A Formal Analysis of the Bitcoin Protocol

Laneve Cosimo, Adele Veschetti

A Formal Analysis of the Bitcoin Protocol

Cosimo Laneve
Dept. of Computer Science and Engineering, University of Bologna – INRIA Focus, Italy
cosimo.laneve@unibo.it

Adele Veschetti
Dept. of Computer Science and Engineering, University of Bologna – INRIA Focus, Italy
adele.veschetti2@unibo.it

Abstract

We study Nakamoto's Bitcoin protocol that implements a distributed ledger on peer-to-peer asynchronous networks. In particular, we define a principled formal model of key participants – the miners – as stochastic processes and describe the whole system as a parallel composition of miners. We therefore compute the probability that ledgers turn into a state with more severe inconsistencies, e.g. with longer forks, under the assumptions that messages are not lost and nodes are not hostile. We also study how the presence of hostile nodes mining blocks in wrong positions impacts on the consistency of the ledgers. Our theoretical results agree with the simulations performed on a probabilistic model checker that we extended with dynamic datatypes in order to have a faithful description of miners' behaviour.

2012 ACM Subject Classification Theory of computation → Formalisms

Keywords and phrases Bitcoin, Distributed consensus, Distributed ledgers, Blockchain, `PRISM`, forks

Digital Object Identifier 10.4230/OASIcs.Gabbrielli.2020.2

Related Version A full version of the paper with the proofs of the main statements is available at `http://cs.unibo.it/~laneve/papers/LaneveVeschetti.pdf`.

1 Introduction

Bitcoin is a distributed application that implements a ledger on peer-to-peer asynchronous networks that are dynamic (nodes may either join or leave) [20]. This technology is particularly critical because it manages and transfers relevant assets in the form of cryptocurrencies.

The basic problem of implementing a distributed ledger on a dynamic peer-to-peer asynchronous network is the management of inconsistent updates of the ledger performed by different nodes, which are called *forks*. This problem, known as distributed consensus in the literature, has been proved to be unsolvable since 1985 [9]. To overcome this shortcoming, the Bitcoin protocol uses an ingenious breakthrough: 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 [10]. Overall, the protocol is very complex and the current research is actively involved in understanding all the critical points that a potential attacker might use. We refer to [24] for an overview of possible attacks to Bitcoin.

Following [13, 11, 22, 23], we study the foundational principles of the Bitcoin algorithm in a formal way, by defining a clean and principled model of the key participants – the *miners*. These miners are stateful nodes communicating with each other by means of asynchronous messages that either announce a transaction (which defines a particular event) or create and broadcast a block that contains (the encoding of) a set of transactions. Once blocks are received, the miners validate them (they verify the correctness of the transactions therein) and, if this process succeeds, add the block to the local copy of the ledger.

As said above, this setting cannot give a global consensus on miners' ledgers because the underlying system is distributed. In fact, in this context, blocks may be delivered in a wrong order, may be duplicated, may be lost, or may be produced by hostile nodes. The Bitcoin algorithm uses several expedients to overcome these issues. First of all, the algorithm controls the generation of new blocks in order to be much less frequent with respect to the broadcast delay (which is around 2 seconds). In particular, in Bitcoin, miners are committed to solve a computationally hard problem in order to mine a new block (and they are rewarded with new Bitcoins when this happens). The complexity of the problem is set in such a way that mining occurs every 10 minutes (the technique is known as *proof of work*). In addition, the Bitcoin algorithm uses *ledgers* that are *trees of blocks* with *a pointer to a leaf node at maximal depth* called *handle*. The (eventual) consistency is not guaranteed on the ledgers (that may be different), but on the *blockchain* of the ledgers, i.e. the chain of blocks starting from the handle to the root node, called *genesis block* (which is assumed to be always the same). In this context, the addition of a new block to the ledger is a critical operation because, besides connecting the block to its parent (every block records the parent node), it may also change the handle (and therefore the corresponding blockchain) if the height of the ledger increases.

In our modelling of the Bitcoin protocol we intentionally leave out a number of details, such as what can go into a transaction or into a block, the exact specifics of the proof of work algorithm, and the validation process. We also overlook standard issues of distributed systems, such as the loss of messages or miners that may become either inactive or may show up in the system at runtime. In our setting, the network and the miners are modelled by means of stochastic processes where actions have rates. These rates are the formal artifice we use for expressing the latency of the network and the time required by miners to solve the computational problem (which is inversely proportional to the so-called *hashing power*). Overall, our model is simple and rigorous, which are, in our opinion, fundamental criteria for reasoning about properties of blockchain-based algorithms and for gaining trust in their basic principles. Once the basic properties have been analyzed and understood, one can address other, possible more complex, scenarios of distributed systems.

Our contribution. The formal model for defining the Bitcoin protocol is an extension of `PRISM` [18]. `PRISM` has been chosen for two reasons. First, because it is a simple process calculus with a formal stochastic semantics that uses rates of actions as parameters of an exponential distribution, which is a standard feature of Bitcoin actions of mining and broadcasting [1, 26, 8, 4]. Secondly, because `PRISM` has a tool for analysing stochastic systems that can be used for complementing our theoretical results with practical simulations.

However, as it is, `PRISM` falls short to model faithfully the Bitcoin protocol because it misses the datatypes of blocks and ledgers. Therefore, in Section 2, following the description in [20], we have defined the values of blocks, queues and ledgers and the corresponding operations. The extension of `PRISM`, called `PRISM+`, with the foregoing datatypes is defined in Section 3. In `PRISM+` a system is a parallel composition of modules that interact on actions that have in common. These actions may update the internal states of the modules (including ledgers and queues) and the next state of the system is defined in terms of a *race condition* between possible actions. The operational semantics of `PRISM+` is also reported in Section 3.

The Bitcoin protocol is defined in Section 4 as a `PRISM+` system consisting of a `NETWORK` module and a set of `MINER` modules. The `NETWORK` has a set of queues, one for every miner, which store the new blocks created by the miners. The blocks in the queues of `NETWORK` are retrieved by the miners through an explicit action. These actions and the corresponding rates allow us to implement the delay and the nondeterminism of the broadcast. The `MINER`

may either (*i*) mine a new block, or (*ii*) retrieve a block from the network and add it to the local queue, or (*iii*) take a block from the local queue and try to add it to the ledger. In case (*i*) the block is added to the local ledger and sent to the `NETWORK` in order to be inserted to the queues of the other miners. In case of (*ii*), the block taken from the network is not inserted into the ledger because the parent block may be still missing. Instead, it is inserted in the local queue of the miner that extracts blocks from time to time with the action (*iii*).

The `PRISM+` system allows us to compute the probability of devolving into a "larger inconsistency", e.g. transiting from a state with a fork of length m to a state with a fork $m+1$. This work, which has required a time-consuming analysis of the stochastic transition system, has given a formula that is parametric with respect to the number of nodes, their hashing power and the latency of the network. Henceforth, it has been possible to analyze different scenarios by tuning the rates of the corresponding actions. For instance, given the current rate-values of the Bitcoin system, the probability of reaching a state of fork of length 2 is less than 10^{-3}.

