Statistical Model Checking of Stochastic Timed-Arc Petri Nets

Tanguy Dubois[1], Kim G. Larsen[2], and Jiří Srba[2]

[1] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France tanguy.dubois@ec-nantes.fr
[2] Aalborg University, Aalborg, Denmark {kgl,srba}@cs.aau.dk

Abstract. Timed-Arc Petri nets (TAPN) are a timed extension of Petri nets where tokens have their age and each arc is associated with a time interval restricting the ages of tokens available for transition firing. Additionally, a TAPN can also contain place invariants constraining the ages of tokens in places, inhibitor arcs preventing a transition from firing and transport arcs that preserve a token's age upon firing. This set of features, as much as it allows us to model complex systems, also often makes verification problems computationally hard or even undecidable. Moreover, in order to model real-life examples, additional stochastic aspects are often necessary to capture the desired behaviour. We suggest (to the best of our knowledge) the first stochastic semantics for TAPNs and design and implement the Statistical Model Checking (SMC) algorithms in the model checker TAPAAL. We argue for the semantic choices we made in the stochastic semantics and prove that the semantics is well-behaving. On a number of case studies we demonstrate the practical applicability of our modelling formalism and its SMC implementation.

1 Introduction

In many complex systems, exact deterministic models fail to capture the inherent uncertainty and variability present in real-world processes. Introducing stochastic elements to formal models allows us to better represent these unpredictable dynamics, providing a more realistic framework for analysis [19]. However, the exact verification of these models often becomes infeasible due to the sheer computational complexity involved. Instead, statistical model checking [30] offers a practical alternative: it provides approximate but computationally efficient results. While statistical methods may not guarantee absolute precision, they offer a valuable trade-off between accuracy and performance, enabling meaningful insights into systems that are otherwise too complex to analyze.

Research on introducing stochastic semantics to Petri nets focuses on incorporating randomness into the modelling of systems with uncertain or probabilistic behaviour. Stochastic Petri nets [18,4] extend traditional Petri nets by assigning probabilistic firing rates to transitions, enabling the modelling of systems with random delays, such as in queuing or communication networks. This model is further extended to generalized stochastic Petri nets [3] which include

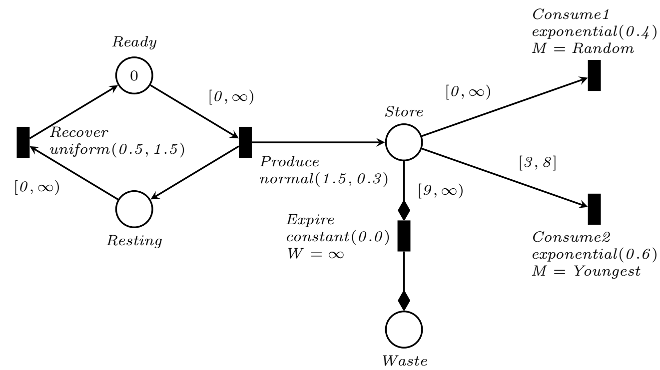


Fig. 1: Stochastic TAPN modelling a produced with two consumers

also immediate transitions and further extensions with e.g. colored tokens [12]. A complex timing behaviour can be modelled by stochastic time Petri nets [14] where apart from the stochastic aspects every transition also contains a firing interval that defines the earliest and latest firing date of every transition. As there are only finitely many transitions (and hence we need to use only a fixed number of clocks), the SMC engine of UPPAAL is used in [14] to perform statistical model checking. On the other hand, the model of timed-arc Petri nets (TAPN) introduced in [7,21] (see also an overview paper [23]), requires a new clock for each token in the net and the total number of clocks cannot be a priori fixed. TAPNs equipped with transport arcs are more expressive than timed Petri nets and timed automata [9,37], however, a stochastic semantics for this model has not been suggested yet. In this paper, we consider the classical TAPN model extended with additional features like transport arcs, age invariants and inhibitor arcs as used in the model checker TAPAAL [15]. We suggest the first stochastic semantics to this extended model, implement a new statistical model checking engine as a part of the TAPAAL suite and evaluate it on a number of realistic case studies.

We finish the introduction by giving an intuition of our stochastic extension of TAPNs by describing a simple producer/consumer system depicted in Figure 1. The net in the figure models a producer that is either in the place *Ready* or *Resting*. As soon as the transition *Produce* is enabled (e.g. in the initial marking that contains one token of age 0 in the place *Ready*), we sample its firing date $d \in \mathbb{R}_{\geq 0}$ from normal distribution with mean 1.5 and standard deviation 0.3. The system will then delay for d time units followed by firing the transition *Produce* which deposits new tokens of age 0 into the places *Resting* and *Store*. At this moment, two transitions become newly enabled, namely *Recover* and *Consume1*, and we sample their firing dates from uniform distribution between 0.5 and 1.5, and exponential distribution with rate 0.4, respectively. The system will now delay until the next interesting event occurs, meaning that either one of the two enabled transitions reaches its firing time and will fire, or a new transition becomes enabled. In the latter case, after three time units the transition *Consume2* becomes enabled as the token's age now fits into the interval $[3, 8]$ and we sample its firing date from the exponential distribution with rate 0.6.

Now three transitions are enabled and the one with the earliest scheduled firing time will fire, unless another 5 time units pass and the transition *Consumer2* gets disabled and its scheduled firing time is deleted. Should no transition be scheduled for an additional time unit then *Expire* becomes enabled. Its firing date is sampled from the constant distribution with mean 0, meaning that it is scheduled to fire immediately and it will move the product from *Store* to *Waste* while preserving its age 9 (the special arrow types are the transport arcs which move tokens without changing their ages). The transition *Expire* has weight ∞, meaning that in case that at the same time other transitions are also enabled, *Expire* will get the priority in firing (assuming that the default weight of all other transitions is 1).

Finally, as the place *Store* can possibly contain multiple tokens of different ages, the firing modes of transitions *Consume1* and *Consume2* specify which tokens will be consumed during the firing. For *Consume1* we shall use randomly selected token that enables the transition, however, *Consume2* will always use the youngest available token in the interval $[3, 8]$.

We may now ask what is the probability that there appears a product in the place *Waste* within the first 20 time units of execution of the system, which we can, using our SMC tool that executed 461 121 runs, estimate to have the probability 0.045 ± 0.002 with 95% confidence. This probability will approach the value 1 as the time horizon gets longer.

2 Related work

There are two main approaches for the verification of probabilistic timed systems. Symbolic probabilistic verification methods [28] aim to produce the exact probability of observing a given property by computing or approximating the probabilistic measure. The tool PRISM [29] is known to implement such methods. On the other hand statistical model-checking (SMC) [30] generates random runs to produce an estimation of the evaluation of a property. While the symbolic method can be very costly by exploring the stochastic state space, the latter is a lot faster at the cost of precision. However, SMC is known to become costly and imprecise when it comes to the verification of rare events, and methods such as importance splitting [24] and importance sampling [25] have been introduced to solve this issue.

