Performance Evaluation of Stochastic Real-Time Systems with the SBIP Framework

Ayoub Nouri, Braham Lotfi Mediouni et al.

Verimag Research Report n^o TR-2017-6

September 27, 2017

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - Grenoble INP - UGA
Bâtiment IMAG
Université Grenoble Alpes
700, avenue centrale
38401 Saint Martin d'Hères
France
tel : +33 4 57 42 22 42
fax : +33 4 57 42 22 22
http://www-verimag.imag.fr/

Performance Evaluation of Stochastic Real-Time Systems with the SBIP Framework

Ayoub Nouri, Braham Lotfi Mediouni et al.

September 27, 2017

Abstract

The SBIP framework consists of a stochastic real-time component-based modelling formalism and a statistical model checking engine. The former is built as a stochastic extension of the real-time BIP formalism and enables the construction of stochastic real-time systems in a compositional way. The statistical engine implements a set of statistical algorithms for the quantitative and qualitative assessment of probabilistic properties. The paper provides a thorough introduction to the SBIP formalism and the associated verification method. In a second part, it surveys several case studies about modelling and verification of real-life systems, including various network protocols and multimedia applications.

Keywords: BIP Framework, Stochastic real-time BIP, Statistical Model Checking, LTL.

Reviewers: Marius Bozga, Axel Legay, and Saddek Bensalem.

How to cite this report:

```
@techreport {TR-2017-6,
  title = {Performance Evaluation of Stochastic Real-Time Systems with the SBIP Framework
},
   author = {Ayoub Nouri, Braham Lotfi Mediouni et al. },
   institution = {{Verimag} Research Report},
   number = {TR-2017-6},
   year = {}
}
```

1 Introduction

Stochastic models are of paramount importance in system design as they allow to capture uncertainties in systems behaviours and to account for variability induced by systems environments. Such models offer, in addition, a mean for high-level reasoning and for dealing with complex systems in an abstract and quantitative fashion. This is highly recommended, especially during the first phases of the design process, where the number of design choices is quite important. Modelling formalism enabling to incorporate stochastic aspects are thus substantial.

In model-based design, it is recommended to rely on the same modelling formalism to handle both performance aspects, e.g., energy consumption, and functional behaviour, e.g., timing constraints. This enables to perform functional verification and performance evaluation in a consistent manner. Many real-life systems operate under stringent timing constraints, e.g., real-time systems, which have the particularity to fail when missing well specified deadlines. Modelling formalism that enable to capture both stochastic behaviour and timing constraints are essential for building faithful system models at a high-level of abstraction, and to allow for their trustworthy assessment.

In this paper we present the SBIP framework which offers a stochastic real-time modelling formalism and a statistical model checking (SMC) engine for quantitative assessment of systems properties. The paper is an extended version of (2) that generalises the current stochastic modelling formalism (23). We rely on the BIP (Behaviour, Interaction, Priority) framework (7) which allows for building heterogeneous and complex system models in a compositional fashion.

The stochastic real-time BIP formalism allows for building components as stochastic timed automata and to compose them by using multi-party interactions in order to build the final system. In these components, time evolves continuously and is modelled by using the classical clock construct, introduced in timed automata (3). The proposed formalism enables for associating system events with timing constraints, e.g., an event only occurs in a particular time interval. The latter can be further annotated by urgency tags, which specify their level of urgency. Furthermore, in order to express uncertainty regarding events occurrences, it is possible to associate them with probability density functions. Hence, the precise moment of executing an event is scheduled according to that density function.

In the SBIP modelling formalism, we consider two types of events, namely *timed* and *stochastic*. The former are associated with time constraints expressed as lower and upper bounds over clocks valuations, as in timed automata. These events are scheduled with respect to a uniform or an exponential probability distribution as it is generally the case in several existing modelling formalism. Although several probabilistic behaviours, e.g., phase-type can be approximated by an exponential density, many real-life systems do not follow such a probabilistic evolution. Restricting to these density functions is not a realistic assumption in our opinion. To overcome this lack, we introduce *stochastic events*, which may be scheduled with respect to arbitrary density functions, e.g., Normal, Poisson.

Enabling arbitrary probability density functions adds more complexity at the level of the mathematical construction induced by this model. An important reason for restricting to exponential probability densities is their particularity of being memoryless, which generally induces Markovian models. That is, the choice of the next system state only depends on the current one. Conversely, arbitrary density functions are not generally memoryless and hence induce models taking into account the system history when moving to the next state. This leads to more general models, which in our case is a Generalised Semi-Markov Process (GSMP) (16).

The SMC engine provided by the SBIP framework for stochastic models assessment implements well known statistical analysis algorithms, namely *hypothesis testing* (28) and *probability estimation* (12). Statistical model checking is a novel technique proposed as a trade-off between purely analitycal verification techniques and purely simulation methods. It requires, as in the classical model checking setting, to build an operational model of the stochastic system of interest and to provide a formal specification of the property to verify, generally using temporal logic. SMC explores a sample of the stochastic model execution traces produced through simulation in order to estimate the probability for the system to satisfy that property. This statistical approach is receiving an increasing attention and is being adopted in a wide range of application domains such as in biology (10), for the assessment of communication protocols (5), multimedia applications (4), and avionics (6).

Outline The remainder of the paper is organised as follows. We discuss some related works in Section 2. Section 3 introduces the stochastic real-time BIP formalism and its simulation semantics. Section 4 recalls some principles of statistical model checking and presents the temporal logic used in the SBIP framework. Technical details about the structure of the SMC engine and its implementation are provided in section 5. In section 6, we survey the main case studies realised with the SBIP framework, and section 7 concludes the paper.

2 Related Works

Several frameworks exist for modelling and analysing stochastic systems. In this paper, we restrict ourselves to discuss frameworks following a model checking-like procedure. Other Methods related to the Queuing Theory and Network Calculus are beyond the scope of this discussion. The considered frameworks generally differ in three points, namely, the expressiveness of their modelling formalism, the proposed analysis technique, and the properties specification language they offer.

For instance, UPPAAL-smc (11; 14) supports *Stochastic Timed Automata* (STAs), which are general models including *Discrete* and *Continuous Time Markov Chains* (DTMCs and CTMCs) for system modelling and *Weighted Metric Temporal Logic* (WMTL) for properties specification. In addition to DTMCs and CTMCs Prism (18) allows to model *Markov Decision Processes* (MDPs) and *Probabilistic Timed Automata* (PTAs). For properties specification, it allows to use *Probabilistic Computation Tree Logic* (PCTL/PCTL*), *Continuous Stochastic Logic* (CSL), *Linear-time Temporal Logic* (LTL).

Other tools like Vesta (26) support, in addition to DTMCs and CTMCs, algebraic specification languages, i.e., PMaude (17). PlasmaLab (15) is a modular statistical model checker that may be extended with external simulators and checkers. Its default configuration accepts discrete-time models specified in the Prism modelling language and properties expressed in Probabilistic Bounded LTL (PBLTL) (23). Ymer (29) is one of the first frameworks to implement hypothesis testing algorithms. It considers GSMPs and CTMCs specified using a dialect of the Prism modelling language, and accepts both PCTL and CSL for requirements specification.

Our framework enables to model DTMCs, MDPs, and GSMPs. From a modelling perspective, it differs from Prism as the latter considers PTAs – as the underlying probabilistic and timed model – which incorporate non-determinism and are generally analysable using numerical probabilistic model checking. From this point of view, the SBIP framework is closer to the UPPAAL-smc and the Ymer frameworks. The latter allows to model GSMPs, whereas we consider GSMPs with fixed-delays events as in (9). The UPPAAL-smc provides a general stochastic timed semantics, i.e., STAs, however it is only limited to exponential and uniform density functions. Furthermore, the stochastic real-time BIP formalism allows for specifying urgency types on systems events. For properties specification, we rely on PBLTL for the moment, but we are planning to consider more expressive logic such as WMTL.

3 Stochastic Real-Time BIP

The *stochastic real-time BIP* framework reconciles the real-time and stochastic extensions of the BIP (7) framework. We recall that BIP has been introduced as a component-based framework where systems are obtained by composition of untimed atomic components with multiparty interactions, and coordinated using dynamic priorities. RT-BIP (1) extended BIP with real-time features and has (dense) real-time semantics based on timed automata concepts (3). S-BIP (23) extended BIP with stochastic features and has (discrete) stochastic semantics based on Markov chains.

In the newly proposed stochastic real-time BIP formalism, atomic components are defined as timed automata extended with stochastic timing constraints. Composition is performed as in BIP using multiparty interactions, that is, n-ary synchronisation among component actions. Priorities are not supported. The underlying semantics is defined as a Generalised Semi Markov Process (GSMP) where the interpretation of time is dense.

We start by defining the syntax of our model at the level of components and their composition. Then, we present the underlying stochastic simulation semantics.

3.1 Stochastic Real-Time Components

Stochastic real-time BIP components are essentially timed automata with urgencies, augmented with a new form of stochastic guards on clocks.

Let Δ be a set of density functions, that is, functions $\rho : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that $\int_0^\infty \rho(t)dt = 1$. We denote by $dom(\rho) = \{t \mid \rho(t) \neq 0\}$ the definition domain of ρ, that is, the set of values with a non-zero probability to occur. Let X be a set of clocks. We consider timed constraints (guards) c^t and stochastic constraints (guards) c^s on X, defined by:

$$c^t ::= \mathit{true} \mid x \sim k \mid x - y \sim k \mid c^t \wedge c^t \qquad\qquad c^s ::= x \bowtie \rho$$

where $x, y \in X$, $k \in \mathbb{R}_{\geq 0}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $\rho \in \Delta$. The meaning of timed constraints is as usual. A stochastic constraint $x \bowtie \rho$ holds iff $x \in dom(\rho)$. Nonetheless, in contrast to a timed constraint, it will enforce the specific stochastic distribution ρ on the values of x used to effectively satisfy the constraint, when used as a transition guard.

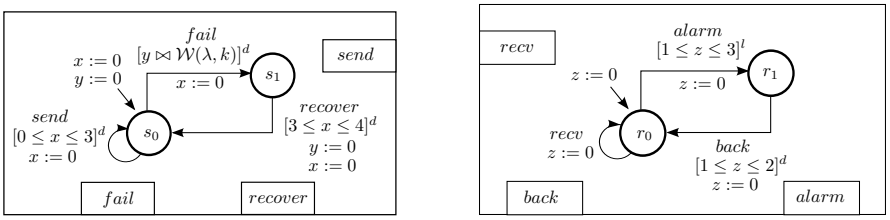
Definition 3.1 *A stochastic real-time BIP component B is an extended timed automaton (L, X, P, T, ℓ^0), where L is a finite set of locations, $\ell^0 \in L$ is the initial location, X is a finite set of clocks, P is a finite set of ports, and T is a finite set of transitions. Every transition is of the form (ℓ, p, g^u, r, ℓ'), denoted for more convenience as $\ell \xrightarrow{p\ g^u\ r} \ell'$, where $\ell, \ell' \in L$ are the source and target locations, $p \in P$ is the triggering port, g^u is a constraint g on X with an urgency $u \in \{lazy, delayable\}$, and $r \subseteq X$ is the set of clocks to be reset.*

The only noticeable difference between our definition of components and the timed automata concerns the meaning to control the progress of time. Usually, timed automata rely on location invariants and/or specific types of locations (e.g., committed) to explicitly constrain the time progress. In our case, we rely on urgency types of transitions with the following intuitive meaning. A $delayable$ transition (abbreviated to d) prevents time progress at the upper time bound in g, i.e., time is enabled to progress in the source location at most to that bound. In contrast, a $lazy$ transition (abbreviated to l) does not have any impact on the time progress. Such a transition might not be fired at all in spite of the upper bound in g, i.e., time is enabled to progress indefinitely in the source location.