In Section 5 we apply the same technique for studying an attack to Bitcoin that has been already discussed in [20]: the presence of hostile nodes mining new blocks in positions that are different from the correct one (blocks are not inserted at maximal depth). The probability that we compute depends on the hashing power of the attacker and the depth m of the fork created by the hostile node. For example, if `BTC.com`, which is a cluster currently retaining the 14,1% of the Bitcoin hashing power, decided to become hostile, then the probability to create an alternative attacker chain and achieving consensus from the other nodes is 8^{-m}.

In the companion paper [2] we discuss the implementation of the library for blocks, queues and ledgers that extends `PRISM` and we analyze the results of simulations with different values of the rate parameters of the `PRISM+` system in Section 4. Remarkably, the results of the simulations are compliant with the upper bounds defined by our formulas and, for completeness, they are also highlighted in our pictures. (Actually `PRISM+` has a scalability issue: due to the state explosion of the Bitcoin model, the simulations were performed on systems with about twenty nodes.)

We analyze related works in Section 6 and report our concluding remarks in Section 7.

For space constraints, the proofs of our main statements are not included. They are reported in the full paper at `http://cs.unibo.it/~laneve/papers/LaneveVeschetti.pdf`.

2 Blocks, queues and ledgers

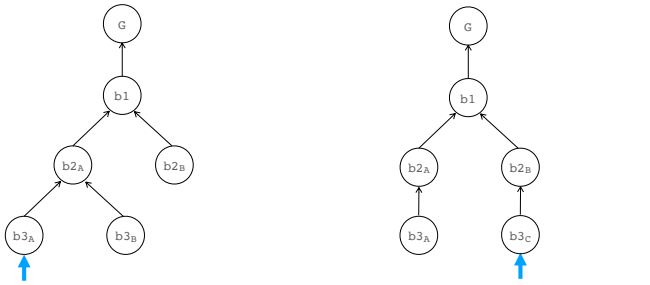
The Bitcoin protocol will be defined by a `PRISM` program, a process calculus with a stochastic semantics and an automatic analyzer of continuous-time Markov chains. However, datatypes used in Bitcoin cannot be modelled in `PRISM`; therefore we extend the language with `block`, `ledger` and `queue` data types and in the following we discuss this extension.

A basic component of the Bitcoin database is the `block`, which records transactions that are going to be certified, mining rewards, its hash value and a pointer to its parent. In this paper we abstract from many informations in blocks because they are not essential in the analysis of Sections 4 and 5 and we focus on the connections between blocks. Therefore, a `block` is a pair $(\texttt{name}, \texttt{father})$, where `name` uniquely identifies the block and `father` is the name of the previous block to which it is connected. Names will be represented by pairs $\texttt{mid}^n$, where `mid` is the name of the miner that mined the block and `n` is a number uniquely identifying the block. The operation that creates blocks is $\texttt{NewB}(\texttt{mid}, n, \texttt{p})$, which returns a block $(\texttt{mid}^{n+1}, \texttt{p})$, where `p` is the father name.

The Bitcoin protocol also uses a further datatype: the *bag of blocks*, namely sets of blocks that must still be appended to a ledger. We implement bags as *queues*, e.g. lists of blocks $[\mathtt{b}_0, \mathtt{b}_1, \cdots, \mathtt{b}_n]$ with the standard operations on queues:

- the empty queue is noted `[]`;
- `isEmpty(Q)` returns *true* if `Q` $=$ `[]`, *false* otherwise.
- the topmost block of a queue `Q` is given by `top(Q)`. When `Q` is empty, `top(Q)` returns *null*.
- the operation that inserts a block `b` at the end of a queue `Q` is `enqueue(Q,b)` (it returns a queue);
- the operation that removes the topmost block from a queue `Q` is `dequeue(Q)` (it returns a queue);
- the operation that removes the topmost block from a queue `Q` and inserts it at the end of the queue (because it cannot be added to the ledger) is `deq_enq(Q)` (it returns a queue). When `Q` is empty, `deq_enq` returns the empty queue.

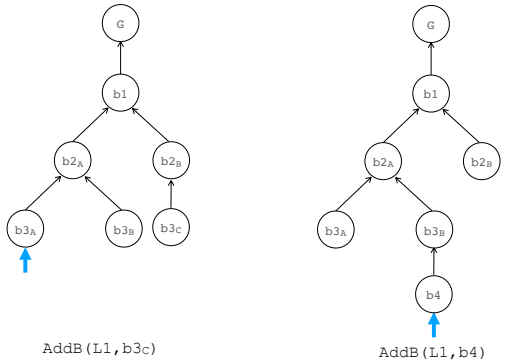
The Bitcoin database is an append-only tree whose nodes are blocks and it is called *ledger*. A `ledger L` is a pair $\langle \mathtt{T}; \mathtt{p} \rangle$, where `T` is the *tree of blocks*, e.g. a set of blocks where each block points to its own parent, and `p`, called *handle*, is the name of a leaf at maximal depth. The root of the tree is the *genesis block* and noted $(\mathtt{gen}^0, \mathtt{gen}^0)$. The handle of a ledger `L` is given by `handle(L)`. The *blockchain of L*, noted `L` $\uparrow$, is the sequence $(\mathtt{b}_0, \mathtt{b}_1, \mathtt{b}_2, \dots)$ such that $\mathtt{b}_0$ is the handle of `L` and, for every i, $\mathtt{b}_{i+1}$ is the parent of $\mathtt{b}_i$ (therefore the last block of the sequence is the genesis block). We illustrate ledgers by means of trees where nodes contain the name of the block and the unique exiting arrow is the pointer to its parent; the handle is represented by a tick arrow pointing to a leaf block at maximal depth. For example, the following picture illustrates two ledgers.



the ledger L1 – L1 $\uparrow = (b3_A, b2_A, b1, \mathtt{G})$ the ledger L2 – L2 $\uparrow = (b3_C, b2_B, b1, \mathtt{G})$

where $\mathtt{L1} = \langle \{\mathtt{G}, (\mathtt{b1}, \mathtt{gen}^0), (\mathtt{b2}_A, \mathtt{b1}), (\mathtt{b2}_B, \mathtt{b1}), (\mathtt{b3}_A, \mathtt{b2}_A), (\mathtt{b3}_B, \mathtt{b2}_A)\}; \mathtt{b3}_A \rangle$ (`G` is the genesis block).

A key operation on ledgers is the *addition* of a new block to the ledger, written `AddB(L,b)`, that returns a ledger where `b` is connected to the block pointed by `b`. This operation may change the handle of the ledger. In particular, the handle of `AddB(L,b)` is equal to the handle of `L` if the new block has not changed the maximal depth of the tree; it is a pointer to `b` if this block has a depth strictly greater than the maximal one of `L`. For example, considering the ledgers `L1` and `L2` in the foregoing picture, let $\mathtt{b2}_B$ be the parent of $\mathtt{b3}_C$ and $\mathtt{b3}_B$ be the parent of `b4`. The ledgers `AddB(L1,`$\mathtt{b3}_C$`)` and `AddB(L1,b4)` are



In these cases, the handle of `AddB`(`L1`,$\texttt{b3}_C$) is the same of `L1`, while this is not so for `AddB`(`L1`,`b4`) because the depth of the tree is changed.