A version of probabilistic semantics for one-clock timed automata has been proposed in [2]. Although they cannot contain multiple clocks, their extension to networks of timed automata [17] can model systems implementing several concurrent timed probabilistic components. These are the semantics implemented in UPPAAL's SMC engine [16].

The first and the simplest stochastic extension of Petri nets (sPN) is detailed in [18]. Every transition becomes a stochastic component and samples its firing

Semantics	Time Constraints	Type
sPN (ORIS) [4]	No	Strong
DSPN (TimeNet) [39]	No	Strong
HPnGs (HYPEG) [35]	No	Strong
sTPN (UPPAAL) [14]	Yes	Strong
sTAPN (TAPAAL)	Yes	Weak

Fig. 2: Stochastic timed semantics for Petri nets

date according to a given distribution, resulting in a race where the transition with the earliest firing date wins and fires. This semantics is simple to understand but only handles transition delays without any timing constraints and every sampled date is considered valid. Verification of sPN can be done e.g. using the tool ORIS [33]. The basic stochastic Petri nets were extended with immediate transitions and transitions with deterministic delays into the Deterministic and Stochastic Petri nets (DSPN) [31], which are implemented in tools like TimeNet 4.0 [39]. Hybrid Petri nets with multiple general transitions (HPnGs) [35] also support these features, but also allow for continuous transitions. The verification of HPnGs can be done using the tool HYPEG (formerly libhpng) [34]. Both essentially use the same method for sampling dates, and thus have the same issues with missing time constraints, however, by mixing stochastic transitions with deterministic timed transitions and continuous transitions, this type of stochastic nets can model more complex timed behaviour.

There exists a stochastic extension of the Time Petri net (TPN) semantics, which achieves even more expressive power by associating a time firing interval to each transition [14]. These nets implement the *strong* semantics [8] that guarantess that a transition must be fired within its firing interval and the distributions used for the transitions are essentially scaled to match the firing intervals. An implementation using the UPPAAL SMC engine was initially used to perform SMC on these semantics [14]. In stochastic TPN time constraints are supported but are limited to the transitions in the net, which cannot be dynamically changed. On contrary, timed-arc Petri nets (TAPN) [7,21], studied in this paper, associate the timing information to potentially unbounded number of tokens in the net and input arcs to transition restrict the ages of tokens that can be consumed by transition firing. Traditionally, TAPNs rely on the *weak* semantics meaning that it is allowed to perform time delays that can disable currently enabled transitions. Further extensions with urgent transitions or age invariants [23] are needed to enforce urgent behaviour. Moreover, an extension of the timed-arc Petri net model with only transport arcs already makes the model more expressive than other types of timed nets and timed automata [9,37]. No stochastic semantics for TAPNs or other Petri net models that includes the weak (nonurgent) semantics has been given yet. This is, including an efficient implementation of the framework, the main contribution of this paper. An overview table of selected semantics and respective tools is given in Figure 2.

3 Timed-Arc Petri Nets

We shall first introduce the semantics of timed-arcs Petri nets without the stochastic attributes. Let $\mathcal{I}$ be the set of *timed intervals* of the from $[a, b]$ where $a \in \mathbb{N}^0$, $b \in \mathbb{N}^0 \cup \{\infty\}$ and $a \leq b$. Let *time invariants* $\mathcal{I}_{\mathsf{inv}} \subseteq \mathcal{I}$ be a subset of timed intervals of the form above where $a = 0$.

Definition 1 (Timed-Arc Petri net). *A timed-arc Petri net (TAPN) is a 7-tuple* $\mathcal{N} = (P, T, IA, OA, \mathsf{Transport}, \mathsf{Inhib}, \mathsf{Inv})$ *where*

- P *is a finite set of places,*
- T *is a finite set of transitions such that* $P \cap T = \emptyset$*,*
- $IA \subseteq P \times \mathcal{I} \times \mathbb{N} \times T$ *is the set of input arcs that connect a place to a transition and are annotated by a time interval and an arc weight, such that if* $(p, I, w, t) \in IA$ *and* $(p, I', w', t) \in IA$ *then* $I = I'$ *and* $w = w'$*,*
- $OA \subseteq T \times \mathbb{N} \times P$ *is the set of weighted output arcs that connect transitions to places such that if* $(t, w, p) \in OA$ *and* $(t, w', p) \in OA$ *then* $w = w'$*,*
- $\mathsf{Transport} \subseteq IA \times OA$ *is the set of transport arcs such that whenever* $((p, I, w, t), (t', w', p')) \in \mathsf{Transport}$ *then* $t = t'$ *and* $w = w'$*, and if* $(\alpha, \beta), (\alpha, \beta') \in \mathsf{Transport}$ *then* $\beta = \beta'$ *and symmetrically if* $(\alpha, \beta), (\alpha', \beta) \in \mathsf{Transport}$ *then* $\alpha = \alpha'$*,*
- $\mathsf{Inhib} \subseteq P \times \mathbb{N} \times T$ *is the set of weighted inhibitor arcs, and*
- $\mathsf{Inv} : P \to \mathcal{I}_{\mathsf{inv}}$ *is the function assigning age invariants to places.*

We note that the definition implies that a given place and a transition cannot be connected by both the normal and transport arc at the same time. For a transition t, we denote by $\mathsf{Pre}(t) = \{p \in P \mid (p, I, w, t) \in \mathrm{IA}\}$ the set of input places, and by $\mathsf{Post}(t) = \{p' \in P \mid (t, w', p') \in \mathrm{OA}\}$ the set of output places. A marking represents the distribution of tokens together with their ages across the places in the net.

Definition 2 (Marking). *A* marking M *is a multiset over* $P \times \mathbb{R}_{\geq 0}$ *such that* (p, x) *represents a token of age* x *in the place* p*. A marking* M *is valid if, for each place* $p \in P$*, whenever* $(p, x) \in M$ *then* $x \in \mathsf{Inv}(p)$*. We denote the set of all valid markings by* $\mathcal{M}$*.*

For a marking $M \in \mathcal{M}$, we use the notation $M(p) = \{x \mid (p, x) \in M\}$ to denote the multiset of token ages in the place p.