We call a port timed (resp. stochastic) if it appears on transitions with timed (resp. stochastic) constraints. We tacitly restrict to components where every port is either timed or stochastic, but not both. Moreover, we restrict to components that are time-port deterministic, i.e., for a given source location ℓ, time t and port p, only one target location ℓ' can be reached.

Example 3.1 *The **Sender** component shown in Figure 1a has control locations s_0, s_1, ports* send, fail, recover *and clocks x, y. This component starts in s_0, where it may send data periodically in a specific time slot, defined through the timed constraint $[0 \leq x \leq 3]^d$. The **Sender** component may fail when executing the stochastic port fail. The latter is associated with the guard $[y \bowtie \mathcal{W}(\lambda, k)]^d$, where $\mathcal{W}(\lambda, k)$ is the* Weibull *probability density function with parameters λ, k, and $dom(\mathcal{W}) \subseteq \mathbb{R}_{\geq 0}$. This guard indicates that the component fails after some time scheduled according to $\mathcal{W}(\lambda, k)$. After a failure, the component recovers after some delay in $[3 \leq x \leq 4]^d$, where it goes back to s_0 and starts sending again according to the same time constraints.*

Example 3.2 *Figure 1b shows a second component, namely the **Receiver**, which has control locations r_0, r_1, ports* recv, alarm, back *and one clock z. The **Receiver** starts in location r_0, where z is set to 0. From this initial location, the component either receives some data through the self loop on r_0 labelled by the timed port recv (recv is a timed port with a guard set to true, i.e. it might be taken whenever the component is in r_0), in which case z is reset to 0. Or, it may fire the timed transition labelled by the port alarm and moves to location r_1, where the* recv *port is not enabled. One may think of this behaviour as a degraded mode, e.g., energy saving with an alarm. Note that the alarm port is associated with the timed constraint $[1 \leq z \leq 3]^l$. Since the latter is lazy, the upper bound 3 could be ignored and the alarm transition might not be fired. From location r_1, the component takes transition* back *after a delay specified through $[1 \leq z \leq 2]^d$, to r_0 and starts receiving again. This transition is delayable so it must be taken at most at $z = 2$.*



(a) Sender (b) Receiver

Figure 1: Examples of stochastic real-time BIP components

3.2 Composition of Stochastic Real-Time Components

Stochastic real-time components are composed using multiparty interactions. An interaction represents a strong synchronisation (i.e, *rendez-vous*) between transitions located in different components.

Given n stochastic real-time components $(B_i)_{i=1,n}$, with disjoint sets of ports P_i, we define interactions a as subsets of ports from $\cup_{i=1}^{n} P_i$, where

- $|a \cap P_i| \leq 1$, for every $i = 1, \ldots, n$, i.e., each component B_i participates in a by at most one port P_i,
- a contains either one stochastic port and any number of timed ports with *true* guards, or any number of timed ports with arbitrary timed guards.

An interaction is called stochastic if it contains a stochastic port, and timed otherwise. The timed ports participating on stochastic interactions are restricted to have precisely *true* guards. Intuitively, this ensures that the execution time for such interactions is solely determined by the stochastic port. While this restriction could be avoided at the price of slightly increasing the complexity of the forthcoming stochastic semantics, it has limited impact on the modelling capabilities – none of examples considered actually required other types of interaction beyond the two categories above.

Definition 3.2 *A stochastic real-time BIP system is defined as the composition $\gamma(B_1, ..., B_n)$ of n components $B_1, ..., B_n$ with a set of interactions γ.*

Example 3.3 *Consider the composition of the **Sender** and **Receive** components shown in Examples 3.1 and 3.2. The composition is operated through the interaction $\{send, recv\}$, which relates the* send *port of the **Sender** with the port* recv *of the **Receiver**. Figure 2 shows the two components and how they interact through the interaction {send, recv}. The nominal behaviour is when the **Sender** sends data to the **Receiver** through interaction* {send,recv}*. However, the former may fail when interaction $\{fail\}$ takes place, which is potentially detected by the **Receiver**. The latter emits an alarm and switches into a non-receiving mode by executing interaction $\{alarm\}$. The two components resume their normal activity after some delay, through interactions $\{recover\}$ and $\{back\}$ respectively.*

We further introduce some additional notations for defining the underlying operational semantics of a stochastic real-time BIP system. Let $\gamma(B_1, ..., B_n)$ be a stochastic real-time BIP system, where $B_{i=1,...,n} = (L_i, X_i, P_i, T_i, \ell_i^0)$.

Definition 3.3 *We define states s as couples $(\vec{\ell}, \vec{v})$, where $\vec{\ell} = (\ell_1, ..., \ell_n) \in L_1 \times ... \times L_n$ is a global location, and $\vec{v} : \cup_{i=1}^{n} X_i \to \mathbb{R}_{\geq 0}$ is a vector of clocks valuations.*

We define *d-succ*$((\vec{\ell}, \vec{v}), a)$ as the discrete successor (partial) function that computes the successor of a state $(\vec{\ell}, \vec{v})$ when taking an interaction a. Let $a = \{p_i\}_{i \in \mathcal{I}}$ such that $\mathcal{I}$ denotes the set of indices of the components participating in a. We define *d-succ*$((\vec{\ell}, \vec{v}), a) = (\vec{\ell'}, \vec{v'})$ whenever:

- for every $i \in \mathcal{I}$, there exists an enabled transition $\ell_i \xrightarrow{p_i \ g_i^{u_i} \ r_i} \ell_i'$ of B_i that is

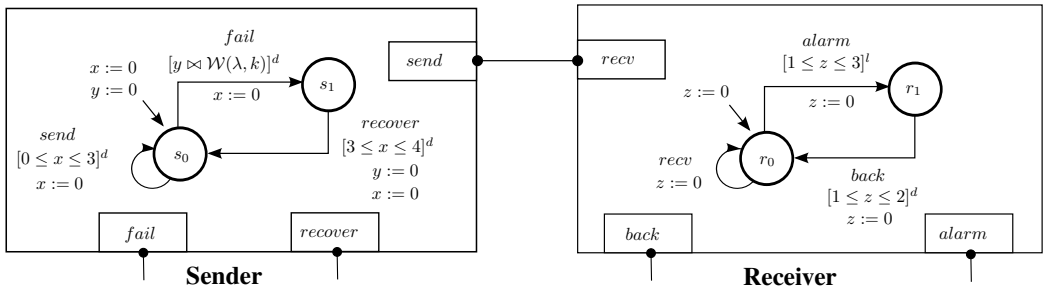


Figure 2: Composition of two stochastic real-time BIP components

  - either g_i is a timed constraint and $\vec{v}_{|X_i} \models g_i$, where $\vec{v}_{|X_i}$ is the projection of the set of clocks on the subset of clocks that are used in g_i
  - or g_i is a stochastic constraint $x \bowtie \rho$ and $\vec{v}(x) \in dom(\rho)$

- all these transitions are simultaneously executed, that is, clocks are reset $\vec{v'}(x) = 0$ for all $x \in \cup_{i \in \mathcal{I}} r_i$ and stay unchanged $\vec{v'}(x) = \vec{v}(x)$ otherwise.

- all the components that do not participate in a remain unchanged, that is, for every $j \notin \mathcal{I}$ it holds $\ell_j = \ell'_j$.

If no successor by interaction a exists at state $(\vec{\ell}, \vec{v})$, we define *d-succ*$((\vec{\ell}, \vec{v}), a) = \bot$.

We define *t-succ*$((\vec{\ell}, \vec{v}), t)$ as the time successor function that computes the successor of a state $(\vec{\ell}, \vec{v})$ for a time progress of t. It is a total function and is defined as *t-succ*$((\vec{\ell}, \vec{v}), t) = (\vec{\ell}, \vec{v} + t)$. That is, it increases all the clocks in $\vec{v}$ by the amount of time t.

Finally, we define the function *succ*$((\vec{\ell}, \vec{v}), t, a)$ that computes the successor of a state $(\vec{\ell}, \vec{v})$ when taking an interaction a after the time progress of t, which is a partial function defined as the composition *succ*$((\vec{\ell}, \vec{v}), t, a) =$ *d-succ*(*t-succ*$((\vec{\ell}, \vec{v}), t), a)$.

Definition 3.4 *The operational semantics of a stochastic real-time BIP system is defined as the timed transition system $\mathcal{T} = (S, s_0, \rightarrow_S)$ where*

- *S is the set of states, and s_0 is the initial state,*

- *$\rightarrow_S \subseteq S \times (\gamma \cup \mathbb{R}_{\geq 0}) \times S$ are transitions defined by the two rules*

$$\textsc{Discrete}\ \frac{\textit{d-succ}((\vec{\ell}, \vec{v}), a) = (\vec{\ell'}, \vec{v'})}{(\vec{\ell}, \vec{v}) \xrightarrow{a}_S (\vec{\ell'}, \vec{v'})}$$

$$\textsc{Time}\ \frac{t > 0, \qquad \forall a\ \textit{delayable}.\ (\exists t'.\ \textit{succ}((\vec{\ell}, \vec{v}), t', a) \neq \bot) \Rightarrow (\exists t'' \geq t.\ \textit{succ}((\vec{\ell}, \vec{v}), t'', a) \neq \bot)}{(\vec{\ell}, \vec{v}) \xrightarrow{t}_S (\vec{\ell}, \vec{v} + t)}$$

That is, according to the first rule, an enabled interaction can be fired at the current instant and the state updated. According to the second rule, time can progress as long as all enabled *delayable* interactions remain enabled. Note that a run of $\mathcal{T}$ is an infinite sequence $\sigma = s_0 s_1 s_2 \cdots$, such that $s_i \xrightarrow{t_i, a_i}_S s_{i+1}$, for some $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \gamma$, for all $i \geq 0$.

3.3 Stochastic Simulation Semantics

So far, we introduced the concepts of stochastic real-time BIP components and presented their composition from an operational viewpoint. In this section, we show how this model embraces a stochastic semantics in terms of a Generalised Semi Markov Process (16).

GSMPs are stochastic process descriptions for a large class of discrete-event systems. A configuration of the GSMP is usually determined by a state and a set of active (enabled) events, every one associated with a *remaining lifetime*, i.e. the amount of time during which it remains enabled. The choice of the event to be executed follows a *race policy*, which consists of selecting the event having the smallest remaining lifetime. The execution itself occurs when the remaining lifetime reaches 0 and triggers a state change and moreover, an update of the set of active events. That is, several events could be disabled and therefore removed from the set, or could be enabled, and therefore added to the set. In the latter case, the remaining lifetime is randomly chosen according to a (usually dense support) probability density function associated to the event.

The stochastic real-time BIP semantics follows the same intuition by considering interactions defined at composition as the GSMP events. Moreover, the associated probability density functions are obtained from the explicit density functions used in stochastic guards of stochastic interactions or by some default densities (uniform or exponential) in the case of timed interactions. In the remaining of this section we introduce the stochastic simulation algorithm and define precisely the different densities and sampling procedures.

3.3.1 Stochastic Simulation Algorithm

As for a GSMP, our simulation keeps track of the remaining lifetime of each interaction in order to implement the race policy. To this end, we define configurations as follows.

