated to the name of `b`.
- `ledger+integer`: the first term is a ledger {`T;p`} and the second is an integer `n`. In this case, the operator `+` represents the creation of a new block. In particular, if the miner is honest, it creates a block whose father is `p` and whose name is the miner's name plus `n+1` (the new block points to `p`, therefore it is at longer depth). The pointer `p` is updated to the name of the new block and the number of nodes created by the miner is set to `n+1`.

[5] For a block to be valid it must hash to a value less than the current target one.

– `ledger+set`: the first term is a ledger {`T;p`} and the second one a set of blocks. The operator extracts one element from the set and adds it to the `T`, if it is possible. If the block is added then it is removed from the set.

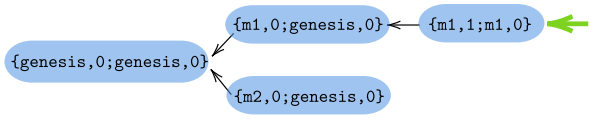

Figure 1. An example of blockchain.

In Figure 1, we represent a ledger with two blocks created by `m1` and one block created by `m2`. The corresponding blockchain is the one pointed by the rightmost arrow. Instead the block created by `m2` is a valid stale block.

3.3 The Set

The `set` data type is implemented as a list of blocks without duplications. The data type has an extraction operation, represented by the operator `+`, that returns an element of the set (which is extracted randomly) and removes it from the set. The list is used to maintain, for each miner, the orphan blocks.

4 The Bitcoin Model

In our model, a Bitcoin system is a parallel composition of n miners and two processes (*Hasher* and *Network*) that simulate, respectively, the miner hashing process and the broadcast of new blocks (see Figure 2). Every process is expressed

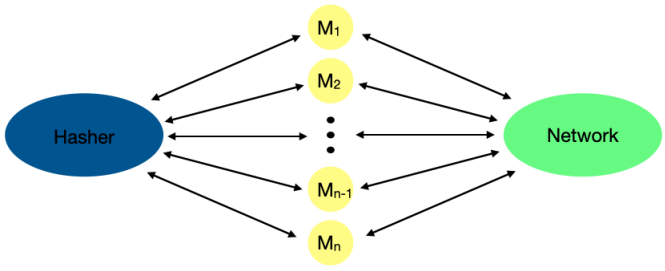

Figure 2. The Bitcoin model architecture.

as a CTMC entity because

1. the time spent by a miner m_i to mine a block can be described by an exponential distribution $1 - e^{-\lambda_{m_i}}$, where the parameter λ_{m_i} depends on the miner hashing power and the difficulty level of the crytopuzzle Nakamoto [21],
2. the communication delay across the Bitcoin network can be also approximated by an exponential distribution [10].

For the sake of clarity, we are presenting a simplified version of the implemented code[6].

```
1  module Hasher
2          [win] (Hashers_STATE=HasherRes) -> mR : τHasher;
3          [lose] (Hasher_STATE=HasherRes) -> lR : τHasher;
4  endmodule
5
6  module Miner
7          Miner_i_STATE : [Mine,Winner,Lost,Add] init Mine;
8          b : block {m,0;genesis,0};
9          B : ledger [{genesis,0;genesis,0}];
10         c : [0..N] init 0;
11         setMiner_i : set [];
12
13         [win] (Miner_i_STATE=Mine) -> hR_i : (Miner_i_STATE'=Winner);
14         [lose] (Miner_i_STATE=Mine) -> hR_i : (Miner_i_STATE'=Lost);
15         [addBlock] (Miner_i_STATE=Winner) ->
16                     1 : (c'=c+1)&(b'=B+c)&(B'=B+b)&(Miner_i_STATE'=Mine);
17         [addBlock] (Miner_i_STATE=Lost) ->
18                     rMw_i : (setMiner_i'=setMiner_i+set)&(Miner_i_STATE'=Add);
19         [] (Miner_i_STATE=Add) -> 1 : (B'=B+setMiner_i)&(Miner_i_STATE'=Mine);
20 endmodule
```

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

The Hasher process is described in lines 1 to 4 of Listing 1. We use it to abstractly represent the PoW made by miners. In particular, miners who want to solve the crypto-puzzle 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]` and rate `mR` is triggered when the synchronizing miner finds a solution for the PoW ; the second one with action `[lose]` and rate `lR` (`lR` $= 1 -$ `mR`) is triggered when the synchronizing miner does not find a solution to PoW.