Definition 3 (Enabled transition). *A transition* $t \in T$ *is* enabled *in a marking* M *if there exist multisets of tokens* $In \subseteq \mathsf{Pre}(t) \times \mathbb{R}_{\geq 0}$ *and* $Out \subseteq \mathsf{Post}(t) \times \mathbb{R}_{\geq 0}$ *such that*

- $In \subseteq M$*, i.e. the* In *set contains only tokens present in the marking* M*,*
- *for every* $(p, I, w, t) \in IA$ *and every* $x \in In(p)$ *we have* $x \in I$ *and* $|In(p)| = w$*, i.e. each input arc has exactly* w *tokens in its input place, all of them satisfying its time guard,*

– $|M(p)| < w$ *for every* $(p, w, t) \in \mathsf{Inhib}$*, i.e. none of the inhibitor arcs has enough tokens in its input place to inhibit the firing of* t*,*
– *for every* $((p, I, w, t), (t, w, p')) \in \mathsf{Transport}$ *and every* $x \in \mathit{In}(p)$ *we have* $x \in \mathsf{Inv}(p')$ *and* $x \in \mathit{Out}(p')$ *where* $|\mathit{Out}(p')| = w$*, i.e. all* w *tokens in* p *can be moved to* p' *and still satisfy the age invariant of* p'*, and*
– *for any* $(t, w', p') \in \mathit{OA}$ *that does not appear in* $\mathsf{Transport}$ *we have* $|\mathit{Out}(p')| = w'$ *and* $x = 0$ *for every* $x \in \mathit{Out}(p')$*, i.e. every output arc that is not a transport arc must create exactly* w' *new tokens of age* 0*.*

We denote the set of transitions enabled in a marking M by $\mathsf{en}(M)$.

Definition 4 (Transition firing). *If* t *is enabled in a marking* M *by the multisets of tokens* In *and* Out *then* t *can* fire *and produce a marking*

$$M' = (M \setminus \mathit{In}) \cup \mathit{Out}$$

where $\setminus$ *and* $\cup$ *are operations on multisets, and we write* $M \xrightarrow{t} M'$*.*

Definition 5 (Delay). *Let* M *be a marking and let* $d \in \mathbb{R}_{\geq 0}$ *be a real number. We can delay* d *time units from* M *if* $x + d \in \mathsf{Inv}(p)$ *for all* $(p, x) \in M$*, and write* $M \xrightarrow{d}$ *to indicate that a delay is possible. If the delay is possible, we write* $M \xrightarrow{\delta} M[d]$*, where* $M[d]$ *is a marking defined by* $M[d] = \{(p, x{+}d) \mid (p, x) \in M\}$*.*

A *run* of a TAPN from an initial marking M_0 is an alternating sequence of time delays and transition firings such that

$$M_0 \xrightarrow{d_0} M_0[d_0] \xrightarrow{t_0} M_1 \xrightarrow{d_1} M_1[d_1] \xrightarrow{t_1} M_2 \xrightarrow{d_2} M_2[d_2] \xrightarrow{t_2} \dots$$

A run is *maximal* if it is either infinite or ends in a deadlock, i.e. in a marking M such that for any possible delay d we have $\mathsf{en}(M[d]) = \emptyset$. By $\mathsf{runs}(\mathcal{N})$ we denote the set of all maximal runs of $\mathcal{N}$.

3.1 A Logic for Reasoning about Runs

We shall now define a logic to argue about marking properties. A *marking property* φ is a Boolean combination of atomic propositions of the form $p \bowtie n$ where $p \in P$, $n \in \mathbb{N}^0$ and $\bowtie \in \{<, \leq, =, \geq, >\}$ with the obvious semantics that a marking M satisfies an atomic property $p \bowtie n$ if and only if $|M(p)| \bowtie n$, i.e. the number of tokens in a place p satisfies the constraint imposed by the atomic proposition. This is naturally extended to Boolean connectives and we write $M \models \varphi$ if a marking M satisfies the marking property φ. Let Φ be the set of all marking properties. Note that properties that involve the age of the tokens can be encoded by introducing "monitoring" transitions with appropriate time intervals and checking their enabledness.

A run of a TAPN satisfies the (reachability) formula $F\varphi$ if it contains a marking M such that $M \models \varphi$, and it satisfies the formulate $G\varphi$ if every marking M on the run satisfies $M \models \varphi$.

4 Stochastic Timed-Arc Petri Nets

Let $\mathcal{N} = (P, T, \mathrm{IA}, \mathrm{OA}, \mathsf{Transport}, \mathsf{Inhib}, \mathsf{Inv})$ be a TAPN as defined in the previous section. We shall now extend it with stochastic features by adding three additional functions[3] assigning (i) density function to transitions (for sampling firing delays), (ii) weights to transition (for the resolution for firing conflicts) and (iii) firing mode to transitions (for deciding which tokens are used in transition firing).

The function $\mathsf{density} : T \to \mathcal{F}(\mathbb{R}, \mathbb{R}_{\geq 0})$ assigns a probability density function[4] for each transition. It is used to sample firing dates of newly enabled transitions. When we sample a delay for a transition $t \in T$ according to $\mathsf{density}(t)$, we assume that we choose a random value according to the distribution, except for the case when the sampled value is negative, in which case we return the delay of 0.

The function $\mathsf{weight} : T \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ assigns a weight (priority mass) to each transition. The weights are used in the event of a firing date collision in case several transitions sample the same firing date. Let $T' \subseteq T$ be the set of transitions that chose the same firing date then the probability of firing $t \in T'$ is given by a weighted uniform choice

$$\frac{\mathsf{weight}(t)}{\sum_{t' \in T'} \mathsf{weight}(t')} \ .$$

If one of the competing transitions has an infinite weight then it will always win the competition, except for the situation where there are several transitions with infinite weight—in this case we choose uniformly between them. On the other hand, if one of the competing transitions has a zero weight, then it will never be chosen, unless every other competing transition also has a zero weight—in that case we choose again uniformly among the zero-weight transitions.

Finally, the firing mode function $\mathsf{mode} : T \to \{\mathit{Youngest}, \mathit{Oldest}, \mathit{Random}\}$ is a function that determines for each transition which tokens are consumed when the transition is fired in a given marking. Assume that $t \in T$ is enabled in M by the multisets of tokens $In_1, \ldots, In_n$ (enumerating all the possiblities). If $\mathsf{mode}(t) = \mathit{Youngest}$ then the transition t will be fired using the set In_i, $1 \leq i \leq n$, that minimizes the sum of ages of all tokens in In_i and similarly if $\mathsf{mode}(t) = \mathit{Oldest}$ then the sum will be maximized. In case that that there are several such sets that minimize/maximize the sum, we uniformly choose one such set. In case $\mathsf{mode}(t) = \mathit{Random}$, we select In_i with uniform probability i/n.

Stochastic TAPNs will use a *single-server policy* [5], meaning that once a transition has fired, it starts its firing process again (if enabled); and an *enabling memory policy* [5], meaning that firing dates are stored as long as their transition is enabled, and then get forgotten as soon as it gets disabled. Transition firings that both consume and produce a token to some place do not change enabledness of other transitions that also consume from such a place.

[3] These functions are defined over the domain of real numbers but in the actual implementation are represented as doubles.

[4] Where for each $t \in T$, the area under the function is one: $\int_{\mathbb{R}} \mathsf{density}(t)(\theta) d\theta = 1$.

4.1 Algorithm for Generating Random Runs

The heart of an SMC algorithm is the generation of random runs from an initial marking M_0, until we reach a deadlock, a given time or step (number of transition firings) bound, or a marking that satisfies a given marking property. The generation of random runs is executed according to the following strategy.