Definition 3.5 *We define a configuration z as a couple $\langle(\vec{\ell},\vec{v}),\vec{w}\rangle$, where $(\vec{\ell},\vec{v})$ is a state and $\vec{w}:\gamma\to\mathbb{R}_{\geq 0}$ is a vector of remaining lifetime of active interactions.*

For a given interaction a, the value $\vec{w}(a)$ represents the remaining lifetime at the current global location $\vec{\ell}$, if a is active.

Moreover, we need to identify dependencies between interactions. As explained for the GSMPs, the execution of an interaction might enable and/or disable other interactions. In the case of stochastic real-time BIP we consider that an interaction a has an impact on another interaction b, denoted by $a \triangleright b$, iff the guard of b changes due to the execution of a, that is, either because b has different timing constraints at the location(s) reached after executing a, or because a resets some clocks explicitly involved in one of the constraints of b (before or after executing a).

Algorithm 1 below presents the stochastic execution dynamics of a stochastic real-time BIP system. The algorithm shows how to move from one configuration $z_k=\langle(\vec{\ell}_k,\vec{v}_k),\vec{w}_k\rangle$ to another $z_{k+1}=\langle(\vec{\ell}_{k+1},\vec{v}_{k+1}),\vec{w}_{k+1}\rangle$, starting from an initial configuration $z_0=\langle(\vec{\ell}_0,\vec{v}_0),\vec{w}_0\rangle$. The first part of the algorithm computes this initial configuration as a vector $\vec{\ell}_0$ of the initial locations of the components B_i of the system, a vector of initial valuations of the clocks $\vec{v}_0$, and a vector of initial remaining lifetimes of interactions $\vec{w}_0$. In the latter, each interaction b which is not enabled at the initial state, i.e., $\forall t.succ((\vec{\ell}_0,\vec{0}),t,b)=\bot$, is assigned an infinite remaining lifetime, each enabled interaction b, is assigned a remaining lifetime through the sampling function $\mathcal{R}_b((\vec{\ell}_0,\vec{v}_0))$, which will be formally defined in the next sub-section.

The main loop of the algorithm is executed while there still interactions with a finite remaining lifetime in $\vec{w}_k$. Each iteration determines the next configuration z_{k+1} from the current one z_k. Given the current configuration, enabled interactions race to determine which one will be executed, i.e., the one with the minimum remaining lifetime in $\vec{w}_k$. Given the winning interaction a_k and its remaining lifetime t_k, we compute the successor state $(\vec{\ell}_{k+1},\vec{v}_{k+1})$ by using the *succ* function defined earlier. Finally, the remaining lifetimes of the interactions enabled in the new state are updated. Three cases can be distinguished for updating the vector of remaining lifetimes $\vec{w}_{k+1}$.

1. if interaction a_k has no impact on b, then the remaining lifetime of b at $(\vec{\ell}_{k+1},\vec{v}_{k+1})$ is its remaining lifetime at $(\vec{\ell}_k,\vec{v}_k)$ decreased by t, i.e., the amount of time progress,

2. if interaction a_k has an impact on b, and b is not active at $(\vec{\ell}_{k+1},\vec{v}_{k+1})$, then $\vec{w}_{k+1}(b)$ is set to ∞, that is, will not race in this new configuration,

input : $\gamma(B_1, \ldots, B_n)$, where $B_{i=1,\ldots,n} = (L_i, X_i, P_i, T_i, \ell_i^0)$
output: An execution trace

/* Compute the initial state $(\vec{\ell}_0, \vec{v}_0)$ */
$\vec{\ell}_0 := (\ell_1^0, \ldots, \ell_n^0)$ /* ℓ_i^0 is the initial location of B_i */
$\vec{v}_0 := \vec{0}$ /* $\vec{0}$ is the vector of initial clocks valuations */
/* Compute the initial remaining lifetime */
foreach *interaction* $b \in \gamma$ **do** $\vec{w}_0(b) := \begin{cases} \infty & \text{if } \forall t.\ succ((\vec{\ell}_0, \vec{0}), t, b) = \bot \\ \mathcal{R}_b((\vec{\ell}_0, \vec{v}_0)) & \text{if } \exists t.\ succ((\vec{\ell}_0, \vec{0}), t, b) \neq \bot \end{cases}$
/* Compute the initial configuration z_0 */
$z_0 := \langle(\vec{\ell}_0, \vec{v}_0), \vec{w}_0\rangle$
$k := 0$
/* Main loop: computes z_{k+1} from z_k */
while $\exists b \in \gamma.\ \vec{w_k}(b) \neq \infty$ **do**
  /* Race: determines the interaction a_k to execute */
  Let $t_k = min_{a \in \gamma} \vec{w}_k(a)$, and let a_k be the associated min event
  /* Update successor state */
  $(\vec{\ell}_{k+1}, \vec{v}_{k+1}) := succ((\vec{\ell}_k, \vec{v}_k), t_k, a_k)$
  /* Update remaining lifetime for interactions */
  foreach *interaction* $b \in \gamma$ **do**
  $\vec{w}_{k+1}(b) := \begin{cases} \vec{w}_k(b) - t & \text{if } \neg(a_k \triangleright b) \\ \infty & \text{if } a_k \triangleright b \text{ and } \forall t.\ succ((\vec{\ell}_{k+1}, \vec{v}_{k+1}), t, b) = \bot \\ \mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1})) & \text{if } a_k \triangleright b \text{ and } \exists t.\ succ((\vec{\ell}_{k+1}, \vec{v}_{k+1}), t, b) \neq \bot \end{cases}$
  /* Compute the next configuration z_{k+1} */
  $z_{k+1} := \langle(\vec{\ell}_{k+1}, \vec{v}_{k+1}), \vec{w}_{k+1}\rangle$
  $k := k + 1$
end

Algorithm 1: Stochastic Simulation Algorithm

3. if interaction a_k has an impact on b, and b is active at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$, then its remaining lifetime is sampled according to the function $\mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1}))$.

Note that the enumerated settings include the case where new interactions are appearing at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$. The reason is that any interaction b that is going to appear at this state can be classified as either was impacted by a_k or not.

It is worth explaining that the difference between interactions impacted by the executed interaction a_k and the non impacted ones, regarding the sampling operation is as follows. The former interactions involve ports of a shared component, thus by executing a_k the state changes (potentially, the location of the shared component changes, some clocks were reset, etc) and they need to be re-sampled as they are really seen as new interactions. For the latter interactions, from their point of view nothing has changed but time has evolved, so we do not need to re-schedule them (by re-sampling) but just to update their remaining lifetime accordingly.

3.3.2 The Sampling Procedure

The sampling function $\mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1}))$ used in Algorithm 1 computes the remaining lifetime for each interaction b enabled when entering the state $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$ by taking interaction a_k from the state $(\vec{\ell}_k, \vec{v}_k)$. It depends on the type of interaction b, that is timed or stochastic, and delayable or lazy. For the sake of simplicity, we define the sampling procedure in two steps: (1) in this subsection, we define the sampling procedure without detailing the underlying probability density function, (2) in the next subsection, we will define how the density function is actually computed.

First let us consider the partitioning of enabled interactions from a configuration $\langle(\vec{\ell}, \vec{v}), \vec{w}\rangle$ as either *fixed-delay* interactions, denoted $\mathcal{F}$ or *variable-delay* interactions, denoted $\mathcal{V}$. *Fixed-delay* interactions are induced by timed interactions having equality on their associated time constraints (also potentially by stochastic interactions following them), whereas *variable-delay* interactions are timed or stochastic interactions with an interval of possible remaining lifetime values.

$$\begin{aligned} \mathcal{F} &= \{a \in \gamma \mid \vec{w}(a) \neq \infty,\ \exists! t.\ \mathit{succ}((\vec{\ell}, \vec{v}), t, a) \neq \bot\} \\ \mathcal{V} &= \{a \in \gamma \mid \vec{w}(a) \neq \infty\} \setminus \mathcal{F} \end{aligned}$$

Based on this partitioning, the sampling function for an enabled interaction b when entering a new state $(\vec{\ell}, \vec{v})$ is as follows.

$$\mathcal{R}_b((\vec{\ell}, \vec{v})) = \begin{cases} t & & & \text{if } b \in \mathcal{F} \text{ is delayable and enabled at } t \\ \text{if } X \text{ then} & t & \text{else } \infty & \text{if } b \in \mathcal{F} \text{ is lazy and enabled at } t \\ & \mathrm{F}^{-1}_{\tilde{\rho}_b}(Y) & & \text{if } b \in \mathcal{V} \text{ is delayable} \\ \text{if } X \text{ then} & \mathrm{F}^{-1}_{\tilde{\rho}_b}(Y) & \text{else } \infty & \text{if } b \in \mathcal{V} \text{ is lazy} \end{cases}$$

where $X \sim \mathcal{B}(\frac{1}{2})$ is a random Bernoulli variable over $\{\textit{true}, \textit{false}\}$, i.e., *true* and *false* have a probability $\frac{1}{2}$, $Y \sim \mathcal{U}(0,1)$ is a random variable with standard uniform distribution, and $\mathrm{F}^{-1}_{\tilde{\rho}_b}$ is the inverse *cumulative distribution function* (CDF) of the probability density function $\tilde{\rho}_b$ associated to b at $(\vec{\ell}, \vec{v})$.

For *fixed-delay* interactions ($b \in \mathcal{F}$), if b is delayable, the sampling function $\mathcal{R}_b((\vec{\ell}, \vec{v}))$ returns the single time value t that satisfies the guard g_b. Whereas, if b is lazy, a discrete choice according to X is first performed to determine whether b will be considered and scheduled to t, or not considered and scheduled to ∞.

The sampling function in the case of *variable-delay* interactions ($b \in \mathcal{V}$) is slightly more involved since it requires choosing from an interval of time values. The same treatment with respect to the urgency types of interactions is performed i.e., a discrete choice on X is used to consider a lazy interaction or not. The time value is obtained by sampling according to the probability distribution $\tilde{\rho}_b$. Technically, this corresponds to computing the inverse CDF ($\mathrm{F}^{-1}_{\tilde{\rho}_b}$) on a random value Y uniformly distributed in the interval $[0, 1]$. The detailed definition of the probability density function $\tilde{\rho}_b$, in the case of timed and stochastic interactions, is given below.

3.3.3 Density Functions for Variable-delay Interactions

In this subsection, we define the density function $\tilde{\rho}_b$ associated with a *variable-delay* interaction b at a state $(\vec{\ell}, \vec{v})$. We recall that such an interaction may be either timed or stochastic. For the former case, since no density function is explicitly specified on guards, the function $\tilde{\rho}_b$ is obtained from a uniform or exponential density function. For the latter case, the function $\tilde{\rho}_b$ is obtained from the density function associated with the guard of b.

$$\tilde{\rho}_b(t) = \begin{cases} \frac{1}{u-l} \cdot \mathbf{1}[l \leq t \leq u] & \text{if } b \text{ is timed with guard } g_b \text{ true on } [l,u] \\ & \text{that is, } \vec{v}_b + t \models g_b \text{ iff } t \in [l,u] \\ \lambda e^{-\lambda(t-l)} \cdot \mathbf{1}[l \leq t] & \text{if } b \text{ is timed with guard } g_b \text{ true on } [l,\infty) \\ & \text{that is, } \vec{v}_b + t \models g_b \text{ iff } t \in [l,\infty) \\ \frac{\rho(\vec{v}_b(x)+t)}{\int_{\vec{v}_b(x)}^{\infty} \rho(s)ds} & \text{if } b \text{ is stochastic with guard } [x \bowtie \rho] \end{cases}$$

where $\mathbf{1}[t \in D]$ is the identity function, which gives 1 if $t \in D$, and 0 otherwise.

