Simulation of stochastic blockchain models

Workshop on Blockchain Dependability

Pierre-Yves Piriou
EDF R&D
Chatou, France
pierre-yves.piriou@edf.fr

Jean-Francois Dumas
EDF R&D
Chatou, France
jean-francois.dumas@edf.fr

***Abstract*—This paper build the foundations of a simulation tool for blockchain-based applications. It takes advantage of the huge expressiveness and extensibility of PyCATSHOO framework to deal with the important variability of blockchain implementations and properties of interest. A simple stochastic model of generic blockchain-style distributed consensus system and associated performance indicators are proposed (performance in terms of consistency and ability to discard *double-spending* attacks). Monte Carlo simulations are applied to assess the indicators and determine their sensitivity to the variation of input parameters.**

***Index Terms*—Blockchain, Markov process, stochastic automata, Monte Carlo simulation, consistency, double-spending attack, PyCATSHOO**

I. Introduction

Blockchain technology recently benefits from a widespread interest because of its huge potential for securing decentralized applications. In practice, it refers to an important range of implementations (*Bitcoin* [1], *Ethereum* [2], *Hyperledger* [3], *etc.*) sharing a common purpose, basically: to register data on a cryptographic ledger written by a peer-to-peer network performing a protocol to ensure a consensus (an edition is symbolized by a block chainage). Those implementations are characterized by a set of primitives: consensus protocol, type of registered data, fork resolution rule, *etc.* Each of them have advantages and drawbacks and for a given use case, it is often non-obvious to determine *a priori* what implementation will be the most relevant. A tool to assess dependability and performance of a solution soon in the design process would be a major asset to assist the engineering of blockchain-based applications. The tool should be based on a generic blockchain model, that can be used in the raw to evaluate properties across implementations or be customized to fit with a particular one. This paper introduces PyCATSHOO framework to build the foundations of such a tool. PyCATSHOO is a Python library shaped to build hybrid stochastic automaton developed by EDF R&D. It comes with a Monte Carlo simulation [4] engine to assess probabilistic attributes of the model, in an easier way than with a tedious mathematical analysis of the model. In particular, this paper shows how well-known -while difficult to prove- results on blockchain consistency and ability to discard a *double-spending* attack can be computed by easy to set up Monte Carlo simulations.

This work has been funded by the EDF R&D project DURIN (Dependable Uses of Reliable blockchaIN).

Section II compares the paper to a selection of related works. In order to clarify our approach while staying as generic as possible, a lightweight stochastic model of blockchain are proposed in section III (only the blocks appending and broadcasting aspects are considered). The model is thereafter implemented into PyCATSHOO framework in section IV. The concepts of blockchain protocol consistency and ability to prevent from double-spending attack are therefore interpreted into our stochastic framework and some related probabilistic indicators are assessed using PyCATSHOO in section V. Finally, section VI concludes this paper and proposes directions for future works.

II. Related works

Current state of the art on blockchain modelling can be split into two categories: deterministic and stochastic. The first one is generally associated to formal proof purposes, either on blockchain protocol itself or on smart contract built on top of it.

- [5] proposes a Coq-aided proven "agnostic" blockchain protocol (currently the consistency is ensured only if the network topology is a clique, but stronger guarantees are targeted by authors for future works).
- [6] exploits infinite Mealy machines' expressiveness to capture the blockchain construction process in a generic way and describes properties a protocol should possess to build a consistent blockchain, with two qualities of the criteria: *strong* and *eventual* (this contribution will be discussed in subsection V-A).
- [7] proposes a communicating automata model of the triptych: blockchain construction, smart contract, users behaviour (honest and hacker) and exploits the statistical model-checker BIP to quantify the risk for an implementation to not satisfy its specification ([8] proposes a NuSMV model with the same idea of three-fold behaviour and the same purpose of model-checking although differently developed).

Contributions on stochastic modelling of blockchain are mostly shaped to compute probabilistic attributes analytically, even if the last referenced paper below validates its model by a simulation.

- The original Nakamoto's paper introducing *Bitcoin* [1] provides the first results on assessing the risk that an attacker could win the race against honest peers in a blockchain.

- [9] shows the security of *Bitcoin* protocol, which is reduced to two properties: *persistence* and *liveness*. Actually these properties are based on probabilistic indicators, namely the *common prefix* between blockchain copies and the *chain quality* that measures the influence of adversarial peers on the blockchain. This approach has shown its genericity in [10], where it is applied to extend the proof on the more generic protocol GHOST [11] (adopted by several blockchains including *Ethereum*).
- [12] models the evolution of partition between honest and adversary nodes along the blockchain as a 1-dimensional random walk. This model allows them to demonstrate interesting results on safety of several blockchains, namely *Bitcoin-NG*, *PeerCensus* and *BizCoin* (this contribution will be discussed in subsection V-B).
- Finally, [13] models the mining process by an inhomogeneous Poisson process. It shows that when the hash rate increases exponentially, the difficulty control implemented by *Bitcoin* protocol works so that the block rate -i.e. the mean time between the mining of two consecutive blocks- converges to a constant (10 minutes in practice). Then it proposes an improvement of such control to speed up the convergence. Its analytic results are confirmed by simulation and confronted with actual *Bitcoin* and *Namecoin* histories.