A Miner, described in Listing 1 line 6 to 20, behaves as follows:

- it may receive a block from the network.
- it may create (e.g. mine) a new block: in our setting, mining a block amounts to winning the proof of work contest and it is represented by a rate. This rate indicates the nodes' rate of generating new blocks. Therefore, it corresponds to the computational power of miners to solve the cryptopuzzles of the proof-of-work. In this way we abstract away from the proof-of-work technique for mining blocks. When a block is created by a miner, it is added to the local ledger and it is forwarded to all the other miners of the network.

[6] The actual implementation can be found here `https://github.com/adeleveschetti/bitcoin-analysis/`

– it may try to add blocks stored in his local set to the local ledger.

A Miner has four state variables: `B` represents its own ledger corresponding tp the local view of the state of the system; `b` indicates the last valid block added to `B`; `setMiner` indocates the miner's local set in which the block received by the network are stored before being (eventually) added to the local blockchain `B`, and `c` is a counter of the minted blocks.

The Network process is defined in Listing 2.

```
1  module Network
2          Network_STATE : 0;
3          n : numberOfMiners
4      for e from 0 to n:
5              set_e : set [];
6
7      for i from 0 to n:
8          [addBlock] (Miner_i_STATE=Winner)  ->
9          1 :
10         for e from 0 to n:
11            if e != i:
12              set_e'=set_e+b;
13 endmodule
```

Listing 2. Simplified model of the Network.

It has one set per miner that stores the messages (the blocks) to be delivered to the corresponding miner.

As the reader can observe from code in Listing 1, Miner and Hasher synchronize over the `win` and `lose` actions. `Miner` receives from `Hasher` the answer for its work, which can be either positive or negative depending on the difficulty of the problem (represented by the hasher values `mR` and `lR`) and 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, and the rate of loosing the competition is `lR`×`hR_i`

If the miner wins, it changes its status in *Winner* and creates a new block, which is used to update its ledger – action `addBlock` – and to send it to the Network in order to communicate the new block to the other miners (e.g. updating other miners' sets with the new block). If the miner looses, its status becomes *Lost* and it checks for new blocks in the Network with a certain rate `rMw_i`, which simulates the latency of the network. If there are new blocks, it adds one of them, chosen randomly to simulate the delay of the network, to its local set. Then, the state of the Miner becomes *Add* and it tries to add the blocks to its ledger. Finally, its status returns to *Mine* and the process starts again.

5 Simulations

In this section we discuss the outcome of two probabilistic analyses resulting from the use of PRISM to model the behavior of Bitcoin. The first analysis validates our model with respect to Bitcoin; the second one studies the trade-off between the security and the efficiency/scalability of the network (e.g. we study the relationship between the probability of reaching a fork and the difficulty of

the cryptopuzzle). 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.

5.1 Bitcoin coherence

We analyse a system with 16 miners representing the main pools in the Bitcoin network. To each miner we assign a hashing power (a rate) taking the current hashing power distribution of Bitcoin illustrated in Figure 3. Figure 4 shows

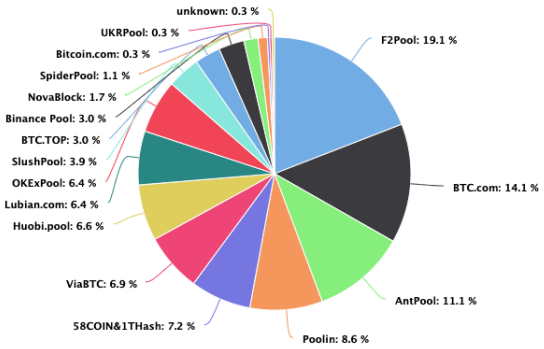

Figure 3. Hashrate distribution of Bitcoin mining pools on May 2020.
Source: https://www.blockchain.com/.