- Newly enabled transitions randomly sample their firing date according to the corresponding density function.
- The net then delays to the next interesting moment where either one of the enabled transitions is scheduled to fire or the enabledness of transitions changes and their scheduled firing dates get updated (if a transition becomes disabled, it is unscheduled and if it becomes newly enabled, we sample according to its density function).
- If several transitions are scheduled to fire at the same date, we select the winner according to the function weight and fire the transition using the tokens selected by the function mode. The winner then resets its firing date.
- By firing the winning transition, other transitions may become enabled or disabled. We update their scheduled firing dates accordingly and repeat the whole process until a marking property is satisfied (in which case we return true) or until we exceed the given number of steps (transition firings) or the given time horizon. In this case we return false.

The random run generation is formally described in Algorithm 1. An explanation of this algorithm follows.

1. While the accumulated delay and steps are under the specified bounds, we repeat until reaching a marking satisfying the given marking property, or a deadlock:
   (a) We look for the next smallest interesting delay, which is a date where the enabledness of some transition changes (line 7). In particular the condition (ii) expresses the fact that when an age of a token reaches the upper bound of a time interval on some arc, an arbitrarily small delay can disable a currently enabled transition.
   (b) If we cannot find such delay, and there are no scheduled transition firings, we reached a deadlock and terminate (line 8).
   (c) Otherwise, we delay up to the minimum between the smallest interesting delay and the earliest scheduled firing date (line 11).
   (d) If more than one transition is scheduled to fire at this date, we randomly choose a winner t using the weight function, and we fire it by selecting a multiset of consumed tokens according to $\mathsf{mode}(t)$ (lines 15 to 21).
   (e) We update the scheduled firing dates of each transition: disabled transitions are scheduled at ∞, the firing date of scheduled transitions is shifted according to the delay and newly enabled transitions are sampled according to their density functions (lines 22 to 28). The check at line 27 truncates negative sampled values to zero.
   (f) We check if any scheduled transition will get disabled after an arbitrary small delay and unschedule such a transition (line 29).

Algorithm 1 $\mathsf{RandRun}_{\mathcal{N}}(M_0, \varphi, c, s)$

Input: Marking M_0, property φ, time bound $c \geq 0$, and steps bound $s \geq 0$
Output: Boolean indicating if a random run generated from M_0 contains a marking satisfying φ before c time units passed and k transitions were fired
for all $t \in T$: $\mathsf{scheduled}(t) \leftarrow$ sample from $\mathsf{density}(t)$ if $t \in \mathsf{en}(M_0)$, else ∞
$\mathsf{AccumulatedDelay} \leftarrow 0$; $\mathsf{AccumulatedSteps} \leftarrow 0$; $M \leftarrow M_0$
while $\mathsf{AccumulatedDelay} \leq c$ and $\mathsf{AccumulatedSteps} \leq s$ **do**
    if $M \models \varphi$ **then return** true
    select the smallest positive delay d, $0 < d \in \mathbb{R}_{\geq 0}$, satisfying
 (i) $\mathsf{en}(M) \neq \mathsf{en}(M[d])$, or
 (ii) $\exists \delta > 0. \forall \varepsilon \in (0, \delta].\ \mathsf{en}(M) \neq \mathsf{en}(M[d + \varepsilon])$.
    if no such d exists and $\mathsf{scheduled}(t) = \infty$ for all $t \in T$ **then**
        return false ▷ We are in a deadlock
    else
        $d \leftarrow \min\{\ d,\ \min_{t \in T} \mathsf{scheduled}(t)\ \}$ ▷ Select an earliest interesting delay
    end if
    $M \leftarrow M[d]$ ▷ Advance to the next interesting time point
    $\mathsf{AccumulatedDelay} \leftarrow \mathsf{AccumulatedDelay} + d$
    $\mathsf{candidates} \leftarrow \{t \in T \mid \mathsf{scheduled}(t) - d = 0\}$ ▷ Transitions that can fire
    if $\mathsf{candidates}$ is not empty **then**
        randomly select $t_{\mathsf{fired}} \in \mathsf{candidates}$ using the weight function $\mathsf{weights}$
        let $M \xrightarrow{t_{\mathsf{fired}}} M'$ according to $\mathsf{mode}(t_{\mathsf{fired}})$; $M \leftarrow M'$
        $\mathsf{scheduled}(t_{\mathsf{fired}}) \leftarrow \infty$ ▷ The executed transition is unscheduled
        $\mathsf{AccumulatedSteps} \leftarrow \mathsf{AccumulatedSteps} + 1$
    end if
    for $t \in T$ **do** ▷ Update the scheduled firing times of each $t \in T$
        if $t \notin \mathsf{en}(M)$ **then**
            $\mathsf{scheduled}(t) \leftarrow \infty$
        else
            $$\mathsf{scheduled}(t) \leftarrow \begin{cases} \mathsf{scheduled}(t) - d & \text{if } \mathsf{scheduled}(t) \neq \infty \\ \text{sample from } \mathsf{density}(t) & \text{otherwise} \end{cases}$$
            $\mathsf{scheduled}(t) \leftarrow 0$ if $\mathsf{scheduled}(t) < 0$
        end if
        if $\mathsf{scheduled}(t) > 0$ and $\exists \delta > 0. \forall \varepsilon \in (0, \delta].\ t \notin \mathsf{en}(M[\varepsilon])$ **then**
            $\mathsf{scheduled}(t) \leftarrow \infty$ ▷ Unschedule t as it will get disabled
        end if
    end for
end while
return false

We shall first point out that we do not need urgent transitions (once enabled time cannot elapse) in the stochastic semantics, as urgency can be simulated by Dirac density function which always samples the value 0 and hence time cannot elapse as long as at least one such urgent transition is enabled. Contrary to non-stochastic semantics, age invariants in places cannot be used to enforce urgency. Once a token age in a place reaches the invariant upper bound, the execution

Algorithm 2 ProbabilityEstimation$_{\mathcal{N}}(M_0, \varphi, c, s, \rho, \epsilon)$

Input: Marking M_0, property φ, time bound $c \geq 0$, steps bound $s \geq 0$, confidence $0 < \rho < 1$, precision $0 < \epsilon < 1$
Output: The probability $\pm\epsilon$ that φ is satisfied with confidence ρ in no more than c time units and s steps
$N \leftarrow \frac{\ln(2/(1-\rho))}{2\epsilon^2}$ ▷ The number of runs to execute
$x \leftarrow 0$
for $i = 1$ to N **do**
    $x \leftarrow x +$ RandRun$_{\mathcal{N}}(M_0, \varphi, c, s)$ ▷ Consider true as 1 and false as 0
end for
return $\frac{x}{N}$

of the random run will end in a deadlock (unless some transition was actually scheduled to fire at that time).

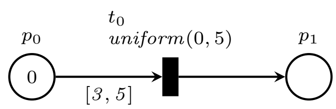
The statistical model checking algorithm presented in Algorithm 2 now uses the random run generator to perform Monte-Carlo simulations for quantitative probability estimation [17]. It uses the Chernoff-Hoeffding bound to compute, for a given precision ϵ and a confidence level ρ, the number of runs N to be executed in the SMC algorithm. Now, a confidence-interval estimating process—that executes N runs and returns the proportion of φ-satisfying runs—will have probability larger than ρ of returning a value that is no more than ϵ away from the real (unknown) probability of φ being satisfied.