While the above referenced contributions give complementary keys to build a blockchain generic model and define meaningful performance attributes, none of them propose a framework to ease the tuning of the model and exploit Monte Carlo simulation to assess attributes, which is a convenient way when models become complex[1].

III. Blockchain stochastic model

A basic stochastic model is proposed in this section to capture the block creation and broadcasting process. To stay as generic as possible, blocks are in this model abstract objects that should be elicited to capture the underlying ledger evolution (what implies to define the type of registered data and the registering mechanisms).

A. Blocktree data structure

A blockchain is actually a particular branch of a rooted tree, i.e. a directed acyclic graph such as all nodes have a unique father except one, called the root, which has none. In blockchain jargon, the root is called the *genesis block*. Calling $\mathcal{B}$ the set of blocks, we can formally define a blocktree.

Definition 1. A blocktree $bt \in \mathcal{BT}$ is a 2-tuple $\langle B, E\rangle$, where:

- $B \subset \mathcal{B}$, is a finite non-empty set of *valid*[2] blocks, including at least one element b_0 (the *genesis block*).
- $E \subset B^2$ such as it exists a unique path from b_0 to any other block.
  Formally, $\forall b \in B\backslash\{b_0\}, \exists n \in \mathbb{N}^*, \exists (b_1, ..., b_n) \in B^n |$
  $b_n = b \wedge \forall i \in [\![1, n]\!], (b_{i-1}, b_i) \in E$
  n is called the *depth* of b (and its associated path).

To expand a blocktree $bt = \langle B, E\rangle$ from a block $b \in B$ with a new block $b' \notin B$ results in the blocktree $bt' = \langle B \cup \{b'\}, E \cup \{b, b'\}\rangle$. The *expansion* operation is then a partial mapping from $\mathcal{BT} \times \mathcal{B}^2$ to $\mathcal{BT}$.

Moreover a blockchain protocol defines a total order $\succ$ over B, preserving E, i.e $(b, b') \in E \implies b' \succ b$. This order is the cornerstone of the protocol since the definition of the so-called blockchain is built on it.

Definition 2. Given a blocktree $bt = \langle B, E\rangle$ and a total order $\succ$ over B, the *blockchain* is the unique path from the *genesis block* b_0 to the *last block* $b_l = \max_{\succ} B$.

B. Distributed handling of the blockchain

A blockchain is built by a network of processes applying sequentially the *expansion* operation starting from the inital blocktree $\langle\{b_0\}, \emptyset\rangle$. In practice, every process refers to its own view of the blocktree and strives to build a consensus with each other on the shared blockchain while increasing its depth. For this purpose, they continuously try to "build" valid blocks. When such valid block is found by a process, it expands its local blocktree from its *last block* (i.e. the last block of its own view of the blockchain) and broadcasts it to the others. When a process receives a new block from another one, it updates its local blocktree expanding it with the new block. In practice, forks may actually be observed due to blocks broadcasting delay.

Building a new valid block is an operation that may take many forms depending on protocols. It consists at least to provide a *proof* that the process is legitimate to append a block. The two widely considered kind of such proof are called Proof-of-Work (PoW) and Proof-of-Stake (PoS). In case of a PoW-based protocol, the process has to solve a hard computational problem (often a constrained hashing), whereas for a PoS-based protocol, it has to show that it is deeply involved in the blockchain (in practice a proof that it holds a lot of tokens). We propose to model the chance for a process i to build a valid block by a unique parameter $m_i \in \mathbb{R}^+$, called the ***merit*** (which symbolizes for example the process *hashrate* in case of PoW or the amount of its *balance* in case of PoS). Then we introduce an abstract *oracle* that randomly chooses a process to build each new block according to their *merit* (a similar idea can be found in [6]). A blockchain protocol is usually designed in order that a block is appended regularly with a constant *mean block time* $t_b \in \mathbb{R}^+$ (e.g. 10 minutes for *Bitcoin*, 12 seconds for *Ethereum*). We propose then a Markovian model for the oracle behaviour.

Definition 3. The oracle behaves as a continuous Markov process which infinitely selects a new process i among a set

[1]Note that this list of article is a selection of contributions that have most inspire this work but do not target any completeness.

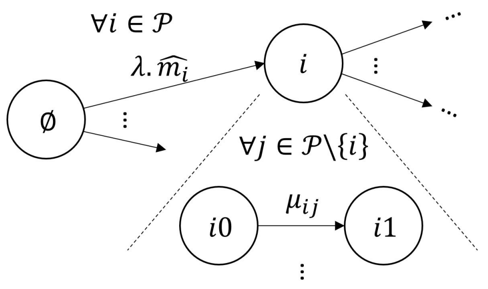
[2]The definition of a block's validity comes with the elicitation of a block.

of processes $\mathcal{P}$ to build a new valid block, according to its normalized merit $\hat{m}_i = \dfrac{m_i}{\sum_{j \in \mathcal{P}} m_j}$, with a rate $\lambda = \dfrac{1}{t_b}$.

The delay from a block creation by a process i to its reception by a process j depends on several network-related factors (like topology, bandwidth, instantaneous load). As a first approximation, we can abstract these factors introducing a global *mean network transit time* $t_n \in \mathbb{R}^+$. But we can be a bit more precise to consider network asymmetries, by defining a *mean network transit time* $t_{n,i}$ for each process i. With this refinement, the mean time to transfer a block between two processes i and j is $\dfrac{t_{n,i} + t_{n,j}}{2}$. Hence the last definition snippet of our model can be stated.