The first two cases correspond to timed interactions. We distinguish two situations in this setting, **(i)** when interaction b is timed and has a right-bounded guard, i.e., u is finite, the sampling in the interval $[l,u]$ is done uniformly, **(ii)** when the timed constraint is of the form $[l,\infty)$, the sampling is done according to the exponential density function. In both scenarios, the time t to sample must be withing the interval specified by the time constraint. Stated differently, the current valuations of clocks in $\vec{v}_b$ increased by the sampled time t must satisfy the guard g_b.

Remark that for **(i)** and **(ii)**, i.e., for timed interactions, the time constraint g_b may involve several clocks (potentially because of the composition, recall that an interaction involves several ports). Moreover, when entering a new state $(\vec{\ell}, \vec{v})$, the concerned clocks $\vec{v}_b$ may have valuations different from 0. Hence, the computation of the final time bounds u, l in which the time t will be sampled, for b, either uniformly or exponentially is more involved. Generally, given a guard g_b of the form $\bigwedge_i (l_i \leq x_i \leq u_i)$ and the valuations $\vec{v}_b(x_i)$, the bounds of the sampling interval of b are actually computed as $l = max(l_i - \vec{v}(x_i))$ and $u = min(u_i - \vec{v}_b(x_i))$ as illustrated in the next example.

Example 3.4 *The situation depicted in Figure 3 shows a global state of the system $(\vec{\ell}, \vec{v})$, where the valuations of clocks x and y are respectively $\vec{v}(x) = 1$ and $\vec{v}(y) = 2$, and the time constraint is $[(2 \leq x \leq 6) \wedge (2 \leq y \leq 5)]^d$. For the clock x, the remaining lifetime interval t_x is computed as $(2-1) = 1 \leq t_x \leq (6-1) = 5$. Similarly, for y, $(2-2) = 0 \leq t_y \leq (5-2) = 3$. Hence, the obtained sampling interval $[l,u]$ is $max(1,0) \leq t \leq min(5,3)$. Note that guards of the form $x - y \sim k$ have the same interpretation since the difference $x - y$ is constant over time as both clocks evolve identically.*

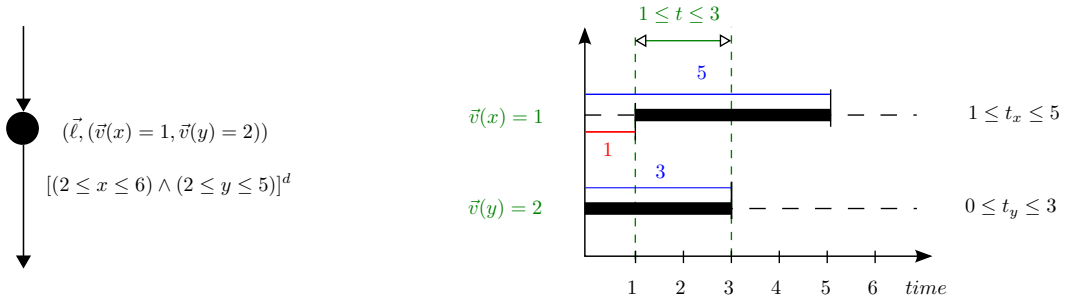


Figure 3: Computation of upper and lower bound in the case of timed interactions; $l = max(1,0) = 1$ and $u = min(5,3) = 3$, hence the sampling will be uniform in $[1,3]$.

The third case in the definition of $\tilde{\rho}_b(t)$ concerns variable-delay interactions obtained from a stochastic interaction b with a guard $[x \bowtie \rho]^d$. In this scenario, the sampling is done in $dom(\rho)$ according to a potentially shifted and normalised density function. This transformed function takes into account the case

where the clock valuation of x, i.e., $\vec{v}(x)$ is not 0 when entering the state $(\vec{\ell}, \vec{v})$. Below is a concrete illustration of the transformation.

Example 3.5 *The transformation is illustrated in Figure 4, where $\rho(t)$ is a Normal density function and $\vec{v}(x) = 1$. The function is first shifted to the current valuation of x, i.e., $\rho(1+t)$. Since this shifted function is no longer a proper probability density function, i.e., its area is lower than 1, it is normalised, i.e., divided by $\int_1^\infty \rho(s)\mathrm{d}s$.*

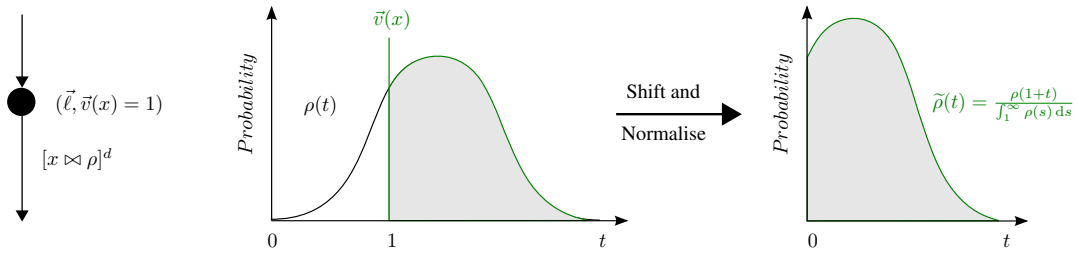


Figure 4: Shifting and normalising a Normal density function in the case of stochastic interactions

3.4 An Example of Stochastic Simulation

In Figure 5, we illustrate the stochastic semantics on Example 3.3 of the Sender-Receiver. We actually show a specific execution trace by sampling particular time values in each configuration. In this figure, configurations are of the form $\langle (s_i, r_j), (\vec{v}(x), \vec{v}(y), \vec{v}(z)), (\vec{w}(\{send, recv\}), \vec{w}(\{fail\}), \vec{w}\{recover\}, \vec{w}(\{alarm\}), \vec{w}(\{back\}))\rangle$. In each configuration, newly sampled remaining lifetimes are denoted by a box $\boxed{\text{t}}$, and updated remaining lifetimes are either ∞ or underlined $\underline{t}$ according to the definition of the sampling function $\mathcal{R}_b$. To make the example readable, we only show the discrete transition, i.e. induced by the uniform choice over lazy interactions.

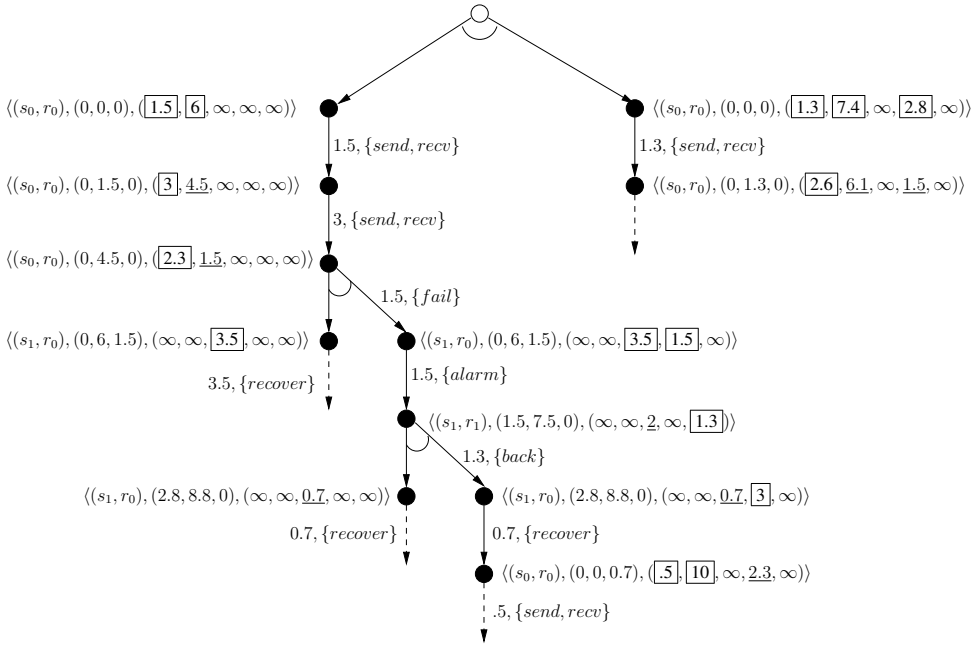


Figure 5: Illustration of the stochastic simulation semantics on Example 3.3

In this example, there are two possible initial configurations corresponding to the choice of considering the lazy interaction *alarm* $\langle (s_0, r_0), (0,0,0), (\boxed{1.3}, \boxed{7.4}, \infty, \boxed{2.8}, \infty)\rangle$ or not $\langle (s_0, r_0), (0,0,0), (\boxed{1.5}, \boxed{6}, \infty, \infty, \infty\rangle$

at the beginning. Both configurations have the same global location and clocks valuation $(s_0, r_0), (0, 0, 0)$, but differ in their sampling of the remaining lifetime of the initially racing interactions, namely $\{send, recv\}, \{fail\}$ and $\{alarm\}$. In one case (left branch), we have $(1.5, 6, \infty, \infty, \infty)$, i.e., $alarm$ is scheduled at ∞, while in the second case (right branch), $(1.3, 7.4, \infty, 2.8, \infty)$, i.e., $alarm$ is scheduled at 2.8. Note that the probability to start in one of these configurations corresponds to the probability to get the sampled remaining time values weighted by a half. For the sake of simplicity, we preferred to detail only one branch of the execution trace, i.e., the one on the left in Figure 5. The complete execution trace shown in the example consists of the sequence of transitions $\xrightarrow{1.5,\{send,recv\}}\xrightarrow{3,\{send,recv\}}\xrightarrow{1.5,\{fail\}}\xrightarrow{1.5,\{alarm\}}\xrightarrow{1.3,\{back\}}\xrightarrow{0.7,\{recover\}}\xrightarrow{0.5,\{send,recv\}}$, which corresponds to two send-receive operations, followed by a fail of the **Sender**, which is detected by the **Receiver** that emits an alarm and moves to a degraded mode then gets back to its normal working mode, followed by a recover of the **Sender**, and finally another send-receive operation.

3.5 Additional Modelling Features

It is worth mentioning that the proposed model can be extended to allow for handling the usual cost/reward structures and data variables. For the sake of simplicity, we refrain from providing formal details for these additional modelling features and briefly provide some intuitions.

A cost/reward structure in this model can be obtained in a straightforward manner by adding a data-structure in our simulation algorithm and associate it with states and interactions. Since in our model we know how long the system remains in each state (as we keep track of the remaining lifetimes of interactions), and which interactions are executed; we can, by specifying unit cost/reward for states and interactions, compute the global cost for each execution trace of the system. For instance, in the previous example, assume that we have this modelling feature and that we specified a cost of a fail to be 2, and the cost of remaining in a failure mode as 1 per time unit. The total cost of the fragment of the execution trace shown in Figure 5 will be 5.5. That is, 2 (the fail interaction cost) plus $[(1.5 + 1.3 + 0.7) \times 1]$ (the total time spent by the **Sender** component in a failure mode, i.e., from executing interaction *fail* to executing interaction *recover*.

4 Statistical Model Checking

In this section we briefly recall the statistical model checking technique. We start by an overview of the temporal logic used to specify systems properties and then we describe a set of well known SMC algorithms.