It is also possible that a block cannot be added to a ledger because the parent block is not in the ledger. We use the boolean function `canAdd`(`L`, `b`) that returns *true* or *false* according to `b` can be added to `L` or not, respectively. `canAdd`(`L`, `b`) also returns *false* when `b` is *null*.

3 The modelling language: PRISM+

The language we use to analyze the Bitcoin protocol is an extension of `PRISM` [18] with the data types ledger, set and block. We call the language `PRISM+`.

To define `PRISM+` we use a set of *action names* A, ranged over a, b, ..., a set of *module names* ranged over $\texttt{M}$, $\texttt{M}_1$, ..., and a set of *variables*, ranged over by x, y, z. Let α ranges over $A \cup \{\varepsilon\}$, where ε indicates *no-action*; let also ρ range over reals (called `double`).

A `PRISM+` program `P` is a parallel composition of *modules*, that is

$$\texttt{P} \;=\; \texttt{M}_1 \parallel \cdots \parallel \texttt{M}_n$$

where $\texttt{M} \parallel \texttt{M}'$ is the parallel composition of modules `M` and `M'` synchronizing only on actions appearing in both `M` and `M'`. Let *actions*(`M`) be the set of actions in A that occur in `M`. A module `M` is defined by the syntax

$$\begin{array}{rcl}
\texttt{M} & ::= & \texttt{module}\ \texttt{M}\ \texttt{:}\ \texttt{D}\ \texttt{C}\ \texttt{endmodule} \\
\texttt{D} & ::= & \texttt{T}\ x = v\ \texttt{;} \;\mid\; \texttt{T}\ x = v\ \texttt{;}\ \texttt{D} \\
\texttt{T} & ::= & \texttt{int} \mid \texttt{double} \mid \texttt{bool} \mid \texttt{block} \mid \texttt{ledger} \mid \texttt{queue}
\end{array}$$

That is, a module has a name, a sequence `D` of *local variable declarations* with initializations and a *set of commands* `C`. It is assumed that pairwise different modules in a `PRISM+` program have different names and have also different local variables names.

Sets of commands `C` are written $c_1\ \texttt{;}\ \cdots\ \texttt{;}\ c_m$, where every c has the form:

$$\begin{array}{rcl}
c & ::= & [\alpha]\, e \to \sum_{i\in I} \rho_i : \texttt{upd}_i \\
\texttt{upd} & ::= & \varepsilon \;\mid\; x' = e\ \&\ \texttt{upd} \\
e & ::= & v \;\mid\; x \;\mid\; e\ \texttt{op}\ e \;\mid\; \texttt{!}e \\
v & ::= & \mathit{true} \;\mid\; \mathit{false} \;\mid\; \mathit{integers} \;\mid\; \mathit{doubles} \;\mid\; \mathit{ledgers} \;\mid\; \mathit{queue} \\
 & & \mid\; \mathit{blocks} \\
\texttt{op} & ::= & - \;\mid\; + \;\mid\; * \;\mid\; = \;\mid\; \neq \;\mid\; \&
\end{array}$$

In a command $[\alpha]\, e \to \sum_{i\in I} \rho_i : \texttt{upd}_i$, α may be either empty or an action, e, called *guard*, is a boolean expression over all the variables in the program (including those belonging to other modules), and the right hand-side of the arrow describes a *transition*. In particular, when α is empty, if e is true then one of the corresponding updates may be performed. Each

update is defined by giving new values of *the variables in the module*, possibly as a function of other variables. Each update has also a rate, which will be given to the corresponding transition. Updates are written with the prime symbol: $x' = e$ means that, if v is the value of e in the current state then the value of x in the *next state* is v. We assume that, in an update $x_1' = e_1 \ \& \ \ldots \ \& \ x_n' = e_n$, left hand-side variables are all different.

When α is an action then the transition must be performed simultaneously with the other modules in parallel that have the same action (i.e. the modules *synchronize*). This is the standard CSP parallel composition [15]. The rate of the overall transition is equal to the product of the individual rates. Since the product of rates does not always meaningfully represent the rate of a synchronised transition, `PRISM` uses the technique to make exactly one action *active*, with a generic rate, and all the others *passive*, with rate 1. The rate of a synchronization is therefore defined by the unique active action.

Semantics. The semantics of a `PRISM+` program is defined as a transition system whose states s are maps $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ where $\{x_1, \ldots, x_n\}$ is the set of local variables of the program's modules. The transition relation uses the following auxiliary definitions:

- $s[x \mapsto v]$ is the state

$$(s[x \mapsto v])(y) \stackrel{\text{def}}{=} \begin{cases} v & \text{if } y = x \\ s(y) & \text{otherwise} \end{cases}$$

- $[\![e]\!](s)$ returns the value of an expression e in the state s. The value is computed by replacing the variables with their values in s and evaluating the operations. The formal definition is omitted because standard.
- $[\![\texttt{upd}]\!](s)$ returns the state s' defined as follows:

$$[\![x_1' = e_1 \ \& \ \ldots \ \& \ x_n' = e_n]\!](s) \stackrel{\text{def}}{=} s[x_1 \mapsto [\![e_1]\!](s), \ldots, x_n \mapsto [\![e_n]\!](s)]$$

The transition relation of `PRISM+` is defined in Table 1 where we let $\mathcal{M}$ range over parallel compositions of modules and we assume $\|$ to be commutative. We use the judgment $\texttt{P} \Vdash s \xrightarrow{\alpha,\rho} s'$ meaning that the program `P` transits from s to s' with an action α and rate ρ. The auxiliary judgment $\mathcal{M} \Vdash s \xrightarrow{\alpha,\rho} \texttt{upd}$ collects all the updates in the synchronizing modules in $\mathcal{M}$ (according to our assumptions, different updates modify different variables). Rule [Upd] defines the semantics of a command. We write $c \in \texttt{M}$ if $\texttt{M} = \texttt{module M : D C endmodule}$ and $c \in \texttt{C}$. If e is true, then an update $\texttt{upd}_i$ is enabled with rate ρ_i and label α. The update $\texttt{upd}_i$ is a set of evaluated variables expressed as a conjunction of assignments. Rule [Sync] collects commands of synchronizing modules. We notice that the rate is the product of the rates of every single transition, which is actually the one of the unique active transition. Rule [Nosync] enables the interleaving of transitions (because of commutativity of $\|$, it also covers the symmetric rule). A `PRISM+` program is a parallel composition of modules; its semantics is described in [Program].

`PRISM+` supports different kinds of probabilistic formalisms; in this contribution we focus on CTMCs models [17], which are tuples $(\mathit{States}, s_{\mathit{init}}, \mathbf{R}, L)$ where:

- *States* is a countable set of states;
- $s_{\mathit{init}} \in \mathit{States}$ is the initial state; the initial state of a `PRISM+` program

$$\texttt{P} = \prod_{i \in 1..n} \texttt{module M : D}_i \ \texttt{C}_i \ \texttt{endmodule}$$

  is $[\![\texttt{D}_1 \ ; \cdots ; \ \texttt{D}_n]\!]$, where $[\![\texttt{T}_1 \ x_1 = v_1 \ ; \cdots ; \ \texttt{T}_k \ x_k = v_k]\!] = [x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]$;
- $\mathbf{R} : \mathit{States} \times \mathit{States} \to \mathbb{R}_{\geq 0}$ is a transition rate matrix,
- $L : \mathit{States} \to 2^{AP}$ is function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in the state.