Definition 4. The reception by a process i of a block appended by a process $j \neq i$ occurs with a rate $\mu_{i,j} = \dfrac{2}{t_{n,i} + t_{n,j}}$.

Figure 1 represents by intention the system of Markov chains that can be built from Definition 3 and 4.



$\forall i \in \mathcal{P}$
$\lambda.\widehat{m_i}$
i
$\emptyset$
$\forall j \in \mathcal{P} \backslash \{i\}$
$i0$
μ_{ij}
$i1$

Fig. 1. Markov chain system modelling a blockchain protocol

C. Discussion

The proposed model makes several simplifying assumptions, what are discussed hereafter:

- In practice, for protocols that relying on particular consensus mechanisms (like Proof of Work), all distributed processes try to build a valid block in parallel. It is possible that several processes succeed this task in a very short period, what cause a fork between concurrent branches, until a new block is chained after one of them and is received by all other processes. Since our oracle selects a process with a probabilistic time, this scenario is still possible although less probable. The oracle could be refined to increase the chance for several processes to append a new block in a short period but it would be more difficult to parameter.
- For some other protocols, the time between to block creation is constant. This variant of the model could be easily set in the implemented model (we will see in next section that PyCATSHOO allows to declare deterministic delayed transitions as well as stochastic ones).
- All the distributed processes share a global clock for timestamping the blocks creation through the oracle abstraction. In reality, each process timestamps its blocks according to its own clock what may provoke local inconsistencies -what should be fixed by the protocol- and bias in network transit times. But we argue that at this modelling level, this phenomenon can be neglected and abstracted into the *mean network transit time*. Moreover, this assumption results in lowering significantly the simulation time what is essential to perform Monte Carlo simulation (for which a lot of histories has to be simulated).
- The merit of each process is assumed known and fixed during the scenario. To consider variable merits with uncertainties on their values constitutes a way to refine the model.

An advanced mathematical analysis of the model would allow to determine the bias it introduces in comparison to a given blockchain protocol, but it is not in the scope of this paper. We will see in section V that this lightweight model is sufficient to rediscover without pain well known results, what can either be observed on real implemented protocols execution (what spend a lot of time), or be obtained by a rigorous mathematical analysis. But reasoning on our model to solve the dependence between the parameters and indicators on blocktree shape and more generally, on protocol consistency is a tedious work (and even more so on more complex models). Next sections introduces PyCATSHOO framework and shows how to take advantage of it to assess these indicators.

IV. PyCATSHOO implementation

PyCATSHOO[3] is the combination of a Python library to describe distributed hybrid stochastic automata and a tool to perform Monte Carlo analysis on models. It is a convenient approach to perform probabilistic assessments on systems that combines both discrete and continuous behaviours[4]. The main exploitation of PyCATSHOO at EDF is for model based safety analysis of power plants (discrete and continuous behaviours are caused by respectively failure/repair events of components and evolution of physical variables such as pressure and temperature). In this section, some principles of PyCATSHOO paradigm are recalled (it can also be found in [14] and [15]), then the implementation keys of the model defined in section III in this framework are provided.

A. Reminder on PyCATSHOO principles

A PyCATSHOO model is a system of components that communicates through message boxes. Each component is defined by a 4-tuple $\langle V, B, A, R \rangle$, where:

- $V = I \cup E$ is a set of variables partitioned into a subset of internal variables and a subset of references to external variables (next we denotes by $\overline{V}$ the set of all possible valuations of V);
- $B \subseteq \mathcal{P}(V)$ is a set of message boxes declaring *output* and *input* ports through which *internal* and *external* variables are respectively *exported* and *imported*;

[3]PyCATSHOO is freely accessible at *http://pycatshoo.org* and should be open source soon.

[4]Note that the model introduced in this paper is purely discrete.

- A is a set of stochastic automata;
- $R = D \cup C$ is a set of evolution rules of the component state variables partitioned into a subset of rules that are applied on the occurrence of *discrete events* and the subset of rules determining the *continuous dynamics*.

A stochastic automaton $a \in A$ being defined by a 3-tuple $\langle S, s_0, T \rangle$, where:

- S is a set of states;
- $s_0 \in S$ is the initial state;
- $T = T_s \cup T_d$ is a set of transitions t which is itself defined by a 4-tuple $\langle s, g, d, p \rangle$, where:
  - $s \in S$ is the origin state;
  - $g \in \overline{V} \longrightarrow \{True, False\}$ is a guard built on the variables determining the validation condition of the transition;
  - $d \in \mathbb{R}^+$ is a parameter used to generate a delay before firing a validated transition. If the transition is of kind *stochastic* ($t \in T_s$), the delay is randomly chosen according to an exponential law of parameter d, whereas if the transition is of kind *temporized* ($t \in T_d$), the delay is simply the value of the parameter d. Note that d can be specified using variables in V such that its value may change with the model evolution.
  - $p \in S \longrightarrow [0,1] | \sum_{s \in S} p(s) = 1$ is a probabilistic distribution on state space to select the destination state (if a single destination state $s \in S$ is possible, then $p(s) = 1$). Once again, p can be specified using variables in V.