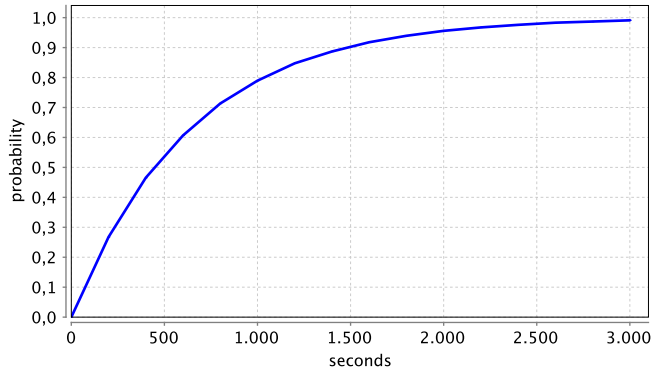

Figure 4. Probability of mining a block within 3000 seconds.

the probability that a block will be mined in 3000 seconds. As the reader can observe, the probability of a block being found 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. We would like to stress that our model follows what is observed in the real execution of the protocol.[7] It is, indeed, well known that the Bitcoin protocol automatically adjust the difficulty of the cryptopuzzle to maintain constant, around 10 minutes, the average time between the creation of two blocks [21].

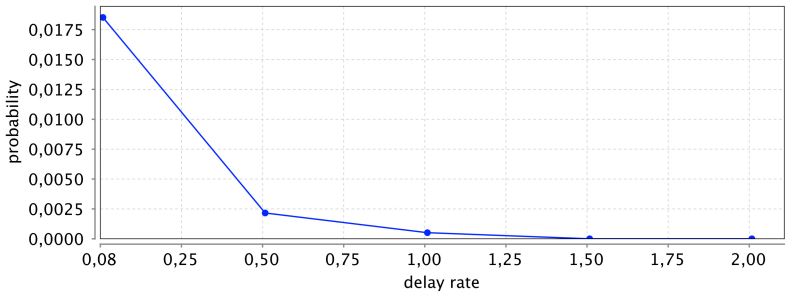

Figure 5. Probability of reaching a fork of length 1 by varying the broadcast delay.

In Figure 5, we analyze the probability of reaching a state where at least two different blockchains differ for one block (fork of length 1). The reader may notice that the probability decreases with the increase of the communication delay rate. This follows form the remark that the higher is the rate, the smaller is the expected time for the transition to occur. We notice that, with rate $r_{broadcast} = 0.08$, we obtain results in line with those of [10].

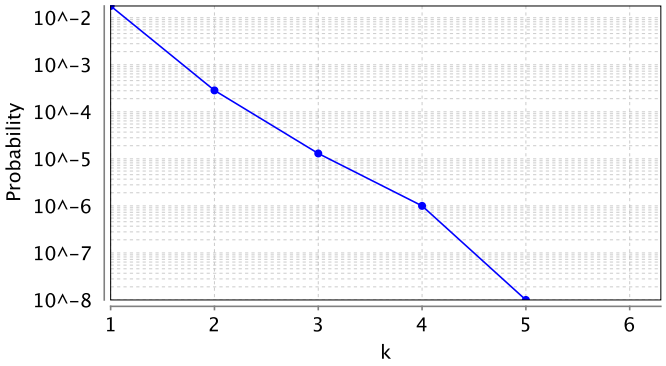

Figure 6. Probability of a fork of increasing length.

[7] `https://en.bitcoin.it/wiki/Confirmation`

Figure 6 reports the probability of having forks of increasing length. In this case, we fix the broadcast delay to $r_{broadcast} = 0.08$. As the reader can observe, the probability to obtain a fork of length 5 is of the order of 10^{-8}, while it is approximately zero when the length of the fork reaches 6. This is a key result, because a fork of length 6 is considered to be the longest Bitcoin fork.

5.2 Variation of Cryptopuzzle Difficulty

In this section, we compare the probability of mining new blocks and having a fork while varying the difficulty of the cryptopuzzle.