Table 1 The semantics of the PRISM language.

$$\text{[Upd]}\quad \frac{[\alpha]\, e \to \sum_{i\in I} \rho_i : \mathtt{upd}_i \in \mathtt{M} \qquad [\![e]\!](s) = \mathit{true}}{\mathtt{M} \Vdash s \xrightarrow{\alpha,\rho_i} \mathtt{upd}_i}$$

$$\text{[Sync]}\quad \frac{\mathcal{M} \Vdash s \xrightarrow{a,\rho} \mathtt{upd} \qquad \mathcal{M}' \Vdash s \xrightarrow{a,\rho'} \mathtt{upd}'}{\mathcal{M} \parallel \mathcal{M}' \Vdash s \xrightarrow{a,\rho\times\rho'} \mathtt{upd}\&\mathtt{upd}'}$$

$$\text{[Nosync]}\quad \frac{\mathcal{M} \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd} \qquad \alpha \notin \mathit{actions}(\mathtt{M})}{\mathcal{M} \parallel \mathtt{M} \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd}}$$

$$\text{[Program]}\quad \frac{\mathtt{M}_1 \parallel \cdots \parallel \mathtt{M}_n \Vdash s \xrightarrow{\alpha,\rho} \mathtt{upd} \qquad \mathtt{P} = \mathtt{M}_1 \parallel \cdots \parallel \mathtt{M}_n}{\mathtt{P} \Vdash s \xrightarrow{\alpha,\rho} [\![\mathtt{upd}]\!](s)}$$

A transition rate matrix assigns rates to each pair of states, which are used as parameters of the exponential distribution. A transition from state s to s' is possible only if $\mathbf{R}(s,s') > 0$. When multiple commands with the same update and that lead to the same state s' are enabled, the corresponding transitions are combined into a single transition whose rate is the sum of the individual rates. Furthermore, when there are several s' with $\mathbf{R}(s,s') > 0$, a *race condition* occurs: the transition triggered determines the next state. Technically, the time spent in s before a transition occurs is exponentially distributed with the *exit rate* of the state s:

$$E(s) = \sum_{s'\in S} \mathbf{R}(s,s')$$

Thus, the probability of leaving a state s within t seconds is $1 - e^{-tE(s)}$. Additionally, the choice between the transitions is independent of the time at which it occurs. This means that, if the state s has n outgoing transitions labeled with rates $\rho_1, \ldots, \rho_n$, then the probability that the j-th transition is taken is $\rho_j/(\sum_i \rho_i)$.

4 The abstract modelling of Bitcoin and its analysis

Bitcoin realises a distributed ledger on a peer-to-peer network of *miners*, which are processes that create blocks of the ledger and forward them to the nodes of the network. The Bitcoin system written in PRISM+ is

$$\mathtt{MINER}_1 \parallel \cdots \parallel \mathtt{MINER}_n \parallel \mathtt{NETWORK}.$$

where $\mathtt{MINER}_i$ and NETWORK are the modules defined in Listing 1.

In Listing 1, miners are defined from line 6 to 25. Every miner $\mathtt{Miner}_i$ has five state variables: a state variable $\mathtt{Miner}_i$_STATE, the last block $\mathtt{b}_i$ added to the ledger; the local ledger $\mathtt{L}_i$ that represents miner's view of the state of the system, a counter $\mathtt{c}_i$ of the mined blocks, and a queue $\mathtt{QMiner}_i$ that stores the blocks received by the network and that must be added to $\mathtt{L}_i$. $\mathtt{Miner}_i$ behaves as follows:

```
1  // states of Miner_i: Init = 0, Winner = 1
2  // mR = 1/600 is the Bitcoin mining rate
3  // hR_i is the percentage of hashing power of Miner_i, 0≤hR_i≤1
4  // r_i is the communication delay rate of Miner_i
5
6  module Miner_i
7    integer Miner_i_STATE = Init;
8    block b_i = (gen^0,gen^0);
9    ledger L_i = ⟨{(gen^0,gen^0)};gen^0⟩;
10   integer c_i = 0;
11   queue QMiner_i = [];
12
13   [] Miner_i_STATE=Init -> mR×hR_i : L_i'= AddB(L_i,NewB(Miner_i,c,handle(L_i))
14                                    & c_i'= c_i+1 & b_i' = NewB(Miner_i,c,handle(L_i))
15                                    & Miner_i_STATE'= Winner;
16
17   [] Miner_i_STATE=Init&canAdd(L_i,top(QMiner_i)) -> r : QMiner_i'= dequeue(QMiner_i)
18                                                        & L_i'= addB(L_i,top(QMiner_i));
19
20   [] Miner_i_STATE=Init&!canAdd(L_i,top(QMiner_i)) -> r : QMiner_i'= deq_enq(QMiner_i);
21
22   [addBlock_i] Miner_i_STATE=Init -> r_i : QMiner_i'= enqueue(QMiner_i,top(Q_i))
23
24   [addBlock_i] Miner_i_STATE=Winner -> r_i : Miner_i_STATE'= Init;
25 endmodule
26
27 module Network
28   integer n = numberOfMiners;
29   queue Q_1 = []; ...; queue Q_n = [];
30         ...
31   [addBlock_i] (Miner_i_STATE=Winner) -> 1:
32                                 for ((j∈ 1..n) & (j≠i)) do (Q_j'= enqueue(Q_j,b_i)) ;
33
34   [addBlock_i] Miner_i_STATE=Init&!isempty(Q_i) -> 1 : Q_i'=dequeue(Q_i);
35         ...
36 endmodule
```

Listing 1 Simplified model of Bitcoin.

lines 13-15: it may mine a new block. This operation has a rate $\texttt{mR} \times \texttt{hR}_i$ that indicates the miner's rate of generating new blocks ($\texttt{hR}_i$ is the miner's hashing power, while mR is the difficulty level of the crytopuzzle [20]; this is how we abstract away from the proof-of-work technique for mining blocks). When a block is created by a miner – operation NewB –, it is added to the local ledger – operation AddB – and it is stored in the variable $\texttt{b}_i$. The state of the miner becomes Winner.

line 24: when Miner's state is Winner, the miner synchronizes with Network using the action $\texttt{addblock}_i$; the Network stores the new block in the bags of the other miners. The state of $\texttt{Miner}_i$ is set back to Init.

lines 17-18: it may add a block to the ledger from the local queue. The predicate $\texttt{canAdd}(\texttt{L}_i, \texttt{top}(\texttt{QMiner}_i))$ verifies that the parent of the block on top of the queue $\texttt{QMiner}_i$ is already stored in $\texttt{L}_i$. The corresponding updates on $\texttt{L}_i$ and $\texttt{QMiner}_i$ are performed. The time spent in doing this action is simulated by the rate r. Clearly, this rate is much higher than the other rates because it corresponds to local management operations of the Miner (therefore, the probability that a Miner tries to add a block in his ledger is way higher than the probability of receiving a new block or mining).

lines 20: it may try to add a block to $\texttt{L}_i$ that cannot be added either because parent's block is still not stored in $\texttt{L}_i$ or because $\texttt{QMiner}_i$ is empty. In this case, the block is enqueued in $\texttt{QMiner}_i$ (thus guaranteeing a fair behaviour).

line 22: it may receive a block from the network through the action $\texttt{addBlock}_i$. In this case the block is added to $\texttt{QMiner}_i$. The synchronization on $\texttt{addBlock}_i$ has a rate $\texttt{r}_i$, which simulates the latency of the network. In fact, as explained in [6], the communication delay across the Bitcoin network can be also approximated by an exponential distribution.