An evolution rule $r \in C$ (determining the *continuous* dynamics of variables) are specified as ordinary differential (or not) equations, whose resolution for a given simulation instant is handled by the simulator engine. The other kind of evolution rule $r \in D$ (determining the *discrete* dynamics of variables) are specified as functions called *sensitive methods* because they are executed when a specified event occurs:

- when the simulation start (used to specify the initial assignation of the variables);
- when a transition is fired;
- when an automaton state is left or entered (the sequential order for the three events associated to the firing of a transition t from state s_1 to state s_2 is quite intuitive: *leaving* $s_1 \rightarrow$ *firing* $t \rightarrow$ *entering* s_2);
- when a referenced external variable moves (used to propagate the effects of an event occuring in an other component).

Finally, components can be connected through their message boxes. A connexion between x and y through their respective message boxes b_x and b_y is valid if and only if any imported variables by one is an exported variables by the other. Formally:

$$\begin{cases} b_x \cap E_x = b_y \cap I_y \\ b_y \cap E_y = b_x \cap I_x \end{cases}$$

The formal semantics of a PyCATSHOO model will not be detailed in this paper. Let us only state the general idea of what the simulator engine provides: a finite history of variables' assignments, randomly generated according to stochastic and deterministic evolution of the model which is specified by components' automata and rules. Then this engine can be exploited to assess probabilistic indicators on the model (e.g. the mean value of a variable) taking advantage of Monte Carlo simulations.

The main benefit to exploit PyCATSHOO framework is that it inherits from the expressiveness of Python itself. Indeed, although the types of PyCATSHOO variables are basic (*boolean*, *integer*, *float*, *string*), convenient intermediate objects can be created and manipulated through (discrete) evolution rules alongside the proper PyCATSHOO variables to ease the model specification.

B. PyCATSHOO model of blockchain

The model described in section III can be implemented into PyCATSHOO framework, defining three components, namely: the *Process*, the *Oracle* and the *Blocktree*. An overview of these components is depicted on Figure 2.

A block is implemented as a pure Python object, fully determined by a (unique) **`hash`**, its **`father`** block (*None* for the genesis block), a **`timestamp`** and the **`author`** process that build it (its **`depth`** is simply the depth of its father incremented). A Python dictionary stores all blocks and is used to retrieve a block instance given its hash. We define a basic order for blocks (this order is total since two blocks cannot be built at the same instant):

$$b_1 \succ b_2 \iff \vee \left| \begin{array}{l} b_1.depth > b_2.depth \\ \wedge \left| \begin{array}{l} b_1.depth = b_2.depth \\ b_1.timestamp > b_2.timestamp \end{array} \right. \end{array} \right.$$

The (unique) Blocktree component has always a perfect knowledge of already appended blocks. For a process (identified by a unique **`address`**), a block can be either known (yet received) or pending (not yet received). The block creation is scheduled by the oracle firing the transition from state **`waiting`** to state **`tokenGenerated`** (parameter λ is the inverse of **`meanBlockTime`** as stated in section III). When this transition is fired, the following instantaneous sequence is performed:

1) the sensitive method **`selectProcess`** randomly selects one of the process according to its **`merit`** and assign the variable **`tokenHolder`** with its **`address`**;
2) **`tokenGenerated`** becomes *True*[5] what triggers the transition from **`working`** to **`claimTkn`** for each process;
3) the transition from **`tokenGenerated`** to **`waiting`** is fired;
4) the transitions from **`claimTkn`** to **`tknHeld`** then to **`working`** are fired only for the selected process;

[5]To declare an automaton state as a message box port is a convenient syntactic way to define a boolean variable which is *True* whenever the state is active.