4.1 The PBLTL Temporal Logic

We first recap Bounded Linear-time Temporal Logic (BLTL) and then define its probabilistic extension. The BLTL formulas that can be defined from a set of atomic propositions $\mathcal{P}$ are the following.

- $true$, $false$, p, $\neg p$, for all $p \in \mathcal{P}$;
- $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$;
- $\mathsf{N}\phi_1$, $\phi_1 \mathsf{U}^t \phi_2$.

where N is the next operator, U^t is the bounded until operator, ϕ_1 and ϕ_2 are BLTL formulas, and t is a positive integer. We also consider the usual temporal operators, namely, the bounded eventually $\mathsf{F}^t\phi = true\mathsf{U}^t\phi$, and the bounded always $\mathsf{G}^t\phi = \neg(true\mathsf{U}^t(\neg\phi))$.

The semantics of a BLTL formula is defined with respect to an execution trace $\pi = s_0 s_1 \ldots$ in the usual way (**?**). Roughly speaking, an execution trace $\pi = s_0 s_1 \ldots$ satisfies $\mathsf{N}\phi_1$, which we denote $\pi \models \mathsf{N}\phi_1$, if state s_1 of π satisfies ϕ_1. The execution π satisfies $\phi_1 \mathsf{U}^t \phi_2$, which we denote $\pi \models \phi_1 \mathsf{U}^t \phi_2$, iff there exists a state s_i with $i \leq t$ that satisfies ϕ_2 and all the states in the prefix from s_0 to s_{i-1} satisfy ϕ_1.

In the SBIP framework, the properties specification language for stochastic systems is a probabilistic variant of BLTL denoted PBLTL. More precisely, it consists of a BLTL formula preceded by a probabilistic

operator $\mathbf{P}$. Using this language, it is possible to formulate two types of queries on a given stochastic system as follows.

1. Qualitative queries : $\mathbf{P}_{\geq\theta}[\phi]$, where $\theta \in [0,1]$ is a probability threshold and ϕ is a BLTL formula, also called path formula,

2. Quantitative queries : $\mathbf{P}_{=?}[\phi]$, where ϕ is a BLTL formula, also called path formula.

Note that it is possible through these queries to either determine if the probability for the system to satisfies ϕ is greater or equal to the threshold θ (using 1), or to ask for the actual probability for the system to satisfy that property ϕ (using 2). For instance, the PBLTL formula $\mathbf{P}_{=?}[\mathsf{G}^{1000}(p)]$ stands for "What is the probability that the atomic proposition p is always satisfied?". In this example, the path formula $\mathsf{G}^{1000}(p)$ specifies that the length of the considered traces is 1000.

4.2 The Main SMC Algorithms

We now present a model checking procedure to decide whether a given stochastic system B satisfies a property ϕ. Statistical model checking refers to a series of simulation-based techniques that can be used to answer two questions: (1) **Qualitative:** is the probability for B to satisfy ϕ greater or equal to a certain threshold θ? and (2) **Quantitative:** what is the probability for B to satisfy ϕ? Both questions can serve to decide a PBLTL property.

The main approaches (28; 12) proposed to answer the qualitative question are based on *hypothesis testing*. Let p be the probability of $B \models \phi$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds is less or equal to α (respectively, β). Since it is impossible to ensure a low probability for both types of errors simultaneously (see (28) for details), a solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$.

Several hypothesis testing algorithms exist in the literature. Younes(28) proposed a logarithmic based algorithm that given p_0, p_1, α and β implements the *Sequential Ratio Testing Procedure* (*SPRT*) (see (27) for details). When one has to test $\theta \geq 1$ or $\theta \geq 0$, it is however better to use *Single Sampling Plan* (*SSP*) (see (28; 8; 12) for details) that is another algorithm whose number of simulations is pre-computed in advance. In general, this number is higher than the one needed by *SPRT*, but is known to be optimal for the above mentioned values. More details about hypothesis testing algorithms and a comparison between *SSP* and *SPRT* can be found in (8).

In (12) Peyronnet et al. propose an estimation procedure (*PESTIMATION*) to compute the probability p for B to satisfy ϕ. Given a *precision* δ, Peyronnet's procedure computes a value for p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* (13).

The efficiency of the above algorithms is characterised by the number of simulations needed to obtain an answer. This number may change from system to system and can only be estimated (28). However, some generalities are known. For the qualitative case, it is known that, except for some situations, *SPRT* is always faster than *SSP*. *PESTIMATION* can also be used to solve the qualitative problem, but it is always slower than *SSP* (28). If θ is unknown, then a good strategy is to estimate it using *PESTIMATION* with a low confidence and then validate the result with *SPRT* and a strong confidence.

It is worth mentioning that the statistical model checking technique is known to work for purely stochastic system model, i.e., non-determinism free. The stochastic real-time BIP modelling formalism guarantees that all non-determinism is resolved through stochastic choices over interactions. SMC can be extended to handle non-deterministic models, e.g., MDPs, in which case it provides an interval of probabilities of satisfaction a given property, by exploring all the possible schedules of the non-deterministic model.

5 The BIP$^{\mathcal{SMC}}$ Engine

5.1 Architecture

The BIP$^{\mathcal{SMC}}$ engine implements several statistical testing algorithms for stochastic systems verification, namely, Single Sampling Plan (*SSP*), Simple Probability Ratio Test (*SPRT*) (27; 28), and Probability Estimation (*PESTIMATION*) (12). Figure 6 shows the most important modules of the tool and how they interact together in order to perform statistical model checking. The tool takes as inputs a stochastic model description in the stochastic BIP real-time format, a PBLTL property to check, and a set of confidence parameters required by the statistical test.

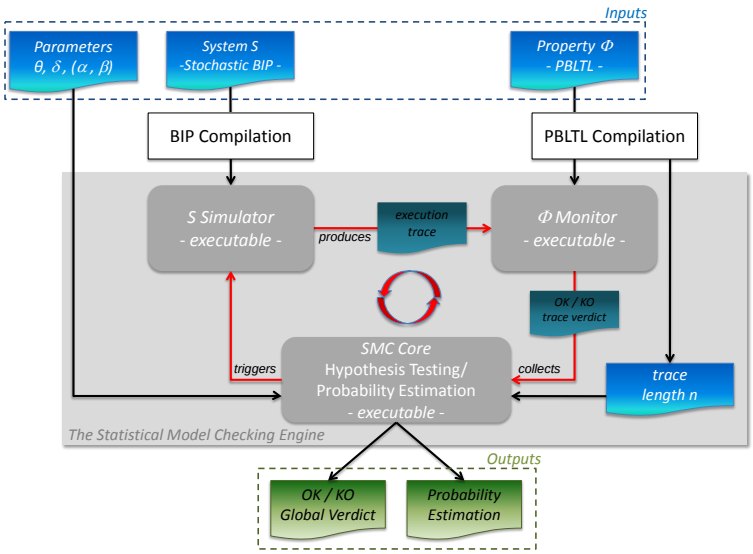


Figure 6: The BIP$^{\mathcal{SMC}}$ Architecture

During the initial phase, the tool performs a syntactic validation of the PBLTL formula through a parser module. Then, it builds an executable model and a monitor for the property under verification. Next, it will iteratively trigger the stochastic BIP engine to generate execution traces which are monitored to produce local verdicts. This procedure is repeated until a global decision can be taken by the SMC core module (that implements the statistical algorithms). As our approach relies on SMC and since it considers bounded LTL properties, we are guaranteed that the procedure will eventually terminate.

It is worth mentioning that in our implementation, atomic propositions of PBLTL properties are constructed from the system variables. For instance, the PBLTL formula $\mathbf{P}_{=?}[\mathsf{G}^{1000}(abs(Master.tm - Slave.ts) \leq 160)]$ stands for "What is the probability that the difference between master variable tm and slave variable ts is always under the bound 160 ?". In this example, $Master.tm$ and $Slave.ts$ are systems variables pertaining to components $Master$ and $Slave$ respectively. Note that properties specification language offers the possibility to use built-in mathematical functions. In the example above, the $abs()$ function is used to compute the absolute value of $(Master.tm - Slave.ts)$.

5.2 Technical details and Availability

BIP$^{\mathcal{SMC}}$ is fully developed in the Java programming language. It uses JEP 2.4.1 library (http://www.singularsys.com/jep/index.html, under GPL license) for parsing and evaluating mathematical expressions, and ANTLR 3.2 (http://www.antlr.org/) for PBLTL properties parsing and monitoring. At this stage, BIP$^{\mathcal{SMC}}$ only runs on GNU/Linux operating systems as it relies on the BIP simulation engine. The current release of the tool has been enriched with a graphical user interface for more convenience (see Figure 7 for a screen shot). The current version also includes supports of the

BIP2 language (http://www-verimag.imag.fr/New-BIP-tools.html) while still ensuring compatibility with the previous version. The model checker is available for download from http://www-verimag.imag.fr/Statistical-Model-Checking.html, where additional information (video tutorial) on how to install it and to use it can also be found.

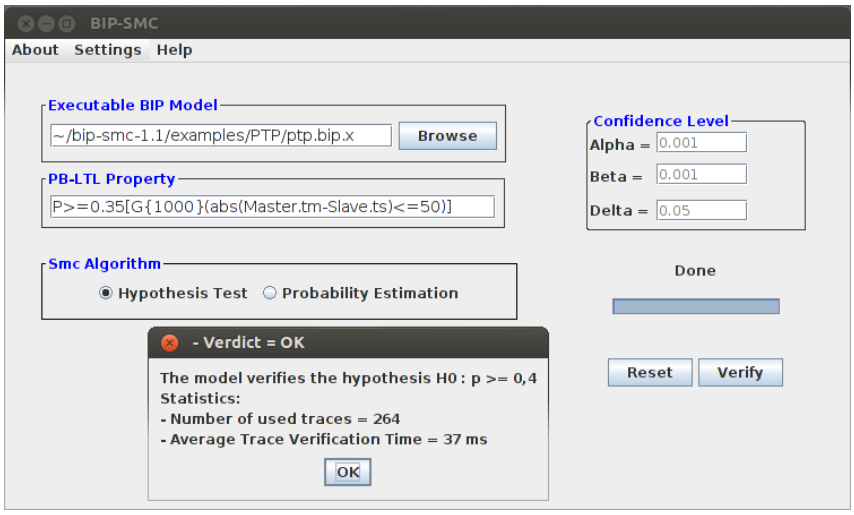


Figure 7: Screen shot of the BIP^{SMC} graphical user interface.

6 Case Studies

While still at the prototype level, the SBIP framework has been used to evaluate several large-scale systems that cover different application domains. In this section we survey some of these studies and discuss their results. The first two studies consider the modelling and assessment of multimedia applications, while the three remaining present networking application and protocols. For the sake of conciseness, we show for some of them how the underlying models are built in the proposed stochastic formalism.

6.1 An MPEG2 Decoder Subsystem

In this study, the SBIP framework is used to check QoS properties of an MPEG2 decoder subsystem part of a video streaming application (4). This work is about finding a good trade-off, when designing the multimedia system, between the required sizes of the system buffers and the quality of delivered videos. It is known in the literature that an acceptable amount of quality degradation – defined as less than two consecutive frames within one second – can be tolerated in order to reduce buffers sizes (4). The study will consist to assess this requirement.