4.2 Examples of Random Runs

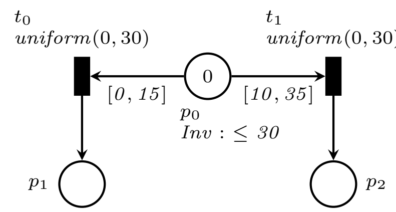
In order to better understand the details of the random run generation algorithm, we shall now exemplify it on a number of examples presented in Figure 3. In these example, the default $[0, \infty)$ time intervals as well as the default weight 1 and the default random firing mode are omitted.

Let us first consider the net from Figure 3a. Initially, t_0 is not enabled so we will delay to the next interesting event, which is after 3 time units where t_0 becomes enabled. At this moment, we sample a delay d uniformly from the interval $[0, 5]$. If $3 + d \leq 5$ then the next interesting delay is $3 + d$ where we fire t_0 and reach a deadlock. If $3 + d > 5$ then the next interesting delay is 2 time units and the token in p_0 reaches the age 5. As t_0 is not yet scheduled to fire at this point and after arbitrarily small delay t_0 becomes disabled, we unschedule the firing of t_0 as the check at line 29 of the algorithm succeeds. The run then deadlocks.

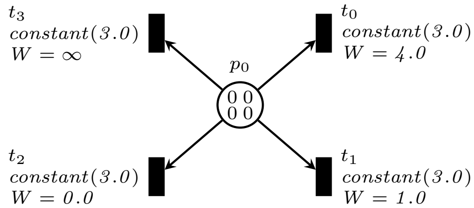
A race behaviour between two transitions is shown in Figure 3b. Initially, t_0 is enabled and it chooses a firing date d_0 uniformly from the interval $[0, 30]$. If $d_0 \leq 10$ then t_0 will fire as the next interesting delay is d_0 where t_0 is the only enabled transition. If $d_0 > 10$ then at 10 units the transition t_1 will also sample its firing delay d_1 from the interval $[0, 30]$. Now if $d_0 - 10 < d_1$ and $d_0 \leq 15$ then the transition t_0 wins the race. Should $d_0 - 10 > d_1$ then t_1 wins the race and fires. If none of the transitions fired until 15 time units, t_0 gets unscheduled.



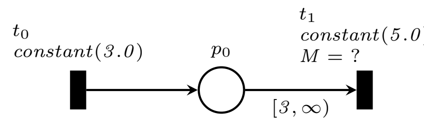
(a) Single transition



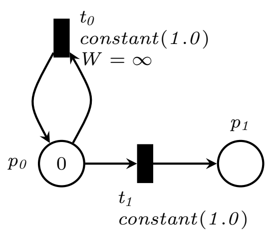
(b) Simple race



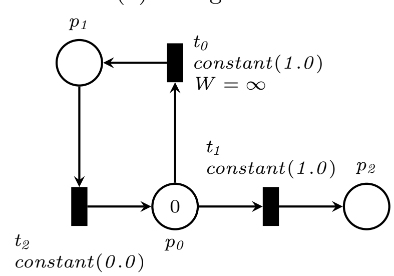
(c) Date collision



(d) Firing mode



(e) Atomic firing



(f) Sequential firing

Fig. 3: Examples of stochastic timed-arc Petri nets

Depending on the sampled delay d_1, either t_1 fires provided that $d_1 + 10 \leq 30$, or the whole net deadlocks if $d_1 + 10 > 30$ due to the invariant on p_0 that blocks any time progress once the token in p_0 reaches the age 30. Note that the firing date collision of t_0 and t_1 is unlikely as the probability that the transition delays are sampled so that $d_0 = d_1 + 10$ is zero.

However, in case of constant distributions, collision of firing dates is possible as demonstrated in Figure 3c. As there are four tokens of age 0 in the initial marking, each of the four transitions can fire. The scheduled firing data of each transition will be 3 and we need to resolve the probability in which order they fire. As the weight of t_3 is infinity, it will be always the first transition to fire. After firing t_3, three transitions remain enabled. Transition t_0 will fire next with the probability 4/5 and transition t_1 will fire with the probability 1/5. After both t_0 and t_1 fired, t_2 will fire last as its weight is 0 and it could not compete with t_0 and t_1.

The example in Figure 3d shows the influence of the firing mode (depicted by $M = \mathit{mode}$ where $\mathit{mode} \in \{\mathit{Random}, \mathit{Youngest}, \mathit{Oldest}\}$) on executions. The transition t_0 produces a token every 3 time units. Once the token in p_0 reaches the age 3, also t_1 becomes enabled and will fire after 5 time units. At that point, there will be tokens of ages 8, 5, and 2 in p_0, and t_1 needs to choose which one of the token ages satisfying the time interval $[3, \infty)$ it consumes during the firing. If $\mathsf{mode}(t_1) = \mathit{Youngest}$, then t_1 select the token of age 5. If $\mathsf{mode}(t_1) = \mathit{Oldest}$, it select the token of age 8. If $\mathsf{mode}(t_1) = \mathit{Random}$, then t_1 uniformly chooses between the tokens of ages 5 and 8.

Finally, Figures 3e and 3f demonstrate a difference when a transition gets unscheduled during the random run generation. In Figure 3e the transitions t_0 and t_1 get scheduled at time 1 as they both sample from the constant distribution which always returns 1. As the weight of t_0 is ∞ and the default weight of t_1 is 1, the transition t_0 fires first, resets the age of the token in p_0 to 0, and samples its next firing date (which is again after 1 time unit). However, after the firing t_0, the transition t_1 is still enabled so its scheduled firing date does not change and it fires (without any further delay) immediately after t_0. The reader may wonder that during the firing of t_0, the transition t_1 got temporarily disabled and it should be resampled too. If such behaviour is desirable then it can be modelled as shown in Figure 3f. In this case the probability that the transition t_1 fires is zero, as it gets always resampled after any firing of t_0 (which always wins the race as its weight is ∞).

4.3 Induced Probabilistic Semantics

The random run algorithm for a stochastic TAPN $\mathcal{N}$ defines an uncountable set of possible runs and induces a probabilistic measure $\mathbb{P}_\mathcal{N}$ on infinite sets of runs of the σ-algebra generated by cylinders of runs of the form

$$\pi = I_0.t_0.\ldots.I_n.t_n$$

where $I_i \in \mathcal{I}$ is a time interval and $t_i \in T$ is a transition for all i, $0 \leq i \leq n$. Given a run $\rho = M_0 \xrightarrow{d_0} M_0[d_0] \xrightarrow{t_0} \rho' \in \mathsf{runs}(\mathcal{N})$ and a cylinder $\pi = I_0.t_0'.\pi'$, we write $\rho \in \pi$ if $d_0 \in I_0$, $t_0 = t_0'$ and $\rho' \in \pi'$; for the base-case we postulate that $\rho \in \varepsilon$ for any run ρ. In other words, the prefix of the concrete run must agree on the transition firings with the cylinder and the concrete delays must belong to the intervals in the cylinder.