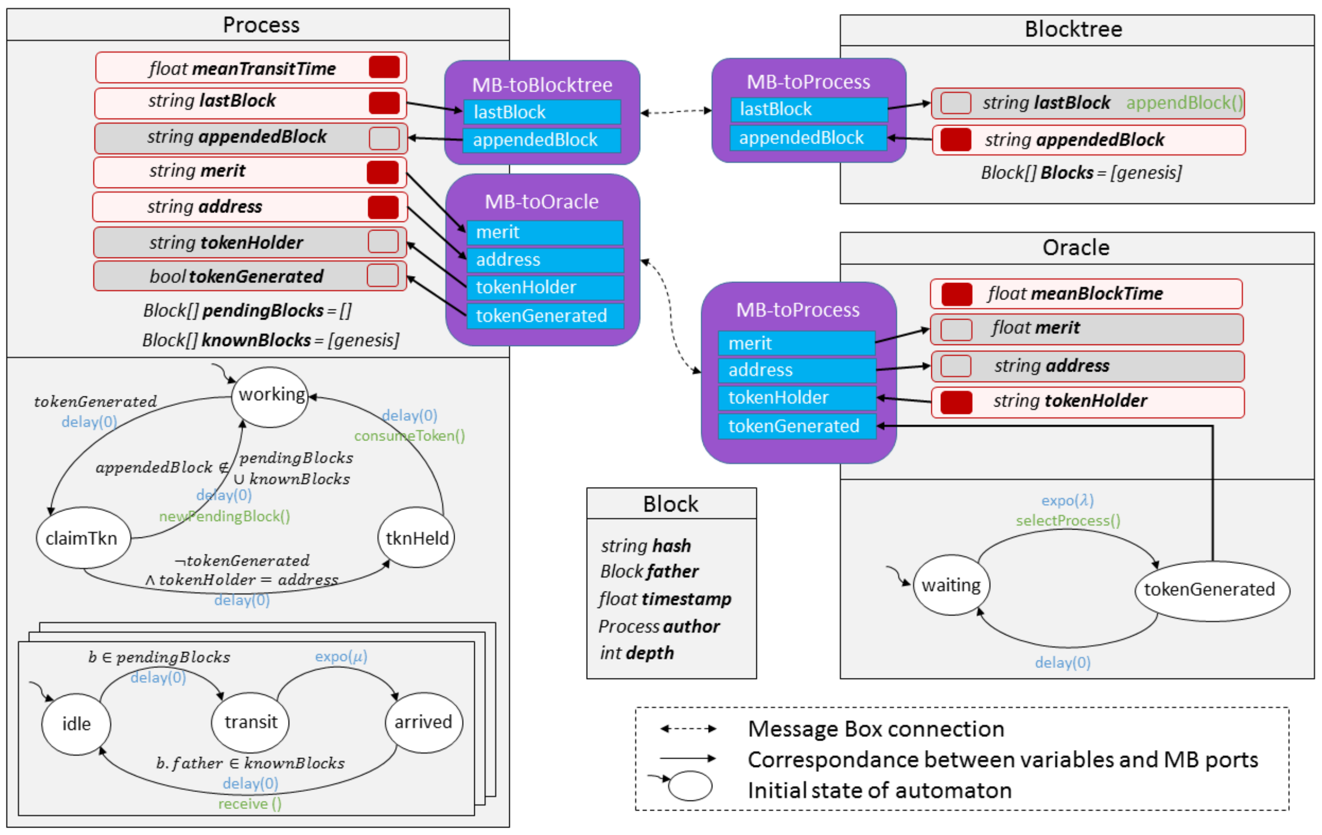


Fig. 2. Overview of the PyCATSHOO model of a blockchain generic protocol

5) the sensitive method **consumeToken** (of the token holder) creates a new instance of Block (whose the timestamp is the current simulation time and the father is the last block known by the process), appends it to **knownBlocks** and updates **lastBlock** with its hash;
6) the modification of a variable **lastBlock** has the effect to call the method **appendBlock** of the Blocktree component. The method append the new block to its list **Blocks**, then updates the variable **appendedBlock**;
7) the guard of the transition from **claimTkn** to **working** is now validated for all remaining processes. The transition is fired and the associated method **newPendingBlock** append the new block to their list **pendingBlock**;

Reception of pending blocks by processes is scheduled by a set of automata (parameter μ is computed from the variables **meanTransitTime** of the receiver and the block author as stated in section III). Because several blocks can be in reception in parallel, a block may arrived before its father, then the guard of the transition between **arrived** and **idle** ensures that a block is actually transferred from **pendingBlocks** to **knownBlocks** after its father. This transfer operation is performed by the sensitive method **receive** which also computes the new blockchain to update the **lastBlock** variable.

The code size is only around 500 lines in Python language[6]. As said in last subsection, our model can be easily enhanced in many ways, for instance to fit a particular blockchain paradigm, to consider faulty behaviour of processes or to take into account variable probabilistic input parameters.

Next section shows how the PyCATSHOO Monte-Carlo simulation engine can be exploited to assess performance indicators of a blockchain protocol in terms of consistency and ability to discard a double-spending attack.

V. Assessment of Blockchain Attributes

A. Blockchain consistency assessment

The consistency is a property of great interest to qualify the performance of a blockchain protocol. Informally, a protocol is *consistent* if its processors succeed to build a consensus on the blockchain. Several formal definitions are proposed in literature. In particular, [6] defines two kind of consistency criteria, built on the prefix relation (a chain c_1 prefixes another chain c_2 if and only if the last block of c_1 is an ancestor of the last block of c_2):

- the *strong consistency* holds when it exists a prefix relation between all processes' blockchains. In other words processes never fork.

[6]The code is available at *http://pycatshoo.org/Model Samples.html*

TABLE I
ASYMPTOTIC VALUES OF THE CONSISTENCY INDICATORS FOR SEVERAL PARAMETERS ASSIGNMENTS

(In each case, upper, middle and bottom values are respectively the consensus probability, the consistency rate and the worst process delay.)

r \ n	2	3	4	6	10	20	40	60	100
0.1	0.913	0.868	0.839	0.803	0.761	0.702	0.657	0.635	**0.598**
	0.955	0.938	0.930	0.922	0.914	0.909	0.908	0.908	**0.907**
	0.094	0.143	0.180	0.220	0.271	0.347	0.415	0.442	**0.505**
0.2	0.837	0.762	0.718	0.660	0.587	0.505	0.455	0.412	0.368
	0.912	0.884	0.870	0.858	0.844	0.832	0.831	0.830	0.832
	0.189	0.279	0.338	0.418	0.533	0.671	0.771	0.853	0.945
0.5	0.686	0.559	0.479	0.391	0.304	0.231	0.180	0.168	0.088
	0.823	0.766	0.735	0.705	0.683	0.663	0.656	0.655	0.646
	0.424	0.614	0.754	0.918	1.080	1.264	1.373	1.406	2.111
0.7	0.602	0.453	0.369	0.280	0.192	0.118	0.073	0.054	0.037
	0.769	0.698	0.665	0.627	0.598	0.579	0.572	0.568	0.559
	0.607	0.895	1.070	1.311	1.617	1.973	2.316	2.500	2.761
0.99	0.515	0.347	0.264	0.173	0.109	0.054	0.026	0.017	0.009
	0.715	0.625	0.586	0.538	0.513	0.484	0.475	0.467	0.463
	0.844	1.238	1.476	1.810	2.169	2.615	3.021	3.279	3.554