Figure 7 highlights the relationship between the probabilities of mining a block in a specific amount of time with two different difficulty rates D. In par-

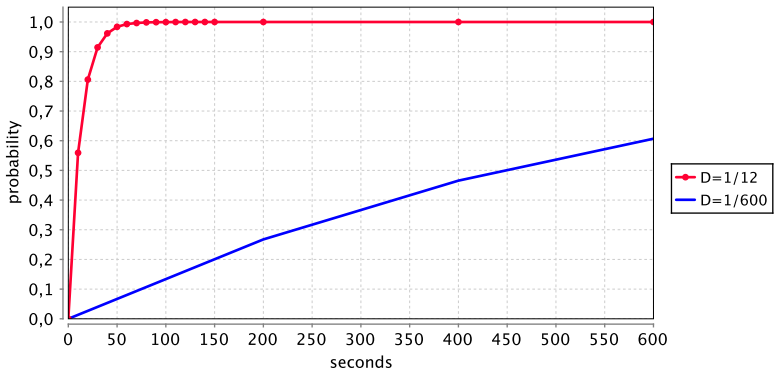

Figure 7. Probability of mining a block within 600 seconds.

ticular, the comparison is between a system with Bitcoin difficulty 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 this second system is way higher than Bitcoin. In particular, after 100 seconds the probability that a miner mines a new block is 1 when the difficulty rate is 1/12. On the contrary, with the Bitcoin rate, the probability is less than 0.2.

Figure 8 shows how the probability of reaching longer forks varies when the difficulty rate changes. We have considered the same difficulty rates of Figure 7. In this case, the probability of reaching a fork of length 6 with difficulty rate 1/12 is greater than 0, whereas it becomes zero with the difficulty rate of Bitcoin.

Finally, we study how the time required to mine a block varies when we consider different cryptopuzzle difficulties. The results in Figure 9 confirm that the easier the cryptopuzzle is, the faster the entire system mines a block.

Since the difficulty of the cryptopuzzle affects the speed, one might be interested in studying the trade off between speed and security. Figure 10 displays

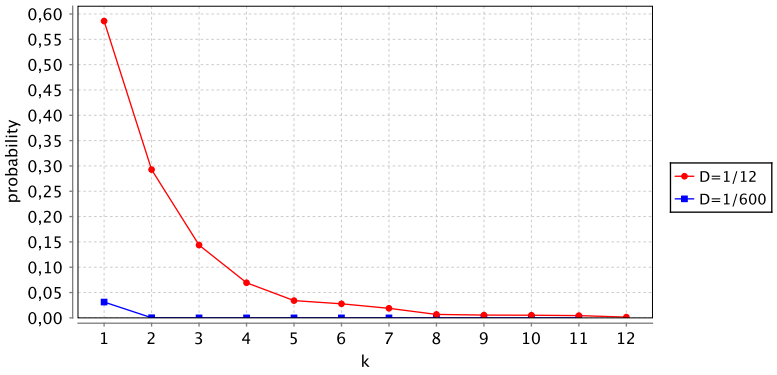

Figure 8. Probability of forks.

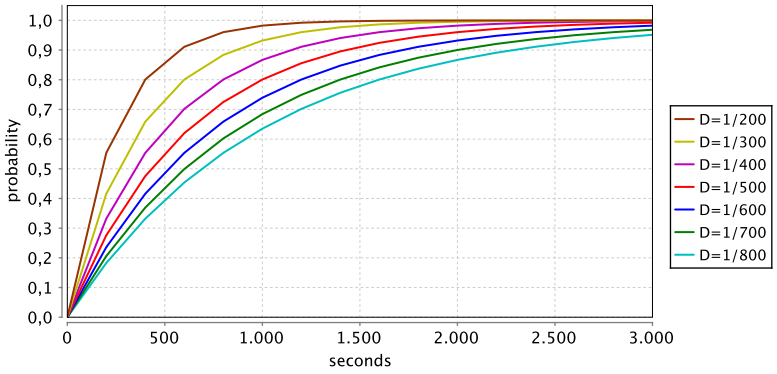

Figure 9. Probability of mining a block within 3000 seconds.

how the probability of reaching a fork of length 1 varies depending on the difficulty parameter. Our results show that a good balance between speed and safety can be obtained with a difficulty rate equals to 1/500. 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 speed of the mining process and the security of the network can be obtained. Since mining in Bitcoin is time and energy consuming, adjusting this trade off may lead to some improvements in the practice.