For the sake of simplicity, we assume that every density function is continuous, hence no firing date collision may occur. This means that the scheduling function $\mathsf{scheduled}$ from Algorithm 1 satisfies that at most one transition may be fireable at the minimum scheduled delay. In the same way, we assume that the firing mode for each transition is deterministic, meaning that given an origin marking and a transition t_f, there is a unique marking reachable by firing t_f. At any point during the algorithm, the next possible delay d is deterministically determined as the minimum between scheduled firing date or the time points where transition enabledness changes. There are three cases, (i) either the next delay is a transition firing date and then a firing is triggered, or (ii) the next delay is an enabledness change and no transition fires, or (iii) there is no next delay and we reach a deadlock.

Therefore, given a set of persistent transitions pers (enabled before and after the delay and firing) and a set of newly-enabled transitions newen (disabled before, but enabled after the delay and firing), a previous scheduling function $\mathsf{scheduled}$, and a delay $d \in \mathbb{R}_{\geq 0}$, we define the set of next scheduling functions $\mathsf{SCH}(d)$ as:

$$\mathsf{ns} \in \mathsf{SCH}(d) \iff \begin{cases} \mathsf{ns}(t_i) = \mathsf{scheduled}(t_i) - d & \text{if } t_i \in \mathsf{pers} \\ \mathsf{ns}(t_i) \in \mathbb{R}_{\geq 0} & \text{if } t_i \in \mathsf{newen} \\ \mathsf{ns}(t_i) = +\infty & \text{otherwise .} \end{cases} \tag{1}$$

By D we denote the density function on $\mathsf{SCH}(d)$ which assigns $D(\mathsf{ns}) \in \mathbb{R}_{\geq 0}$ to every scheduling function $\mathsf{ns} \in \mathsf{SCH}(d)$ given the set of newly-enabled transitions newen as follows:

$$D(\mathsf{ns}) = \prod_{t_i \in \mathsf{newen}} \mathsf{density}(t_i)(\mathsf{ns}(t_i)) \ . \tag{2}$$

Finally, let $\mathbb{1}_{cond}$ be an indicator variable equal to 1 if condition *cond* is true, else 0. The probabilistic measure is now defined on cylinders using one of three formulas. In the main case (i), assuming that in a marking M the given function $\mathsf{scheduled}$ determines to fire the transition t_f after a delay d and reach a new marking M', the measure is defined inductively as:

$$\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), I_0.t_0.\pi') = \mathbb{1}_{d \in I_0} \cdot \mathbb{1}_{t_f = t_0} \cdot \left(\int_{s \in \mathsf{SCH}(d)} D(s) \cdot \mathbb{P}_{\mathcal{N}}((M', s), \pi') \, \mathrm{d}s \right) \ .$$

The other two cases (ii) and (iii) are defined analogously. Observing that $\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \pi)$ decreases when extending the cylinder π, it follows from classical concepts of probability theory (see e.g. [1]) that $\mathbb{P}_{\mathcal{N}}$ extends uniquely to the smallest σ-algebra, $\mathfrak{S}^{\mathcal{N}}$, generated by the above cylinders.

Theorem 1. *Given a stochastic TAPN $\mathcal{N}$, $\mathbb{P}_{\mathcal{N}}$ is a probability measure over the σ-algebra $\mathfrak{S}^{\mathcal{N}}$ generated by the set of cylinders of the form $(\mathcal{I}.T)^*$.*

Most importantly $F\varphi$ and $G\varphi$ properties define measurable sets of runs, thus having well-defined probabilities as postulated by the next theorem.

Theorem 2. *Given a marking property $\varphi \in \Phi$, the formulae $F\varphi$ and $G\varphi$ describe sets of runs belonging to the σ-algebra $\mathfrak{S}^{\mathcal{N}}$, thus having well-defined probabilities $\mathbb{P}_{\mathcal{N}}(F\varphi)$ and $\mathbb{P}_{\mathcal{N}}(G\varphi)$.*

The last theorem proves that the computed probabilities indeed make sense, by exhibiting an equivalence with classical TAPNs.

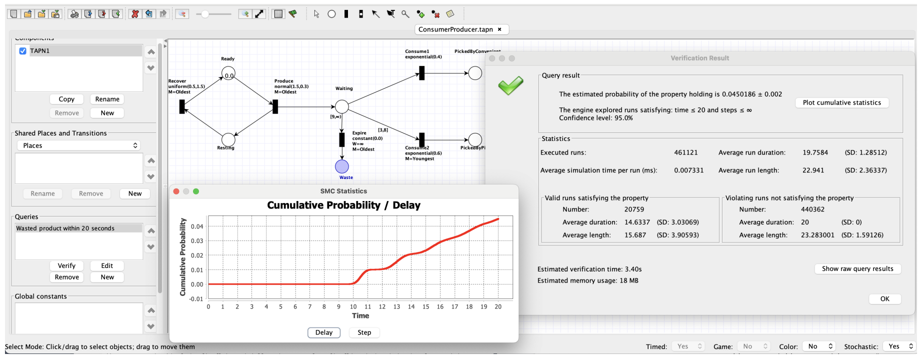


Fig. 4: TAPAAL's GUI with a running example and its SMC verification output

Theorem 3. *Let $\mathcal{N}$ be a stochastic TAPN and let $\mathcal{N}_d$ be a normal TAPN net obtained by removing all stochastic features in $\mathcal{N}$. Let φ be a marking property.*

(i) If there is no reachable marking in $\mathcal{N}_d$ that satisfies φ then $\mathbb{P}_{\mathcal{N}}(F\varphi) = 0$.
(ii) If all reachable markings in $\mathcal{N}_d$ satisfy φ then $\mathbb{P}_{\mathcal{N}}(G\varphi) = 1$.

5 Implementation in TAPAAL and Case Studies

The SMC run generation is implemented in C++ as part of the `verifydtapn` engine [26] and accepts a PNML description of stochastic TAPN together with the query and other SMC parameters and it executes Algorithm 2 and returns the computed probability as well as other useful statistics like average run length and duration, number of satisfying and violating runs and data for plotting cumulative probabilities and other plots that allow to visualize the average/minimum/maximum number of tokens in places during the simulations. The existing TAPAAL GUI [15] is extended to allow for the editing of stochastic TAPNs as well as queries, it communicates with the SMC engine and visualizes the results of SMC verification. A screenshot using our running producer/consumer example with the window showing the result with the statistics as well as the cumulative probability is shown in Figure 4.