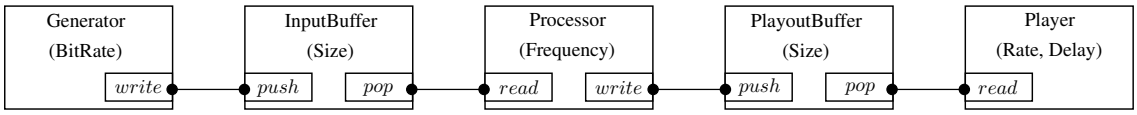


Figure 8: The abstract MPEG2 decoder model.

The model in Figure 8 is used to represent the considered MPEG2 subsystem. It shows its different parts, namely, the Generator, the Input and the Playout buffers, the Processor, which decodes the input videos macro-blocks, and the Player device. In this study, quality degradation is seen as a buffer underflow, which occurs whenever the Player device fails to read sufficient macro-blocks from the Playout buffer. Note that the amount of underflow is impacted by the parameter Delay of the Player, that represents the delay after which it starts playing the decoded video frames.

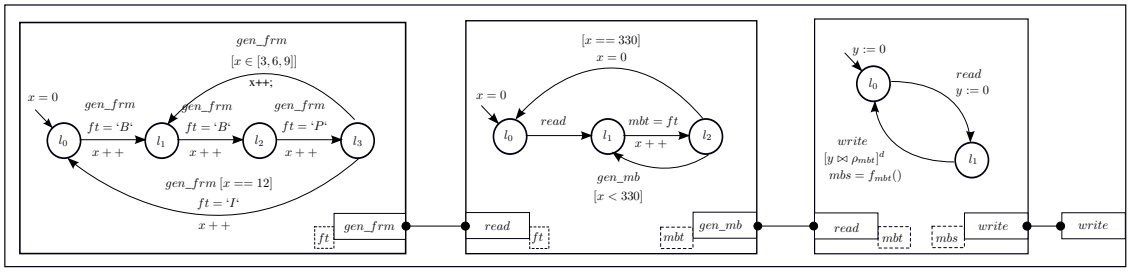


Figure 9: The Generator compound component. The probability density function ρ_{mbt} captures the arrival delays of videos macro-blocks.

Figure 9 shows the detailed behaviour of the Generator component. The latter models the arrival of encoded videos macro-blocks to the MPEG2 decoder subsystem in a stochastic fashion. This component is made of three sub-components operating as follows. The first component (left) generates frames following the MPEG2 GOP pattern (4). Each frame is then decomposed, in the second component, into 330 macro-blocks, which are finally transmitted to the third component that models the macro-blocks stochastic arrival time to the input buffer with respect to ρ_{mtb}.

Some of the obtained results in this study, using the SMC technique, are illustrated in Figure 10. The latter shows three different curves, each corresponding to a different analysed video, namely `cact.m2v`, `mobile.m2v`, and `tennis.m2v`, having the same resolution of 352×240. Each curve shows the evolution of the probability that the quality degradation is always less than two consecutive frames within one second, for different values of the Delay parameter. This result helps to determine the right Delay parameter to use. Note that for these experiments, the BIP$^{\mathcal{SMC}}$ engine required about 44 to 7145 traces each time and spent around 6 to 8 seconds in average to check the property with a confidence bound of 10^{-2}.

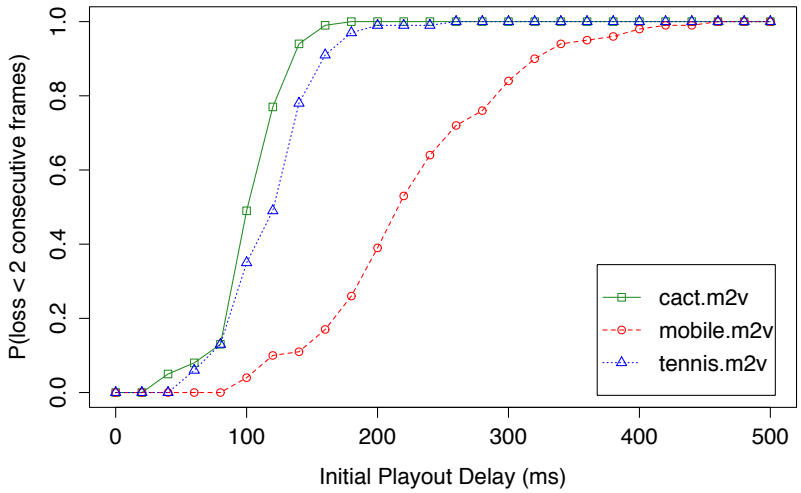


Figure 10: Probability of avoiding quality degradation in three different videos for increasing values of the Initial Playout Delay.

6.2 Image Recognition on Many-cores

In this case study, the SBIP framework is used as part of the design of an embedded system consisting of the HMAX image recognition application deployed on the STHORM many-core architecture (24; 25).

The HMAX model algorithm (22) is a hierarchical computational model of object recognition – in input images – which attempts to mimic the rapid object recognition of human brain. The case study focuses on

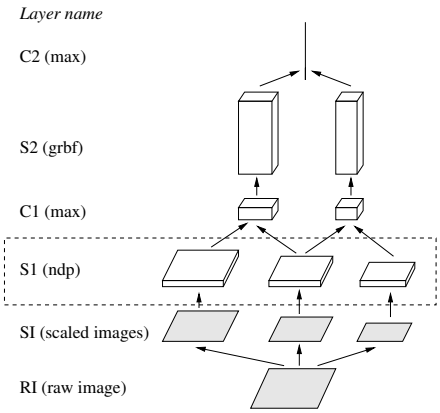


Figure 11: HMAX overview.

the first layer of the HMAX model algorithm, denoted S1 in Figure 11, as it is the most computationally intensive.

The goal of the study is to explore several design parameters with respect to timing constraints, that is, the overall execution time, and the time to process single lines of an input image. More precisely, the analysis consists of probabilistically quantifying the requirement that the overall execution time is always lower than a given bound, denoted Δ, and that the variability in the processing time of successive lines is always bounded by Ψ. To this extent, the above requirements were respectively specified in BLTL as $\phi_1 = \mathsf{G}^l(t_o < \Delta)$, where t_o is the monitored overall execution time, and $\phi_2 = \mathsf{G}^l(|t_s| < \Psi)$, where t_s is the difference between the processing time of successive lines.

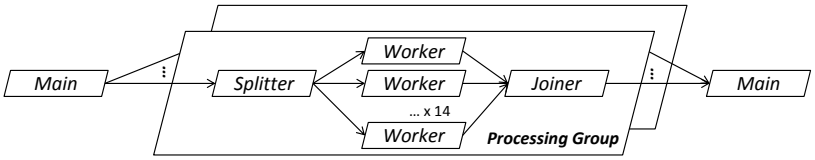


Figure 12: The abstract BIP model of the HMAX application S1 layer.

The parametric stochastic BIP model of the S1 layer of the HMAX model algorithm is shown in Figure 12. In this model, every image is handled by one *processing group* consisting of a single *Splitter*, one or more *Worker* processes, and a single *Joiner*, communicating through FIFO channels. The computation of the entire S1 layer is coordinated by a single *Main* process. In this model, several image scales (obtained by scaling at different sizes the input image as required by the HMAX algorithm) are handled concurrently by different processing groups, and the processing is pipelined using a pipelining rate denoted PR. Figure 13 shows the detailed BIP model of the *Worker* component using the new proposed stochastic semantics.

The aforementioned performance requirements, i.e. ϕ_1 and ϕ_2 were checked for different pipelining rates $PR = \{0, 2\}$ (experiments have shown that different greater values of P do not impact the time of interest.) and different values of the bounds Δ, Ψ. In this experiment, the sizes for the FIFO channels *Main-Splitter*$=$ 10 KB, *Splitter-Worker*$=$ 112 B, *Worker-Joiner*$=$ 336 B, and *Joiner-Main*$=$ 30 KB (see Figure 12) were chosen arbitrarily to fit the STHORM L1 memory of a single cluster.

Table 1 shows the probabilities of satisfying the first requirement ϕ_1 for different values of Δ, in the case where $PR = 0$. The table also reports, in the third raw, a performance metric, that is, the number of traces that were necessary for the *Hypothesis Testing* SMC algorithm to decide each time. For instance, based on these results, one can conclude that the expected overall execution time for processing one image scale is bounded by $\Delta = 572.91ms$ with probability equal to 0.99.

Figure 14 shows the probabilities to satisfy the second requirement ϕ_2 when varying the time bound

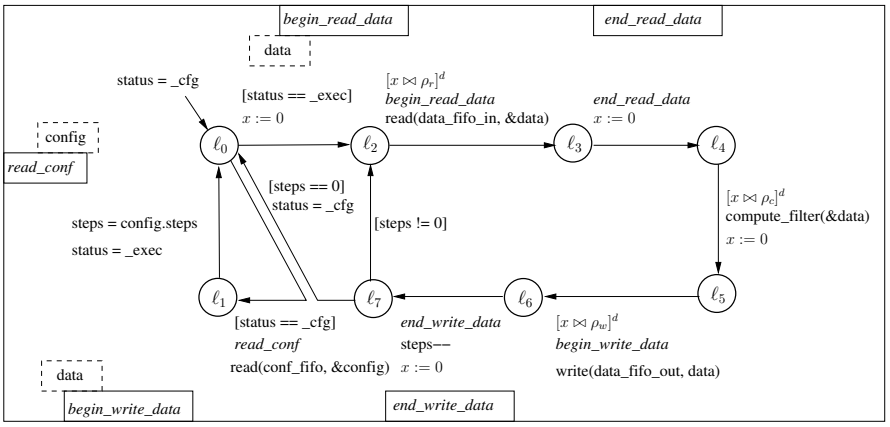
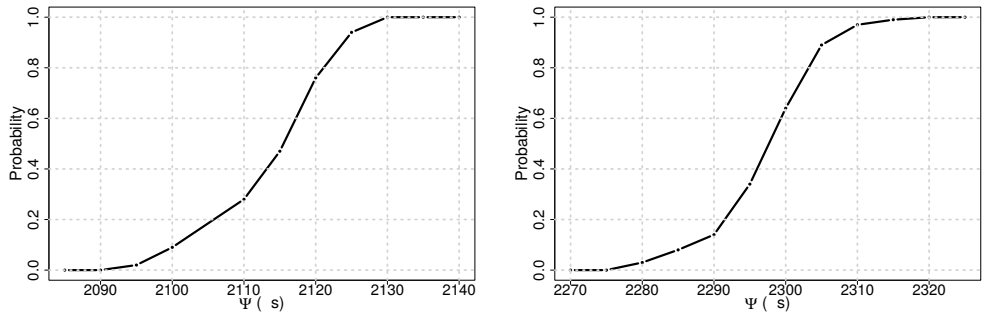


Figure 13: The stochastic BIP model of the *Worker* component, ρ_r, ρ_c, ρ_w are respectively the probability density functions of the time to read, to compute, and to write data.

Table 1: Probabilities to satisfy ϕ_1 for different bounds Δ and a fixed pipelining rate $PR = 0$.

$\Delta(ms)$	572.75	572.8	572.83	572.85	572.89	572.91	572.95
$P(\phi_1)$	0	0.28	0.57	0.75	0.98	0.99	1
Nb. of traces	66	1513	1110	488	171	89	66



(a) $PR = 0$ (b) $PR = 2$

Figure 14: Probability to satisfy ϕ_2 for $PR \in \{0, 2\}$.