The `NETWORK` module is defined in Listing 1, lines from 27 to 36. It simulates the broadcast of new blocks to the miners. In particular, the module has one queue per miner that stores the messages (the blocks) to be delivered to the corresponding miner. When a node i mines a new block, the block is added to every miner's queue, except the one of miner i (line 32).

Properties. In the remaining part of the section we compute the probability of the Bitcoin system defined in Listing 1 to devolve into inconsistent states, e.g. into a state where at least two nodes have different ledgers. In order to ease our arguments, among the possible states of the stochastic transition system obtained from the model, we select those where the blocks have all been delivered. This scenario is usual in Bitcoin because the rate of block delivery is much higher than the one of mining. For example, the nodes that have not yet received the last block after 40 seconds are less than 5%, whilst blocks are mined every 10 minutes [6].

▶ **Definition 1.** *A state of a Bitcoin system is called* completed *when there is no block to deliver (every $\mathtt{Q}_i$ in `NETWORK` is empty) and the blocks in the local queues of $\mathtt{MINER}_i$ have already been inserted in the corresponding ledgers (every $\mathtt{QMiner}_i$ in $\mathtt{MINER}_i$ is empty).*

▶ **Proposition 2.** *Let P be a completed state of a Bitcoin system and let L_1 and L_2 be two ledgers in different nodes. Then the trees of L_1 and L_2 are equal. Therefore, if $L_1 \neq L_2$ then $\mathtt{handle}(L_1) \neq \mathtt{handle}(L_2)$.*

▶ **Definition 3.** *Let L_1 and L_2 be two ledgers and let*

- *m_1 be the length of $L_1\uparrow$,*
- *m_2 be the length of $L_2\uparrow$,*
- *h be the length of the maximal common suffix of $L_1\uparrow$ and $L_2\uparrow$.*

We say that L_1 and L_2 have a fork of length *k, where $k = \max(m_1 - h, m_2 - h)$.*

For the sake of simplicity, in the following theorem:

- we shorten $\mathtt{mR} \times \mathtt{hR_i}$ into r_{w_i};
- the rates r_1, …, r_n of actions $\mathtt{addBlock}_1$, …, $\mathtt{addBlock}_n$ are all considered identical by taking the parameter of the exponential distribution mean, which we call $\widehat{r}$ (actually these rates are parameters of an exponential distribution [6]);
- the rate r that corresponds to local management operations by Miners is approximated to 1 because the other rates are very small values less than 1.

▶ **Theorem 4.** *Let P be a completed state of a Bitcoin system consisting of n miners with ledgers $L_1, \ldots, L_n$, respectively, such that $L_1 = \cdots = L_k$ and $L_{k+1} = \cdots = L_n$ and $L_1 \neq L_{k+1}$. Let L_1 and L_{k+1} have fork of length m. Then the probability $\mathsf{Prob}(P_{\rightsquigarrow m+1})$ to reach a completed state with fork of length $m+1$ is smaller than ($R = \sum_{j=1}^{n} r_{w_j}$)*

$$\sum_{\substack{1 \leq i \leq n \\ H \subset \{1,\ldots,n\} \setminus i \\ i \leq k \;\Rightarrow\; j \in \{k+1,\ldots,n\} \setminus H \\ i > k \;\Rightarrow\; j \in \{1,\ldots,k\} \setminus H}} \Theta(i, |H|, j)$$

where

$$\Theta(i,\ell,j) = \frac{r_{w_i}\ r_{w_j}}{R\ (R + (n-1-\ell)\widehat{r})} \prod_{1 \leq h \leq \ell} \frac{h\ \widehat{r}}{R + (n-h)\widehat{r}} \prod_{1 \leq a \leq 2n-2-\ell} \frac{a\ \widehat{r}}{R + a\ \widehat{r}}\,.$$

It is worth to notice that $\mathsf{Prob}(P_{\rightsquigarrow m+1})$ of Theorem 4 depends on the number n of nodes, their hashing power r_{w_i} and the latency r_i of the network with respect to the node i. To explain the probability, assume to have a fork of length one due to miners having equal ledgers (since the state is completed) and two different handlers. Let $1, \cdots, k$ be the nodes with one ledger and $k+1, \cdots, n$ be the nodes with the other ledger. Assume that a node $1 \leq i \leq k$ mines a new block; the probability will be $\frac{r_{w_i}}{R}$. The new block is then communicated to a set H of nodes that immediately add it to the local ledger. This operation happens with probability $\left(\prod_{1\leq h\leq |H|} \frac{h\ \widehat{r}}{R+(n-h)\widehat{r}}\right)$. At this point, in order to obtain a fork of length 2, a node $j \in \{k+1, \ldots, n\} \setminus H$ must mine a block as well. The probability of this operation is $\frac{r_{w_j}}{R+(n-1-\ell)\widehat{r}}$. Finally, every node receives the two mined blocks, which has a probability $\left(\prod_{1\leq a\leq 2n-2-\ell} \frac{a\ \widehat{r}}{R+a\widehat{r}}\right)$. Obviously, the same result can be obtained if the first node that mines a block belongs to the second partition ($j \in \{k+1, \ldots, n\}$). Henceforth, the probability to reach a completed state with fork of length 2 from the initial state is

$$\sum_{\substack{1\leq i\leq k\\ H\subset\{1,\cdots,n\}\setminus i\\ j\in\{k+1,\cdots,n\}\setminus H}} \Theta(i,|H|,j) + \sum_{\substack{k+1\leq j\leq n\\ H\subset\{1,\cdots,n\}\setminus j\\ i\in\{1,\cdots,k\}\setminus H}} \Theta(j,|H|,i)$$

which is exactly what stated in the theorem.

Using a technique similar to Theorem 4 we may compute the probability that a Bitcoin system in a completed consistent state (the nodes have all the same ledger) devolves into an inconsistent state. In this case, the proof is simpler than Theorem 4 because every node may mine after the first one.