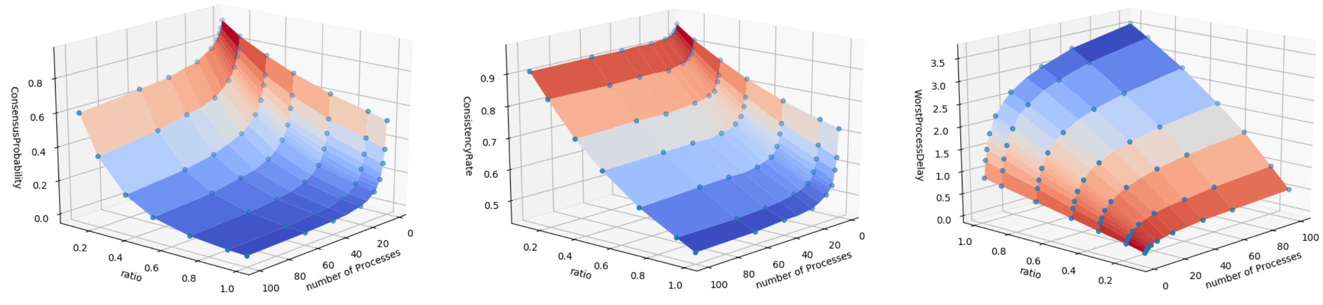


Fig. 3. Interpolation of asymptotic values of the consistency indicators for $2 \leq n \leq 100$ and $0.1 \leq r < 1$ *(red is a better consistency than blue)*

- the *eventual consistency* holds when the *greatest common prefix* between all processes' blockchains always eventually grows.

For our model, it is clear that the *strong consistency* (i.e. the fork probability equals 0) is guaranteed only if the mean network transit time t_n approaches 0. On the other hand, the perpetual eventual growth of the *greatest common prefix* is still possible while $0 < t_n < t_b$, although more or less frequent depending on parameters. Intuitively, the lower the number n of processes and the ratio $r = \dfrac{t_n}{t_b}$ are, the better the consistency is. To validate this intuition while refining the characterization of consistency, we introduce three indicators, which can be seen as three complementary metrics of consistency (next we call the *absolute* blockchain the most advanced among all locally viewed blockchains according to the order $\succ$):

- ***consensus probability***: the probability that all processes agreed on the absolute blockchain (higher is better).
- ***consistency rate***: the mean proportion of processes agreed on the *absolute* blockchain (higher is better).
- ***worst process delay***: the mean length difference between the absolute blockchain and the greatest common prefix (lower is better).

To ease the results understanding, we assume a perfect symmetry of the network and a perfect fairness between processes, formally:

$$\forall (i,j) \in \mathcal{P}^2, \begin{cases} t_{n,i} = t_{n,j} \\ m_i = m_j \end{cases}$$

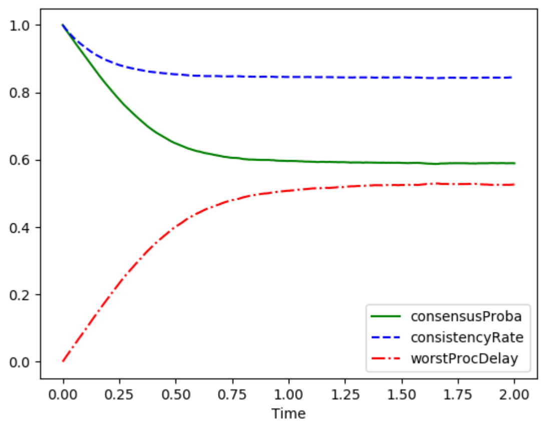


Fig. 4. Time evolution of the consistency indicators for $n = 10$ and $r = 0.2$

Figure 4 shows the time evolution of the three indicators for $n = 10$ and $r = 0.2$, estimated running a Monte

Carlo simulation (100000 histories performed in 9.5 minutes on a single core of an i7-6700HQ CPU). We can see that these indicators approach rapidly an asymptotic value (the simulation time is $2 * t_b$ then the mean number of appended block is only 2). Table I reports the values of these asymptotes for a set of parameters' assignments. An interpolation built from these points is represented on Figure 3. These results show that the consistency is more sensitive to variations of r than to variations of n. Moreover, the higher n and lower r are, the less consistency depends on them. As it happens, for classical blockchain protocols (typically *Bitcoin* and *Ethereum*), r is low and n is high. Extrapolating our results lead to confirm that classical blockchain protocols have a good consistency performance (when applied in a symmetric and fairness network of processes): the mean proportion of up-to-date processes is around 90%, with more than 50% chances that this proportion reaches 100%, and when not, the chances that delayed processes have a lag of more than one block is low (the *worst process delay* trend is around 0.5). We can then conclude that the probability that the *eventual consistency* will be violated for classical blockchains (that are consistent with the assumptions we made) are negligible.