Ψ for two values of the pipelining rate PR. Figure 14a is obtained with no pipelining, i.e. $(PR = 0)$, whereas Figure 14b is obtained with $PR = 2$. One can note that the two curves show similar evolution, albeit the curve in Figure 14b is slightly shifted to the right, i.e. the values of Ψ in this case are greater than those of Figure 14a. This actually means that this configuration induces more processing time variation between successive lines. We recall that when $PR = 0$, all the processes are perfectly synchronised which yields small variation over successive lines processing time. Using $PR > 0$ however leads to greater variation since it somehow alters this synchronisation. Concretely, Figure 14 shows that without pipelining, we obtain smaller expected time variation (of processing successive lines). For instance when $PR = 0$, $\Psi = 2128\mu s$ with probability 0.99, whereas for $PR = 2$, $\Psi = 2315\mu s$ with the same probability. One may conclude that, in this case study, a pipeline implementation will not help enhancing the system throughput.

6.3 The Precision Time Protocol – IEEE 1588

In this study, the Precision Time Protocol (PTP) is deployed as part of a distributed heterogeneous communication system in an aircraft (5). The protocol is used to synchronise the clocks of the different devices of the system. The reference clock is given by a specific device on the network, designated as the Master. This synchronisation is essential to guarantee a correct behaviour of the whole system.

The SBIP framework was used to check the accuracy of clocks synchronisation, which is defined as the absolute value of the difference between the reference clock θ_m of the Master and the clock θ_s of a Slave device. Figures 15 and 16 show, respectively, the architecture of the stochastic BIP model used to check this requirement, the stochastic behaviour Channel A and the deterministic behaviour of the Slave components.

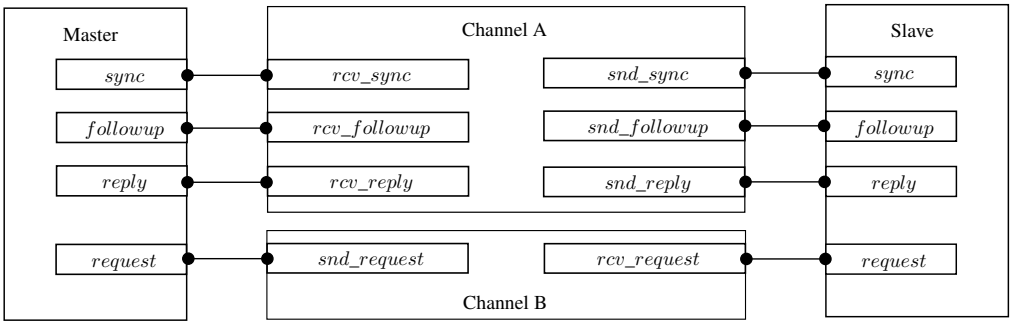
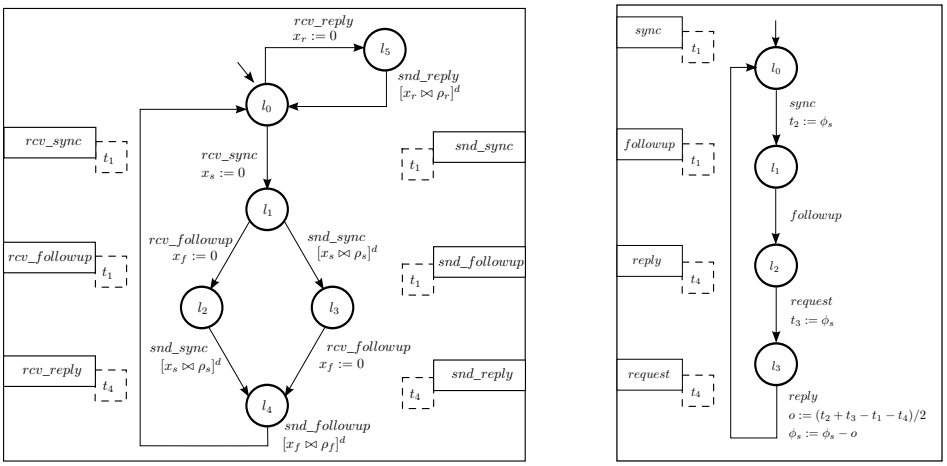


Figure 15: The abstract PTP model; composed of a Master and a single Slave device.



(a) Channel A (b) Slave

Figure 16: A detailed view of the Channel A and the Slave Components

For this model, we want that the deviation between the two clocks remains always under some given bound Δ. Since the model is stochastic, we want to estimate the probability for this to be true, for different values of Δ, and for each Slave device in the system (with respect to its position in the network, captured through the density functions ρ_r, ρ_s, and ρ_f). The goal is to find the value of Δ that gives the highest probability.

The synchronisation requirement is expressed in PBLTL as follows, $\mathbf{P}_{=?}[\mathsf{G}^{1000}(abs(\theta_m - \theta_s) \leqslant \Delta)]$. The ultimate goal of the study is to compute the minimal bound Δ that ensures full synchronisation, i.e, the synchronisation of all the Slaves clocks in the network with the Master clock, with probability 1.

The results illustrated in Figure 17 show the probability evolution of the devices synchronisation (in the y-axis) with respect to various time bounds in micro seconds (in the x-axis). We can see different

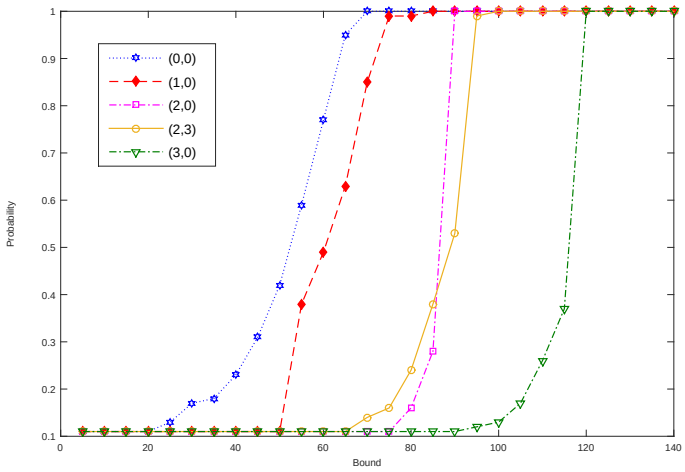


Figure 17: PTP accuracy analysis results

curves corresponding to several devices identified through their addresses in the network. We recall that the synchronisation is guaranteed for a specific device whenever its curve reaches probability 1. Thus, we can conclude from these experiments that the minimal bound Δ that ensures full synchronisation is 120μs.

6.4 Wireless Sensor Network

The SBIP framework has been used to verify several networked systems based on different technologies, CAN-based (20), Sensor Network using Wi-Fi (19), and IoT applications (21). We briefly present its utilisation for the modelling and analysis of a Wireless Sensor Network (WSN) case study.

This case study concerns an audio streaming application over a Wi-Fi network, where several nodes equipped with microphones produce different audio streams, which are transmitted to a base station equipped with a speaker to play the received audio. The goal of the study is to ensure the synchronisation between the different nodes of the network in order to guarantee a consistent audio output. To this extent, a Phase Locked Loop (PLL) synchronisation protocol is deployed as part of the application nodes. The protocol works as follows. The base station broadcasts periodically a frame containing the hardware clock value to all the nodes through the network. Each node applies the PLL synchronisation mechanism, to construct a software clock, that serves to keep its local clock synchronised with the received one.

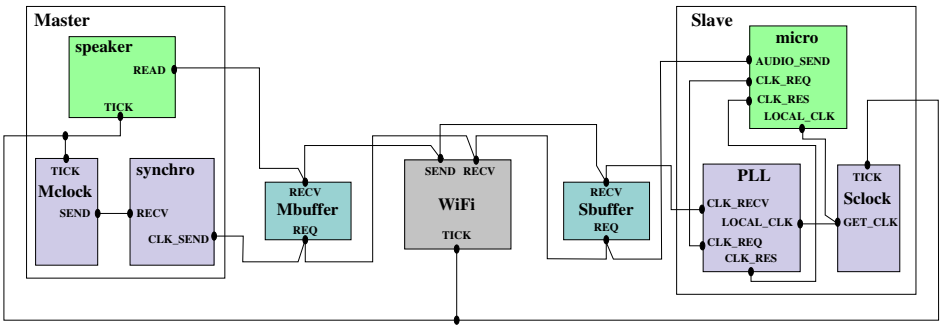


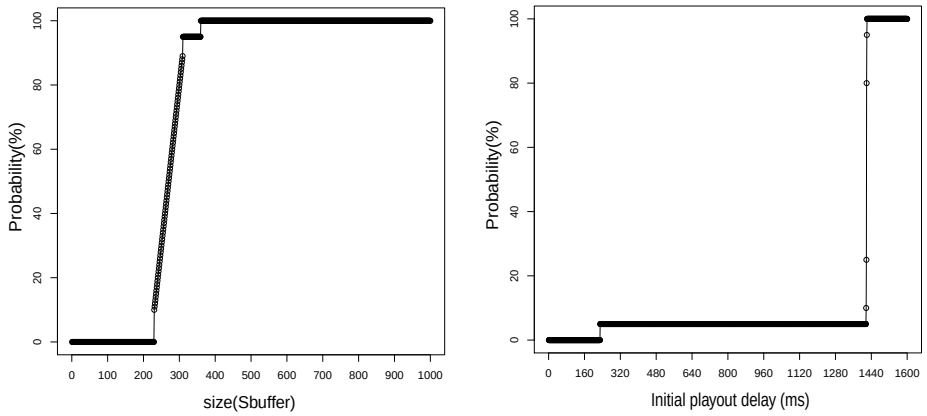
Figure 18: SBIP model of the Wireless Sensor Network

A BIP model of this application, following a Master-Slave architecture, was built. As shown in Figure 18, it consists of a Master component that represents the base station, a Slave component that represents a particular node in the network, a Wi-Fi component modelling the Wi-Fi communication channel, and two buffer components, namely Mbuffer and Sbuffer. In this model, the period of broadcasting the hardware

clock is fixed to $T = 5s$. Each node uses its local clock to timestamp the produced audio frames, so that the base station is able to reproduce the received audio frames in the correct order. The synchronisation accuracy is defined as the difference between the hardware clock and the computed software clock in each node and is required to be lower or equal to $1\mu s$.

The WSN application implementation was generated from the functional model shown in Figure 18 and deployed over three UDOO (http://www.udoo.org/features/) nodes, each consisting of a computational core, a Wi-Fi card, and a sound card. The wireless network is supported by the Snowball SDK (http://www.calao-systems.com/articles.php?pg=6186), which is used as an Access Point (AP). This implementation is used to learn the probability distributions that characterise the communication delays in order to build the stochastic BIP model used later for analysis.

Two sets of experiments were conducted, focusing on equally important requirements for the design of multimedia sensor networks. The first concerns the utilisation of the buffer components regarding the audio streaming capturing and reproduction in the system. The second focuses on the clock synchronisation accuracy. For the second requirement, the difference between the Master clock θ_m and the software clock, computed in every Slave θ_s, without the impact of the audio capturing and reproduction, is observed. Both requirements were described as probabilistic temporal properties, using PBLTL. The obtained results are presented hereafter.