▶ **Proposition 5.** *Let P be a completed state of a Bitcoin system consisting of n miners having ledger $\mathtt{L}$. The probability $\mathsf{Prob}(P_{\rightsquigarrow 1})$ to reach a completed state with fork of length 1 is smaller than ($R = \sum_{j=1}^{n} r_{w_j}$)*

$$\sum_{\substack{1\leq i\leq n\\ H\subset\{1,\ldots,n\}\setminus i\\ j\in\{1,\ldots,n\}\setminus H}} \Theta(i,|H|,j)$$

where

$$\Theta(i,\ell,j) = \frac{r_{w_i}\ r_{w_j}}{R\ (R+(n-1-\ell)\widehat{r})} \prod_{1\leq h\leq \ell} \frac{h\ \widehat{r}}{R+(n-h)\widehat{r}} \prod_{1\leq a\leq 2n-2-\ell} \frac{a\ \widehat{r}}{R+a\ \widehat{r}}\,.$$

In order to bear some numerical results, we instantiate our probability with realistic rates. In [20], the time a miner takes to create a block is exponential with parameter θ, which represents the probability that the miner solves the cryptopuzzle problem in a given time-slot [1]. It follows that $\theta = h/D$, where h is miner's hashing power and D is the cryptopuzzle difficulty set by the protocol in order to set constant to 10 minutes the average duration between two blocks. In our encoding, θ is represented by r_{w_i}, therefore $r_{w_i} = h_i/D$ and, taking the current hashing power distribution of the Bitcoin system illustrated in Figure 1, and letting $D = 600$, we obtain the channel rates of the main pools in the Bitcoin system. As regards the broadcast of messages in the Bitcoin protocol, it is a combination of the transmission time and the local verification of the block. From [6] we know that in a Bitcoin environment, the broadcast can be approximated as an exponential distribution with mean time 12.6 seconds. Therefore we may assume that every r_i is 1/12.6.

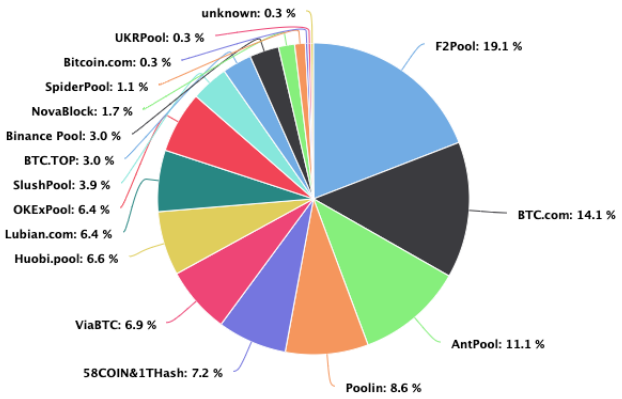


Figure 1 Hashrate distribution of Bitcoin mining pools on May 2020.
Source: `https://www.blockchain.com/`.

In the following Figures, we compare the outputs of our probability formula in Theorem 4 when the nodes are either 1000 (green line) or 17 (blue line). Furthermore, we also highlight the results obtained via the simulation with 17 nodes (red line) from the companion paper [2] because they are compliant with our upper bounds. We did not run simulations on larger sets of nodes because they took too much time (around 48 hours per simulation on a Virtual Machine with 8 VCPU and 64 GB RAM). We have also computed the formula in Theorem 4

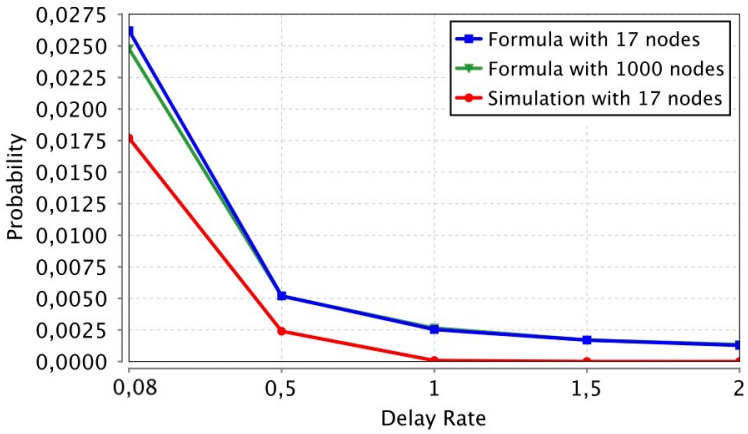


Figure 2 Probability of reaching a fork of length 1 by varying the broadcast delay.

with 10000 nodes: the output has not been displayed because it overlaps with the case of 1000 nodes. The first analysis we present in Figure 2 is the computation of the probability of reaching a state where at least two different blockchains differ for one block (fork of length 1) by varying the broadcast delay. In particular, Figure 2 compares the outputs of our probability formula (both with 17 and 1000 nodes) with results of the simulation we have done with 17 nodes representing the main pools in the Bitcoin system. The reader may notice that the probability decreases with the increase of the communication delay rate. This follows from the remark that the higher is the rate, the smaller is the expected time for the transition to occur. We notice that, with rate $\widehat{r} = 0.08$, we obtain results in line with those of [6]. In particular, for a broadcast delay with mean 2, we obtain that the probability of a

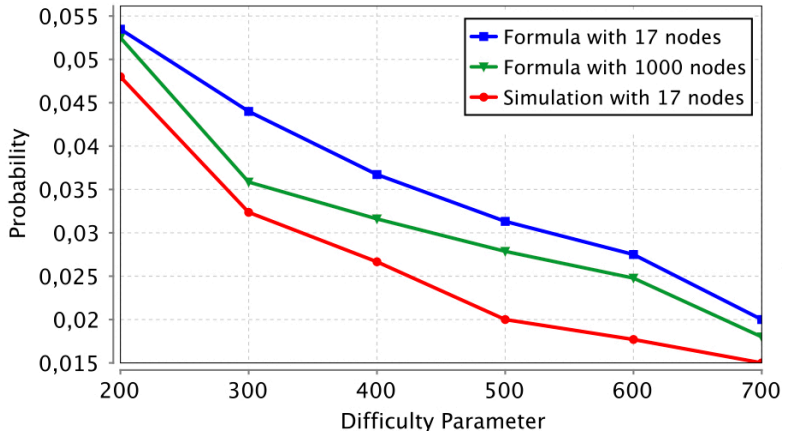


Figure 3 Probability of a fork of length 1 with different difficulty parameter.

fork of length 2 is very low. Figure 3 displays the probability of reaching a fork of length 1 by varying the cryptopuzzle difficulties (parameter D). The reader can observe that the probabilities computed by the formula, also in the case of 1000 nodes, is always an upper bound of the results obtained via simulation. Finally, Figure 4 illustrate the probability of reaching completed states with longer and longer forks. Also in this case, the results of our

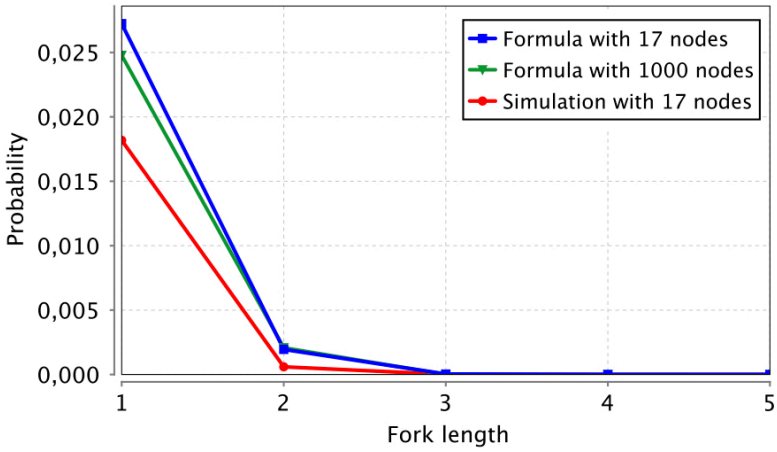


Figure 4 Probability of a fork of increasing length, comparison between formula and simulation results.

simulation are in line with those given by the formula both with 17 and 1000 nodes. In the case of 17 nodes, the probability computed is higher because each miner owns a larger amount of hashing power. Therefore, every miner is more likely to win the cryptopuzzle game. 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 every block at depth 6 is considered permanent in the Bitcoin blockchain (e.g. the majority of miners have consistent blockchains up to depth 6 with probability almost 1).