B. Protocol ability to discard a double-spending attack

We propose now to analyse with our model another widely studied blockchain performance indicator: the risk that malicious processes manage to perform a double-spending attack. We consider the approach developed in [12] where three *Bitcoin*-based blockchain protocols are under analysis: *Bitcoin-NG*, *PeerCensus* and *BizCoin*. Our Monte Carlo simulation results meet the results obtained in this paper by a mathematical analysis. Due to a lack of place, we will detailed only the comparative study for the *BizCoin* protocol which corresponds to a compromise between Bitcoin-NG and PeerCensus.

The model
An adversary controls a proportion $\mu \in [0, 1]$ of the whole set of processes in order to perform double-spending attacks[7]. Consequently, the proportion of honest peers corresponds to $1 - \mu$. The state of the blockchain is defined by $B_k = (h, m)$ with k the epoch that indicates the state in which k blocks have been chained after the genesis block, h and m correspond respectively to the number of honest and malicious appended blocks taking into account the assumption that the genesis block is honest. Thus the only possible state of the blockchain at epoch 0 is $B_0 = (1, 0)$. At every epoch, with a blockchain in state $B_k = (h, m)$, the next block can be malicious with a probability μ making the next state becoming $B_{k+1} = (h, m+1)$ or honest with a probability $1 - \mu$ making the next state becoming $B_{k+1} = (h + 1, m)$. Formally:

$$\mathbb{P}\{B_{k+1} = (h+1, m) | B_k = (h, m)\} = 1 - \mu$$
$$\mathbb{P}\{B_{k+1} = (h, m+1) | B_k = (h, m)\} = \mu$$

Figure 5 depicts the Markov chain modelling this behaviour.

[7]μ here should not be confused with the mean reception rate introduced in section III

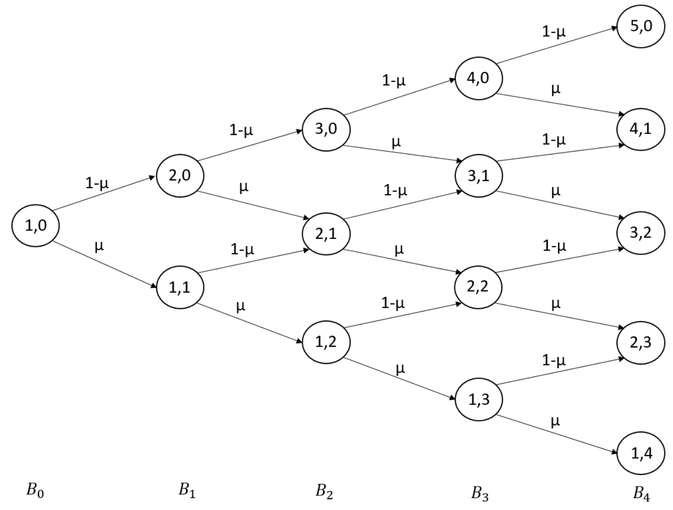


Fig. 5. Markov chain modelling the evolution of the partition between honest and malicious blocks in a blockchain

Finally, several assumptions are made in this article, that we interpret through the model introduced in section III:

- every processes (honest or not) have the same chance to append a new block, what corresponds to a perfect fairness (all merits are equal).
- the block generation time is constant (and not stochastic).
- the network transit time is not taken into account, what we can reproduce by setting the ratio r at an extremely low value.

BizCoin protocol
In this particular protocol, when a process submit a new block, there is a vote among the processes that have appended the $w-1$ last blocks ($w \geq 2$) to prevent a double-spending attempt. The state of the blockchain is *safe* ($\in S_w$) if less than one third of these $w - 1$ processes is malicious and polluted ($\in P_w$) otherwise. Formally (let $m_i = 1$ if the process that append the i-th last block is malicious and $m_i = 0$ otherwise):

$$S_w = \{(m_0, ..., m_{w-1}) \in \{0, 1\}^w | \sum_{i=0}^{w-1} m_i \leq (w - 1)/3\}$$

$$P_w = \{(m_0, ..., m_{w-1}) \in \{0, 1\}^w | \sum_{i=0}^{w-1} m_i > (w - 1)/3\}$$

Figure 6 depicts the discrete-time Markov chain modelling the *BizCoin* protocol for $w = 4$. Safe and polluted states are respectively coloured in green and red.

The asymptotic probability that the blockchain state is safe at epoch k (i.e. the probability to be in a green state in the Markov Chain) is given by theorem 3 in [12]:

$$\mathbb{P}\{W_k \in S_w\} = \sum_{l=0}^{(w-1)/3} \binom{w}{l} \mu^l (1 - \mu)^{w-l} \qquad (1)$$