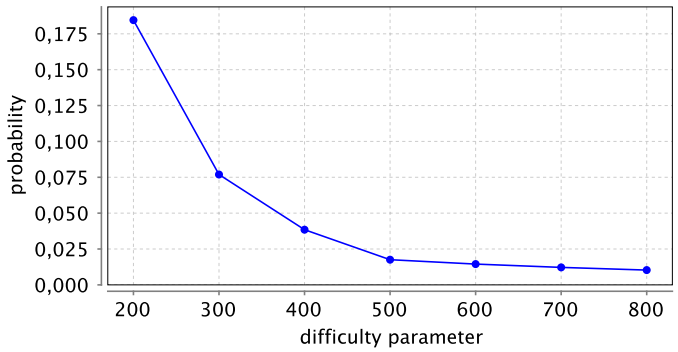

Figure 10. Probability of a fork of length 1 with different difficulty parameter.

6 Related Works

The blockchain protocol was introduced by Haber and Stornetta [15] and only in the last few years, because of Bitcoin, the problem of analyzing the consistency of the ledgers has caught the interest of several researchers.

In [14], Garay *et al.* demonstrate the correctness of the protocol when the network communications are synchronous, focusing on its two key security properties: *Common Prefix* and *Chain Quality*. The first property guarantees the existence of a common prefix of blocks among the chain of honest players. The second property constrains the number of blocks mined by hostile players, when the honest players are in the majority and follow the protocol.

The extension of this analysis to asynchronous networks with bounded delays of communications and with new nodes joining the network has been undertaken in [22]. In the above contributions, the properties are verified by using oracles that drive the behaviours of actors. Then, combining the probabilistic behaviours and assuming possible distributions, one computes expected values. In contrast with the above works, in [24], Pîrlea and Sergey propose a formalization of blockchain consensus focusing on the notion of global system safety. They present an operational model that provides an executable semantics of the system where nondeterminism is managed by external schedules and demonstrate the correctness by means of a proof assistant.

The main difference between these contributions and our work is that we formalize the blockchain protocol as a stochastic system (with exponential distribution of durations) and derive the properties by simulating the model through the PRISM model checker. As regards stochastic models for blockchain, few recent researches use them to select optimal strategies for maximizing profit of a player [4, 29], for formalizing interactions between miners as a game [6, 8] and to check the ability to discard double-spending attacks of blockchain protocols [23].

7 Conclusions and Future Works

In this paper we analyzed the consensus protocol of Bitcoin by resorting to 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: `ledger`, `block` and `set`. Using this extension, we defined a simple model of the Bitcoin protocol where miners' behaviours are described as processes and the whole protocol as a parallel composition of miners.

Then we performed two probabilistic analyses covering different features of the protocol. The first one assessed 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 that of the real Bitcoin system and to the values available in the literature. The second analysis has been concerned with the trade-off between the security and the difficulty of the cryptopuzzle. It pointed out that slightly decreasing the difficulty level of the cryptopuzzle increases the speed of mining at the cost of an almost irrelevant increase of the probability of a fork. To the best of our knowledge, we are the first to use a model checker to study properties of Bitcoin.

The security of blockchains has received much attentions in the last few years and various types of attacks have been found. 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 [16] and Sybil attack [11], can be modeled by changing the behavior of the Network process, whereas the second one, e.g. 51% attack, can be analyzed introducing malicious Miner processes.

As further future work, we plan to extend our PRIM model to faithfully simulate Ethereum. The current Ethereum's consensus algorithm is based on proof-of-work that is different from Bitcoin: Ethereum adopts a chain selection rule to harness the residual mining power in pruned blocks to improve security. The protocol includes such blocks in the blockchain, and rewards the miners [27] who mint them. The analysis of other types of consensus protocols used in blockchains, for example proof of stake [3], requires modifying the Hasher process, and it is also an interesting future work. Another topic worth further investigation would be the generalization of the Network process in order to model different network topologies.

References

[1] A.M. Antonopoulos and G. Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media, Incorporated, 2018. ISBN: 978-1-4919-7194-9. URL: `https://books.google.it/books?id=SedSMQAACAAJ`.

[2] G. D. Battista et al. "Bitconeview: visualization of flows in the bitcoin transaction graph". In: *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*. Oct. 2015, pp. 1–8.

[3] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. "Cryptocurrencies without proof of work". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 142–157.

[4] Bruno Biais et al. "The blockchain folk theorem". In: *The Review of Financial Studies* 32.5 (2019), pp. 1662–1715.