5 Analysis of a possible attack

In this section we model and analyze an attack to Bitcoin that has been described in [20], namely a hostile miner tries to create an alternate chain faster than the honest one. This scenario admits that a merchant can be convinced that a transaction has been accepted and then create a new branch of the chain, longer than the valid one, with some other transaction spending the same money (double spending attack).

Let $\texttt{MINER}_{Hack}$ be the dishonest miner; technically, its behaviour differs from $\texttt{MINER}_i$ because it mines on a block $\texttt{b}_{Hack}$ that is not the correct one (e.g. the handle of the ledger). In particular, the operation `NewB` in $\texttt{MINER}_{Hack}$ takes an ad-hoc block $\texttt{b}_{Hack}$ rather than $\texttt{handle}(\texttt{L}_{Hack})$. The definition of $\texttt{MINER}_{Hack}$ is given in Listing 2.

```
37 // states of Miner_Hack: Init = 0, Winner = 1
38 // mR = 1/600 is the Bitcoin mining rate
39 // hR_Hack is the percentage of hashing power of Miner_Hack, 0≤hR_Hack ≤1
40 // r_Hack is the communication delay rate of Miner_Hack
41
42 module Miner_Hack
43   integer Miner_Hack_STATE = Init;
44   block b_Hack = (gen^0,gen^0);
45   ledger L_Hack = ⟨{(gen^0,gen^0)};gen^0⟩;
46   integer c_Hack = 0;
47   queue QMiner_Hack = [];
48
49   [] Miner_Hack_STATE=Init -> mR×hR_Hack : L_Hack' = AddB(L,NewB(Miner_Hack,c,b_Hack))
50                                          & c_Hack' = c_Hack+1
51                                          & b_Hack' = NewB(Miner_Hack,c,b_Hack)
52                                          & Miner_Hack_STATE' = Winner;
53
54   [] Miner_Hack_STATE=Init&canAdd(L_Hack,top(QMiner_Hack)) ->
55                                       r : QMiner_Hack'=dequeue(QMiner_Hack)
56                                           & L_Hack'=addB(L_Hack,top(QMiner_Hack));
57
58   [] Miner_Hack_STATE=Init&!canAdd(L_Hack,top(QMiner_Hack)) ->
59                                       r : QMiner_Hack'= deq_enq(QMiner_Hack);
60
61   [addBlock_Hack] Miner_Hack_STATE=Init -> r_Hack :
62                                    QMiner_Hack' = enqueue(QMiner_Hack,top(Q_Hack))
63
64   [addBlock_Hack] Miner_Hack_STATE=Winner -> r_Hack : Miner_Hack_STATE'=Init;
65 endmodule
```

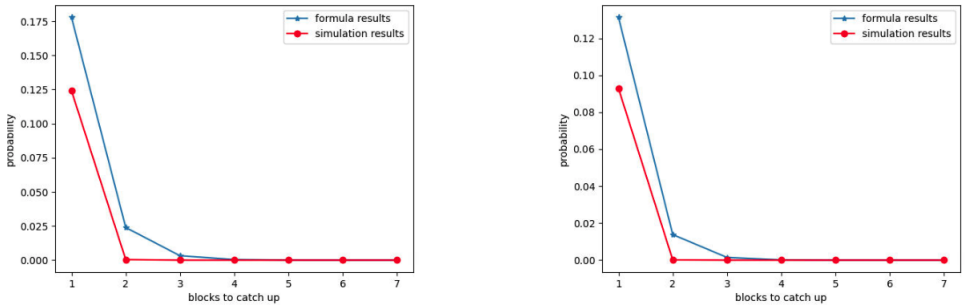
Listing 2 Simplified model of a dishonest Miner.

Following the same pattern of Section 4 and letting $r_{w_{Hack}} = \texttt{mR} \times \texttt{hR}_{\texttt{Hack}}$:

▶ **Theorem 6.** *Let P be a completed state of a Bitcoin system of n miners with exactly one that is hostile and let $r_{w_{Hack}}$ its mining rate. The probability* $\mathsf{Prob}(P_m)$ *to reach a completed state where the hostile miner has created an alternate chain longer than the honest one from $m, m \geq 1$, blocks behind is smaller than ($R = \sum_{j=1}^{n} r_{w_j}$ and we assume that, for every i, j, $\widehat{r} = r_i = r_j$)*

$$\sum_{k\geq 1}\Big[\Phi(r_{w_{Hack}},\widehat{r},R)^k\ \Big(\sum_{1\leq j\leq n-1}\Phi(r_{w_j},\widehat{r},R)\Big)^{k-1}\Big]^m$$

where $\Phi(r_w,r,R) = \dfrac{r_w}{R}\prod_{1\leq a\leq n-1}\dfrac{a\,\widehat{r}}{R+(n-a)\widehat{r}}$.



(a) F2Pool. **(b)** BTC.com.

Figure 5 Probability of a successful attack for two main pools of Bitcoin system.

As for Theorem 4, the technique used for demonstrating the above statement consists of analyzing the stochastic transition system. To explain the probability, assume to be in a completed state and compute the probability to reach a completed state in which the dishonest node has created an alternate chain from m blocks behind. We start by computing the probability that the dishonest node $\texttt{Miner}_{Hack}$ has caught up by one block. This kind of attack succeeds if $\texttt{Miner}_{Hack}$ mines one block and this happens with probability $\frac{r_{w_{Hack}}}{R}$. It may also happens that honest nodes mine k blocks and $\texttt{Miner}_{Hack}$ mines $k+1$ blocks in the same amount of time. Considering also the probability that the new blocks have been received by the miners, we obtain the formula

$$\sum_{k\geq 1}\Big(\frac{r_{w_{Hack}}}{R}\prod_{1\leq a\leq n-1}\frac{a\,\widehat{r}}{R+(n-a)\widehat{r}}\Big)^{k}\Big(\sum_{1\leq j\leq n-1}\frac{r_{w_j}}{R}\prod_{1\leq a\leq n-1}\frac{a\,\widehat{r}}{R+(n-a)\widehat{r}}\Big)^{k-1}$$

Therefore, the probability $\mathsf{Prob}(P_m)$ that $\texttt{Miner}_{Hack}$ creates an alternative chain faster than the honest nodes from m blocks behind is given by

$$\sum_{k\geq 1}\Big[\Big(\frac{r_{w_{Hack}}}{R}\prod_{1\leq a\leq n-1}\frac{a\,\widehat{r}}{R+(n-a)\widehat{r}}\Big)^{k}\Big(\sum_{1\leq j\leq n-1}\frac{r_{w_j}}{R}\prod_{1\leq a\leq n-1}\frac{a\,\widehat{r}}{R+(n-a)\widehat{r}}\Big)^{k-1}\Big]^{m}$$

which is what what Theorem 6 states. It is worth to notice that this technique is different from the one in [20], where Nakamoto assumed *a priori* that the ratio between the blocks mined by the attacker and those mined by the honest miners is the expected value of a Poisson distribution. In particular, we do not assume that miners' behaviour can be described by a certain statistical model, therefore our context is less restrictive. We also notice that Poisson distribution expresses the probability of a certain event occurs in a time period, independently of the time since the last event. Thus, Nakamoto models the attack counting the number of minings of the attacker in an interval of time, assuming that the probability for success does not change during the experiment. In our case, the probability is computed as the attacker was a standard node and its mining activity was in competition with the same process of the other nodes.