(a) $P(\phi_1)$ when increasing the Sbuffer size

(b) $P(\phi_2)$ as function of the playout delay

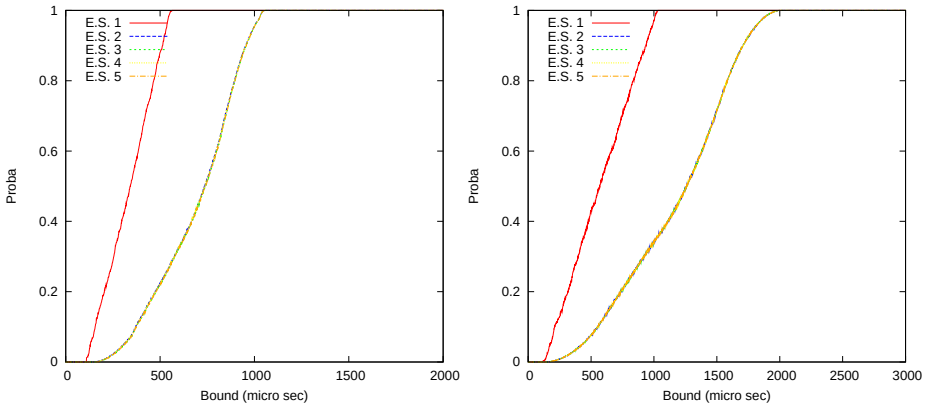
Figure 19: Probability results for properties ϕ_1 and ϕ_2

We evaluated the property of avoiding buffers overflow (respectively underflow) by considering the following property $\phi_1 = \mathsf{G}^l(S_{Sbuffer} < MAX)$ (respectively $\phi_2 = \mathsf{G}^l(S_{Mbuffer} > 0)$), where $S_{Sbuffer}$ (respectively $S_{Mbuffer}$) indicates the size of the Slave (respectively the Master) buffer, and MAX is a positive integer value which represents the capacity of the buffer. The left curve in Figure 19 shows the probability of avoiding an overflow in the Sbuffer, i.e, $P(\phi_1)$, for different values of MAX. One can conclude that a value of $MAX = 400$ ensures that $P(\phi_1) = 1$. It was observed during the experiments that the probability of underflow in the Mbuffer, i.e. $P(\phi_2)$, depends on the initial playout delay, that is, the delay after which the Master starts consuming from the Mbuffer. Figure 19 shows this probability evolution when increasing this delay. One can observe that $P(\phi_2) = 1$ is obtained for delays greater than 1430 ms.

The property of the synchronisation accuracy between the different nodes of the network and the Master was formalised as $\phi_3 = \mathsf{G}^l(|(\theta_m - \theta_s) - A| < \Delta)$, where A indicates a fixed offset between the Master and each computed software clock, and Δ is a fixed non-negative number denoting a specific time bound. The goal was to check the requirement that synchronisation accuracy $\Delta \leq 1\mu s$. For a fixed offset $A = 100\mu s$, the observed bound Δ was always greater than the expected $1\mu s$. In order to find a new bound Δ for the synchronisation accuracy with an offset $A = 100\mu s$, the SBIP framework was used to estimate the probability of ϕ_3. The goal is to find the smallest bound Δ that satisfies $P(\phi_3) = 1$. Different values of Δ between $10\mu s$ and 80 μs were explored. The obtained result was that, for the considered setting, the smallest bound that ensures the synchronisation is $\Delta = 76\mu s$.

6.5 Avionics Full-DupleX Switched Ethernet

SBIP has been also used for the analysis of QoS properties of the Avionics Full-DupleX switched Ethernet (AFDX) (6). The AFDX protocol was proposed as a solution to resolve problems due to the spectacular increase of the quantity of communication and thus of the number of wires in avionics network. The main idea behind AFDX is to simulate point-to-point connections between all the devices in a network using *Virtual Links* (VL). For such systems, one challenge is to guarantee bounded delivery times on every VL.



(a) Probability of having a delay lower than some bound for the configuration ($X = 10$).

(b) Probability of having a delay lower than some bound for the configuration ($X = 20$).

Figure 20: Results of latency analysis for the AFDX case study.

In order to check the latency requirements, two configurations having the same characteristics but with different numbers of virtual links ($X = 10$ and $X = 20$) were considered. This experiment consisted of using the *Probability Estimation* SMC algorithm with precision 0.01 and a confidence of 0.01 to estimate probabilities for bounds in $[0\mu s - 2000]\mu s$ for $X = 10$ and in $[0 - 3000]\mu s$ for $X = 20$. The obtained results are shown in Figures 20a and 20b for respectively $X = 10$ and $X = 20$ links. The figures show that for both configurations, the delivery time is bounded for all the considered End System (E.S), with different bounds depending on the E.S position in the network. In the configuration with 20 VLs the delivery time is more important because of the number of VLs.

For this study, to get more confidence, we also used the *Hypothesis Testing* SMC algorithms with a confidence of 10^{-10} and a precision of 10^{-7}. The obtained results consolidate the previous ones.

7 Conclusion

In this paper, we presented the SBIP framework that offers a stochastic real-time modelling formalism that conciliate the RT-BIP and the stochastic BIP models, in addition to a statistical model checking engine, called BIP$^{\mathcal{SMC}}$, for the quantitative assessment of the built systems models.

The stochastic real-time BIP model enables to build stochastic timed automata and to compose through multi-party interactions. It offers a mean to express stochastic timing constraints over systems interactions by attaching probability density functions to the guards of ports composing them, and to specify different urgency levels for them. As stated in the paper, this new model enables to handle dense time, as opposed to the current stochastic semantics. It would be thus interesting to consider a more expressive temporal logic than PBLTL, in the SMC engine, in order to express more relevant timing requirements for verification.

As shown along Section 6 of the paper, the SBIP framework has been used to model and to analyse several case studies. Nevertheless, several amelioration are still ahead. Especially, to enhance the performance of the BIP$^{\mathcal{SMC}}$ engine compared to more mature tools like Prism (18) or UPPAAL-smc (11). A major foreseen amelioration is at the level of the interface with the simulation engine.

References

[1] Abdellatif, T., Combaz, J., Sifakis, J.: Rigorous implementation of real-time systems - from theory to application. Mathematical Structures in Computer Science 23(4), 882–914 (2013), http://dx.doi.org/10.1017/S096012951200028X 3

[2] et al., A.N.: Performance evaluation of complex systems using the sbip framework. In: Proceedings of the 10th Workshop on Verification and Evaluation of Computer and Communication System, VECoS 2016, Tunis, Tunisia, October 6-7, 2016. pp. 11–26 (2016) 1

[3] Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (apr 1994) 1, 3

[4] Balaji, R., Nouri, A., Gangadharan, D., Bozga, M., Ananda Basu, M.M., Legay, A., Bensalem, S., Chakraborty, S.: Stochastic modeling and performance analysis of multimedia socs. In: International conference on Systems, Architectures, Modeling and Simulation, SAMOS'13. pp. 145–154 (2013) 1, 6.1, 6.1

[5] Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Forum for fundamental research on theory, FORTE'10. LNCS, vol. 6117, pp. 32–46. Springer (2010) 1, 6.3

[6] Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A., Siffakis, E.: Verification of an AFDX infrastructure using simulations and probabilities. In: Runtime Verification, RV'10. LNCS, vol. 6418. Springer (2010) 1, 6.5

[7] Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods. pp. 3–12. SEFM'06, IEEE Computer Society, Washington, DC, USA (2006) 1, 3

[8] Bensalem, S., Delahaye, B., Legay, A.: Statistical model checking: Present and future. In: RV. LNCS, vol. 6418. Springer (2010) 4.2

[9] Brázdil, T., Krčál, J., Křetínský, J., Řehák, V.: Fixed-Delay Events in Generalized Semi-Markov Processes Revisited, pp. 140–155. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-23217-6_10 2

[10] David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for biological systems. Int. J. Softw. Tools Technol. Transf. (STTT) 17(3), 351–367 (jun 2015) 1

[11] David, A., Larsen, K.G., Legay, A., Mikuăionis, M., Poulsen, D.B.: Uppaal smc tutorial. Int. J. Softw. Tools Technol. Transf. (STTT) 17(4), 397–415 (August 2015) 2, 7

[12] Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate Probabilistic Model Checking. In: International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04. pp. 73–84 (January 2004) 1, 4.2, 5.1

[13] Hoeffding, W.: Probability inequalities. Journal of the American Statistical Association 58, 13–30 (1963) 4.2

[14] Jegourel, C., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Importance Sampling for Stochastic Timed Automata, pp. 163–178. Springer International Publishing, Cham (2016) 2

[15] Jegourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking — plasma. In: Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 498–503. TACAS'12, Springer-Verlag, Berlin, Heidelberg (2012) 2

[16] Kulkarni, V.G.: Introduction to Modeling and Analysis of Stochastic Systems. Springer New York (2011) 1, 3.3

[17] Kumar, N., Sen, K., Meseguer, J., Agha, G.: A rewriting based model for probabilistic distributed object systems. In: Najm, E., Nestmann, U., Stevens, P. (eds.) FMOODS. pp. 32–46 (2003) 2

[18] Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: verification of probabilistic real-time systems. In: Proceedings of the 23rd international conference on Computer aided verification. pp. 585–591. CAV'11, Springer-Verlag, Berlin, Heidelberg (2011) 2, 7

[19] Lekidis, A., Bourgos, P., Djoko-Djoko, S., Bozga, M., Bensalem, S.: Building distributed sensor network applications using BIP. In: 2015 IEEE Sensors Applications Symposium SAS 2015. 2015 IEEE Sensors Applications Symposium SAS 2015, Zadar, Croatia, April 13-15, 2015, IEEE, Zadar, Croatia (Apr 2015) 6.4

[20] Lekidis, A., Bozga, M., Mauuary, D., Bensalem, S.: A model-based design flow for CAN-based systems. In: 13th International CAN Conference. iCC'13, Paris, France (October 2013) 6.4

[21] Lekidis, A., Stachtiari, E., Katsaros, P., Bozga, M., Georgiadis, C.K.: Using BIP to reinforce correctness of resource-constrained IoT applications. In: 10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015. pp. 245–253. IEEE, Siegen, Germany (Jun 2015) 6.4

[22] Mutch, J., Lowe, D.G.: Object class recognition and localization using sparse features with limited receptive fields. International Journal of Computer Vision 80(1), 45–57 (2008), http://link.springer.com/article/10.1007/s11263-007-0118-0 6.2

[23] Nouri, A., Bensalem, S., Bozga, M., Delahaye, B., Jegourel, C., Legay, A.: Statistical model checking QoS properties of systems with sbip. Int. J. Softw. Tools Technol. Transf. (STTT) 17(2), 171–185 (April 2015) 1, 2, 3

[24] Nouri, A., Bozga, M., Molnos, A., Legay, A., Bensalem, S.: Building faithful high-level models and performance evaluation of manycore embedded systems. In: Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014. pp. 209–218 (2014) 6.2

[25] Nouri, A., Bozga, M., Molnos, A., Legay, A., Bensalem, S.: Astrolabe: A rigorous approach for system-level performance modeling and analysis. ACM Trans. Embed. Comput. Syst. 15(2), 31:1–31:26 (mar 2016) 6.2

[26] Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: International Conference on the Quantitative Evaluation of Systems, QEST'05. pp. 251–252 (2005) 2

[27] Wald, A.: Sequential tests of statistical hypotheses. Annals of Mathematical Statistics 16(2), 117–186 (1945) 4.2, 5.1

[28] Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005) 1, 4.2, 5.1

[29] Younes, H.L.S.: Ymer: A statistical model checker. In: COMPUTER AIDED VERIFICATION, CAV'05. pp. 429–433. Springer (2005) 2