We currently support the uniform and geometric discrete distributions and dirac delta, uniform, exponential, normal, gamma, triangular and log normal continuous distributions. For the given set of distribution parameters, the GUI also visualizes the density function. We support three types of verification: quantitative probability estimation, using a Monte-Carlo algorithm; qualitative probability test, using a sequential probability ratio test; and a mode to generate and simulate traces, which can be either any traces, traces satisfying a property, or traces violating a property. The SMC engine has an option to use multi-threading and, if enabled, it executes one run generator per available core and achieves almost a linear speedup with increasing number of CPU cores. Moreover, we support the estimation of the total running time of the SMC algorithm from the

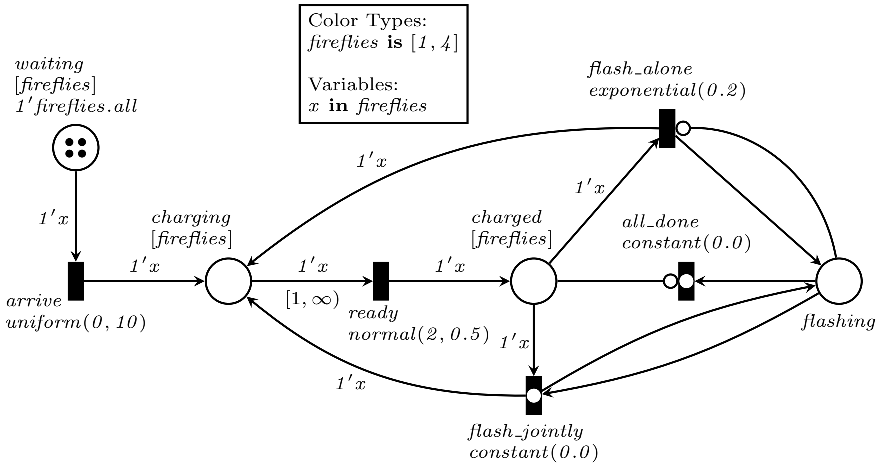


Fig. 5: Colored stochastic Petri net model of fireflies

given confidence and precision parameters, or alternatively suggest a precision to match a given running time and confidence level.

We shall now present three case studies demonstrating the capabilities of our tool. A release of TAPAAL SMC can be downloaded from `https://www.tapaal.net/download/`. The models used in the case studies can be found in other downloads at the bottom of the page.

5.1 Fireflies

Firefly is a beetle that is capable of emitting light flashes in periodic intervals. In nature, some firefly species like Photinus carolinus [38] show a remarkable synchronization capabilities so that a population of fireflies starts, after a while, to flash in synchrony. All this without any leader and in a completely distributed manner. This behaviour attracted the attention among scientists, trying to explain and model such behaviour (see e.g. [36,32]). In a simplified version, each firefly requires some amount of time in order to charge and be able to flash. The discharge (flash) can then occur after some random delay, unless the firefly notices a flash from other firefly, in which case it will join the flash (assuming that it is also charged).

In Figure 5 we present a stochastic timed-arc Petri net model of the fireflies flashing behaviour. We use here the well-known colored extension of Petri nets where tokens can carry additional information (in our case the id of the firefly). Initially, there are four fireflies (tokens) in the place *waiting*, having ids one to four and initially of age 0. With uniform distribution between 0 to 10 seconds, the fireflies independently arrive to the place *charging*. Here each firefly must be charged before the transition *ready* moves it to the charged location. The charging takes at least one second due to the time interval $[1, \infty)$ on the arc. After

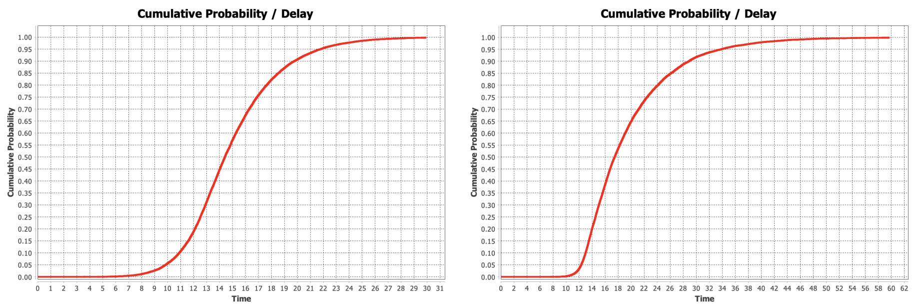


Fig. 6: Cumulative synchronization probability for 4 (left) and 10 (right) fireflies

the transition *ready* is enabled, an additional delay is sampled from a normal distribution with mean value of 2 seconds and a standard deviation of 0.5. Once a firefly arrives to the place *charged*, there are two situations. Either the place *flashing* has no token, meaning that there is currently no one emitting a flash. In this case the firefly waits an amount of time sampled from the exponential distribution of rate 0.2 (corresponding to the mean value of 5 seconds) and once it is schedule to fire the transition *flash_alone*, it returns to the charging location. As a side-effect, a token (uncolored) is placed to the place *flashing*, which disables other fireflies from flashing alone. A token in the place *flashing* indicates that other fireflies in the place *charged* should join the flash immediately. This is done by firing the transition *flash_jointly* which samples from the constant distribution which always returns the value 0. In other words, the transition *firing* is scheduled immediately and the transition hence becomes urgent (indicated by the circle in the middle of the transition). As soon as there are no further fireflies in the place *charged*, the urgent transition *all_done* becomes enabled and removes the token from the place *flashing*.

In our tool, a colored Petri net is unfolded [13,6] to the standard timed-arc Petri net without colors by creating a copy of each place for each color and a copy of each transition for each binding of the variable x to different fireflies ids. The probability distributions of each transition are then simply overtaken by the unfolded copies of the transition.

We can now ask about the probability that all fireflies synchronize within 30 seconds. This can be formulated by the query

```
F (charging = 1 and flashing = 1 and waiting = 0)
```

requiring that we reach a situation where there are no fireflies in the place *waiting*, exactly one firefly in the place *charging* and a token in *flashing*, indicating that a single firefly initiated a flash and all other fireflies (that are in the place *charged*) are joining in a synchronous flash. Our tool executes 18456 runs to estimate that the probability of synchronization within 30 seconds is 0.997 with 95% confidence and precision 0.01. If we increase the number of fireflies to 10, the probability of synchronous flashing within 30 seconds drops to 0.917 and we