In Figure 5 we illustrate the probability of a successful attack by an hostile node, depending on the number m of blocks to catch up. We analyze two scenarios that highlight the cases when two main Bitcoin pools (see Figure 1) decide to become hostile.

In each image we plot the results given by the formula (blue line) and the results obtained via simulation (red line) for two main miners of the Bitcoin system. As well as for the previous analysis, the probability given by the formula is an upper bound for the results

obtained via simulation. We derive from Figure 5 that the probability a hostile miner catches up from 1 block behind increases with the percentage of the hashing power and drops with the number of blocks to catch up.

6 Related works

The protocol used by Bitcoin was introduced by Haber and Stornetta [14] 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.

The formal analysis of the protocol by means of abstract models has been already done in [13, 11, 22, 23]. In [13], the author discusses the blockchain consensus in Bitcoin and Ethereum and compare them with the classic Byzantine consensus. The miners are defined in pseudo-code (without any semantics) and the analysis is probabilistic rather than stochastic. In [11], 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; Chain Quality 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 [23], Pirlea and Sergey propose a formalization of Bitcoin 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 Bitcoin protocol as a stochastic system (with exponential distribution of durations) and derive the properties by studying the model. In fact, the probabilities that we compute are, up to our knowledge, original. As regards stochastic models and Bitcoin, few recent researches use them to select optimal strategies for maximizing profit of a player [1] and for formalizing interactions between miners as a game [5, 3].

A number of researches address attacks to the Bitcoin protocol. The works [6, 25, 12] address the delays of communications and [25] also demonstrates that an attacker with more than 51% of the total hashing power could change the past transactions. A larger set of attacks is analyzed in [19, 11, 22], where it is also proved that the Bitcoin protocol is safe as long as honest miners are in the majority. In [21], Ozisik and Levine give a very detailed description of Nakamoto's double spending attack, gathering the mathematics for its modelling. The probability of a successful double spending attack in several scenarios (both fast and slow payments) is analyzed in [16]. Finally, a fully implemented attack against Ethereum blockchain, which covers both a network and a double spending attack, is delivered in [7]. In contrast with these contributions, our results are achieved by analyzing a stochastic transition system, rather than constraining miners' behaviour to adhere to a certain statistical model.

7 Conclusions

We have studied the probability that the blockchain protocol may devolve the ledger into inconsistent copies because of forks. Two cases have been analyzed: the first one is when the system consists of honest miners; the second one is when the system has an hostile node that

mines blocks in wrong positions. The adversary model used in this paper is not the best one an adversary can implement, but the analysis of further strategies is left to future work. Our results are gathered by modelling the Bitcoin system in a stochastic process calculus, PRISM+, which has also an automatic tool for analysing systems that exhibit random or probabilistic behaviours. PRISM+ extends PRISM [18] with a library that models Bitcoin datatypes, such as ledgers and blocks.

The main contribution of this paper is the formal demonstration of the probability that Bitcoin ledgers may devolve into inconsistent states, also in presence of attacks. Our probabilities are parametric with respect to the number of nodes, their hashing power and the latency of the network. This work has required a time-consuming analysis of the stochastic transition system. It turns out that our results comply with simulations performed on PRISM+ systems with at most 17 nodes because of scalability problems (see also [2]).

Our approach is, as far as we know, original and the technique can be applied to analyze other well-known attacks to the Bitcoin protocol, such as failures either of communications or of miners, the inception of new miners that may be hostile, etc. In the future research we also plan to model other blockchain protocols, such as Ethereum or the so-called Proof-of-Stake. The presence of a probabilistic model checker like PRISM+ will allow us to deliver simulation results without much effort. In this respect, we will try to mitigate the scalability issues we had up to now.

References

1 Bruno Biais, Christophe Bisiere, Matthieu Bouvard, and Catherine Casamatta. The blockchain folk theorem, 2018.

2 S. Bistarelli, R. De Nicola, L. Galletta, C. Laneve, I. Mercanti, and A. Veschetti. Stochastic modelling and analysis of bitcoin. Manuscript submitted for publication, 2020.

3 Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Proc. of SP 2015*, pages 104–121. IEEE Computer Society, 2015.

4 R. Bowden, H. Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. Block arrivals in the bitcoin blockchain. *CoRR*, abs/1801.07447, 2018.

5 Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proc. Computer and Communications Security*, CCS '16, pages 154–167. ACM, 2016.

6 Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Proc. 13th IEEE P2P*, pages 1–10. IEEE, 2013.

7 Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Double-spending risk quantification in private, consortium and public ethereum blockchains. *CoRR*, abs/1805.05004, 2018. arXiv:1805.05004.

8 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, June 2018. doi:10.1145/3212998.

9 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

10 A. Fox and E. A. Brewer. Harvest, yield, and scalable tolerant systems. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, pages 174–178, 1999.

11 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proc. of EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

12 Johannes Göbel, Holger Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Perform. Eval.*, 104:23–41, 2016.

13 Vincent Gramoli. From blockchain consensus back to byzantine consensus. *Future Gener. Comput. Syst.*, 107:760–769, 2020.

14 Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, January 1991. doi:10.1007/BF00196791.

15 Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

16 Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proc. of CCS'12*, pages 906–917. ACM, 2012.

17 M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

18 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. TACAS 2002*, volume 2280 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2002.

19 Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. *Available on line: http://nakamotoinstitute.org/research/anonymous-byzantine-consensus*, 2014.

20 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: http://www.bitcoin.org/bitcoin.pdf.

21 A. Pinar Ozisik and Brian Neil Levine. An explanation of nakamoto's analysis of double-spend attacks. *CoRR*, abs/1701.03977, 2017. arXiv:1701.03977.

22 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Proc. of EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 643–673. Springer, 2017.

23 George Pîrlea and Ilya Sergey. Mechanising blockchain consensus. In *Proc. 7th Certified Programs and Proofs, CPP*, pages 78–90. ACM, 2018.

24 Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, and Aziz Mohaisen. Partitioning attacks on bitcoin: Colliding space, time, and logic. In *39th IEEE Conference on Distributed Computing Systems, ICDCS 2019*, pages 1175–1187. IEEE, 2019.

25 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Proc. of Financial Cryptography and Data Security 2015*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015.

26 Alexei Zamyatin, Nicholas Stifter, Philipp Schindler, Edgar R. Weippl, and William J. Knottenbelt. Flux: Revisiting near blocks for proof-of-work blockchains. *IACR Cryptology ePrint Archive*, 2018:415, 2018.