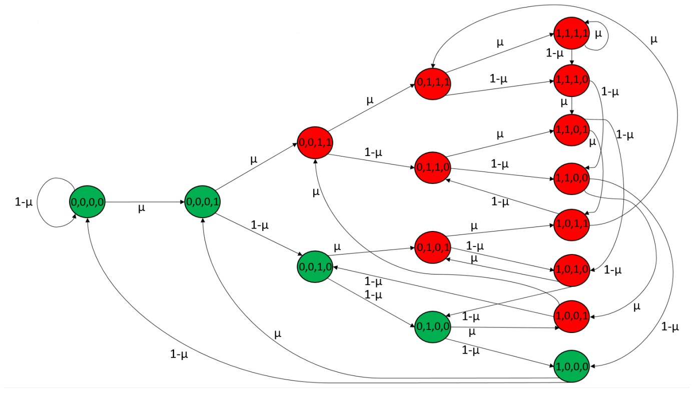


Fig. 6. Markov chain modelling *BizCoin* protocol for w=4

with its limit:

$$\lim_{w\to\infty} \mathbb{P}\{W_k \in S_w\} = \begin{cases} 0 & \text{if } \mu > 1/3 \\ 1/2 & \text{if } \mu = 1/3 \\ 1 & \text{if } \mu < 1/3 \end{cases}$$

Comparative study

On figure 7 are traced in plain line the results obtained using our simulator representing the proportion of safe execution of *BizCoin* depending on μ for four different values of w (what correspond to 1 hour, 6 hours, 1 day and 1 week, if we consider that a block is appended every 10 minutes, like with *Bitcoin*). Corresponding theoretical curves computed thanks to formula 1 are also traced in dashed line. We can see that both curves are overlapped.

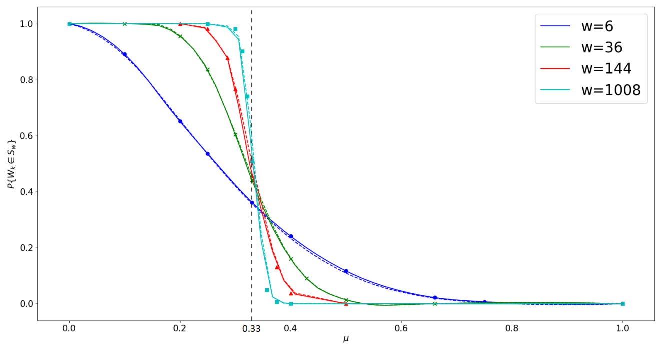


Fig. 7. Probability of being in a safe state depending on μ for different values of w

We can see that the more w increases the more the variation of the probability of having a blockchain in a safe state is concentrated around the value $\mu = 1/3$. This means that with a value of w high enough, if $\mu < 1/3$ the probability of observing a double-spending attack is insignificant.

Let us remind that this quality analysis omitted the network transit delays. Taking into account these delays in a mathematical analysis would be very tricky whereas made so much easier with the simulator as the only thing to change is the value of r.

VI. Conclusion and perspectives

An original although simple continuous-time stochastic model of blockchain protocols has been defined in this paper. Moreover, it proposes an implementation of the model in PyCATSHOO framework, that allows us to perform Monte-Carlo simulations to obtain interesting probabilistic results on consistency and ability to discard double-spending attacks of blockchain protocols.

More generally, this work lays the foundations of a simulation tool for blockchain-based application since the expressiveness of PyCATSHOO is wide enough to envisage many refinement ways: to consider dynamic evolutions of stochastic parameters, to model higher level layers (in particular transactions and ledger), to integrate smart contracts logic or to take into account the execution environment including off-chain continuous dynamics.

References

[1] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008. https://bitcoin.org/bitcoin.pdf.

[2] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, 2014. http://gavwood.com/paper.pdf.

[3] E. Androulaki et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *13th EuroSys Conference*, page 30, Porto, Portugal, April 2018. ACM.

[4] C.Z. Mooney. *Monte Carlo simulation*. Sage Publications, 1997.

[5] G. Pîrlea and I. Sergey. Mechanising blockchain consensus. In *7th SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pages 78–90, New York, NY, USA, 2018. ACM.

[6] E. Anceaume, A. Del Pozzo, R. Ludinard, M. Potop-Butucaru, and S. Tucci-Piergiovanni. Blockchain Abstract Data Type. Research report, Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France, February 2018.

[7] T. Abdellatif and K.-L. Brousmiche. Formal verification of smart contracts based on users and blockchain behaviors models. In *IFIP NTMS International Workshop on Blockchains and Smart Contracts (BSC)*, Paris, France, February 2018.

[8] Z. Nehai, P.-Y. Piriou, and F. Daumas. Model-checking of smart contracts. In *IEEE International Conference on Blockchain*, Halifax, Canada, August 2018. IEEE.

[9] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer.

[10] A. Kiayias and G. Panagiotakos. On trees, chains and fast transactions in the Blockchain. *Cryptology ePrint Archive*, page 545, 2016.

[11] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527, San Juan, Porto Rico, February 2015. Springer.

[12] E. Anceaume, T. Lajoie-Mazenc, R. Ludinard, and B. Sericola. Safety Analysis of Bitcoin Improvement Proposals. In *IEEE Symposium on Network Computing and Applications*, Boston, United States, October 2016. IEEE.

[13] D. Kraft. Difficulty control for blockchain-based consensus systems. *Peer-to-Peer Networking and Applications*, 9(2):19 pages, April 2016.

[14] H. Chraibi. Dynamic reliability modeling and assessment with PyCATSHOO: Application to a test case. In *International Conference on Probabilistic Safety Assessment and Management (PSAM)*, Tokyo, Japan, April 2013.

[15] H. Chraibi, J.C. Houdebine, and A. Sibler. Pycatshoo: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems. In *International Conference on Probabilistic Safety Assessment and Management (PSAM)*, Seoul, South Korea, October 2016.