[5] Stefano Bistarelli et al. "End-to-End Voting with Non-Permissioned and Permissioned Ledgers". In: *J. Grid Comput.* 17.1 (2019), pp. 97–118. DOI: 10.1007/s10723-019-09478-y. URL: https://doi.org/10.1007/s10723-019-09478-y.

[6] Joseph Bonneau et al. "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies". In: *Proc. of SP 2015*. IEEE Computer Society, 2015, pp. 104–121.

[7] Vitalik Buterin et al. "Ethereum white paper". In: *GitHub repository* (2013), pp. 22–23.

[8] Miles Carlsten et al. "On the Instability of Bitcoin Without the Block Reward". In: *Proc. Computer and Communications Security*. CCS '16. ACM, 2016, pp. 154–167.

[9] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*. Ed. by Margo I. Seltzer and Paul J. Leach. USENIX Association, 1999, pp. 173–186. URL: https://dl.acm.org/citation.cfm?id=296824.

[10] Christian Decker and Roger Wattenhofer. "Information propagation in the bitcoin network". In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

[11] John R Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[12] Ittay Eyal and Emin Gün Sirer. "Majority is Not Enough: Bitcoin Mining is Vulnerable". In: *Commun. ACM* 61.7 (June 2018), pp. 95–102. ISSN: 0001-0782. DOI: 10.1145/3212998. URL: http://doi.acm.org/10.1145/3212998.

[13] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (1985), pp. 374–382.

[14] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Proc. of EUROCRYPT 2015*. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 281–310.

[15] Stuart Haber and W. Scott Stornetta. "How to time-stamp a digital document". In: *Journal of Cryptology* 3.2 (Jan. 1991), pp. 99–111. ISSN: 1432-1378. DOI: 10.1007/BF00196791. URL: https://doi.org/10.1007/BF00196791.

[16] Ethan Heilman et al. "Eclipse attacks on bitcoin's peer-to-peer network". In: *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 2015, pp. 129–144.

[17] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Springer, 2011, pp. 585–591.

[18] M. Kwiatkowska, G. Norman, and D. Parker. "Stochastic Model Checking". In: *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*. Ed. by M. Bernardo and J. Hillston. Vol. 4486. LNCS (Tutorial Volume). Springer, 2007, pp. 220–270.

[19] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach". In: *Proc. TACAS 2002*. Vol. 2280. Lecture Notes in Computer Science. Springer, 2002, pp. 52–66.

[20] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176. URL: http://doi.acm.org/10.1145/357172.357176.

[21] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: http://www.bitcoin.org/bitcoin.pdf.

[22] Rafael Pass, Lior Seeman, and Abhi Shelat. "Analysis of the Blockchain Protocol in Asynchronous Networks". In: *Proc. of EUROCRYPT 2017*. Vol. 10210. Lecture Notes in Computer Science. Springer, 2017, pp. 643–673.

[23] Pierre-Yves Piriou and Jean-Francois Dumas. "Simulation of Stochastic Blockchain Models". In: *14th European Dependable Computing Conference, EDCC 2018, Iaşi, Romania, September 10-14, 2018*. IEEE Computer Society, 2018, pp. 150–157. DOI: 10.1109/EDCC.2018.00035. URL: https://doi.org/10.1109/EDCC.2018.00035.

[24] George Pîrlea and Ilya Sergey. "Mechanising blockchain consensus". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2018, pp. 78–90.

[25] Andrew Sward, Ivy Vecna, and Forrest Stonedahl. "Data Insertion in Bitcoin's Blockchain". In: *Ledger* 3 (2018). URL: https://ledgerjournal.org/ojs/index.php/ledger/article/view/101.

[26] Pak Chung Wong and J. Thomas. "Visual Analytics". In: *IEEE Computer Graphics and Applications* 24.5 (Sept. 2004), pp. 20–21. ISSN: 0272-1716.

[27] Daniel Davis Wood. "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER". In: 2014.

[28] Alexei Zamyatin et al. "Flux: Revisiting Near Blocks for Proof-of-Work Blockchains". In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 415.

[29] Zixuan Zhang, Michael Zargham, and Victor M. Preciado. "On modeling blockchain-enabled economic networks as stochastic dynamical systems". In: *Applied Network Science* 5.1 (2020), p. 19. DOI: 10.1007/s41109-020-0254-9. URL: https://doi.org/10.1007/s41109-020-0254-9.