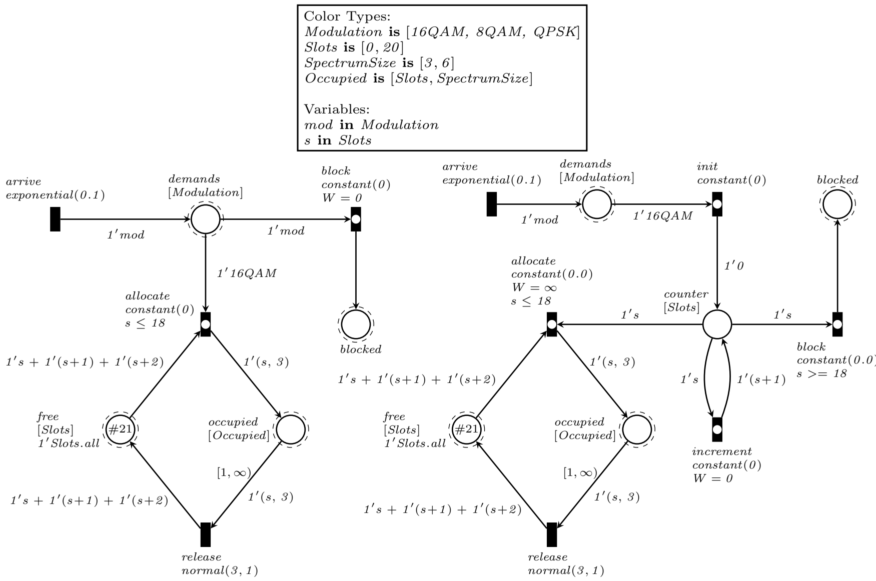


Fig. 7: Random fit (left) and first fit (right) spectrum allocation for 16QAM

need at least 60 seconds to achieve the synchronization probability 0.998. Cumulative probability plots for 4 and 10 fireflies, produced by our tool, are depicted in Figure 6.

5.2 Frequency Spectrum Assignment in Elastic Optical Networks

Elastic all-optical networks [11] allow for fine-grained resource allocation technologies in order to schedule network traffic demands on different light frequencies inside a single optical fiber. When a new demand arrives, the spectrum allocation problem is to find for each optical fiber a sequence of available frequency slots that will carry the demand. The frequency slots must be contiguous [20] (follow each other) and the number of required slots depends on the modulation scheme [27]. After some time, a demand can be released, making the frequency slots allocated for the demand available again. This allocation/deallocation process can create a fragmentation in the allocated frequency slots, possibly resulting in a situation when a newly arrived demand is blocked (there are not enough available consecutive frequency slots to accommodate the demand).

In our stochastic timed-arc Petri net model depicted in Figure 7, we study two spectrum allocation strategies: a *random fit* that randomly chooses from the available slots, trying to uniformly distribute the demands across the whole spectrum, and a strategy called *first fit* [10] that always uses the lowest possible available frequency slot. We consider three common modulation schemes 16QAM, 8QAM and QPSK that require 3, 4 and 6 frequency slots [27], respectively. In Figure 7 we present the Petri net model for 16QAM only but the other two modulations are modelled anologously, just requiring a higher number of slots. We use again the colored extension in order to allow for a more compact model description. In our model, we use the variable *mod* that ranges over the color type *Modulation*, the variable s ranging over the 21 frequency slots declared in the color type *Slots* and we also use the color type *Occupied* which is a Cartesian product of the *Slots* color type and the available spectrum size with values 3 to 6. The dashed circles around places denote the so-called shared places that appear also in the nets for the other modulations (not shown in our Petri net model). The shared place called *free* contains tokens with colors (frequency slots) that are currently available for allocation of a new demand. Initially, there are 21 tokens in the place *free* with color values from 0 to 20. On the other hand, the shared place *occupied* contains tokens with colors being pairs of integers $(\mathit{frequency}, \mathit{size})$ such that e.g. the value $10, 4$ means that the frequency slots 10, 11, 12 and 13 are occupied by a demand that requires 4 consecutive frequency slots.

The random fit slot allocation (Figure 7 left) gathers the arrived demands (assuming an exponential distribution of arrivals with rate 0.1) in the place called *demands*. Immediately after a demand arrival, the urgent transition *block* becomes enabled but it has weight 0, meaning that if the transition *allocate* is also enabled, it will have a priority. The transition *allocate* binds s to some initial frequency slot (no more than 18 as the demand requires three slots) and checks whether the frequency slots s, $s + 1$ and $s + 2$ are available (i.e. there are three tokens with these colors in the the place *free*). If this is the case, these three tokens are removed from the place *free* and a new token with the color $(s, 3)$ is added to *occupied*. After one second holding time, enforced by the interval $[1, \infty)$, the transition *release* becomes enabled and samples its firing date from the normal distribution with mean 3 and standard deviation 1 and releases these three slots once it fires. In case the transition *allocate* is enabled for several bindings of the variable s to different frequency slots, the used binding is selected using a uniform distribution.

In Figure 7 (right) we describe the first fit spectrum allocation algorithm that always uses the lowest available frequency slots. This is achieved by placing a token with the color 0 (the lowest frequency slot) to the place *counter* immediately after any demand arrival. By firing the urgent transition *increment*, we then repeatedly increase the color value of the token by 1 until we find an available slot (using the same modelling approach as in the random assignment) or until we reach the frequency slot 18 where the transition *block* becomes enabled. The weight of the transition *allocate* is infinity, meaning that it has a priority over the transition *block* but in case the transition is still not enabled, the transition *block* will fire as the transition increment has weight 0.

We can now ask our tool to compare the blocking probability of the two allocation algorithms by issuing the SMC query `F blocked >= 1`, checking wheater at least one demand is blocked within the first 30 seconds. With 95% confidence and precision 0.01, our tool executes 18488 runs and estimates the blocking probability to 0.075 for the first fit slot allocation strategy and 0.167 for the random

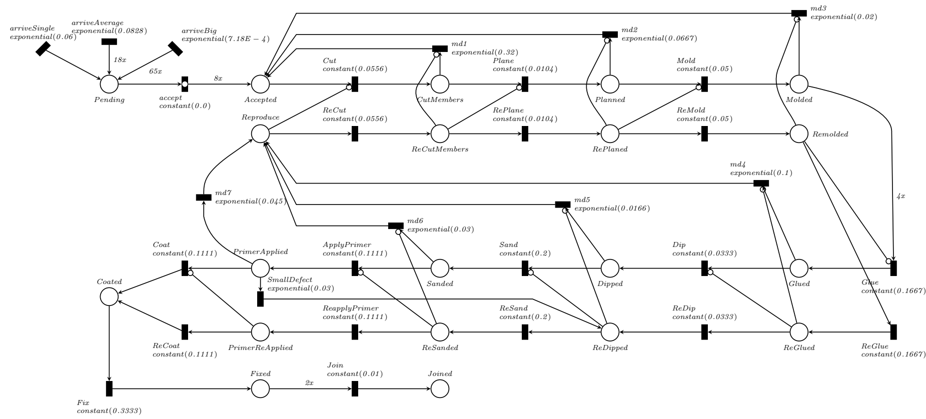


Fig. 8: Manufacturing process of wooden windows [22]

fit strategy, showing an improved performance of the former strategy. This behaviour is also conformed by experiments in computer networking literature [10].

5.3 Manufacturing Process of Wooden Windows

The workflow of a Hungarian wooden window manufacturing company Holz-Team Ltd details a systematic production process, from preparing individual frame components to assembling and finishing complete windows. Orders vary in size, and the process includes measures to address material defects, with early-stage issues being easier to resolve than those found later. The Petri net model is given in Figure 8 and contains timing probabilities derived from real measurements performed by the company as described in [22]. The workflow uses two types of transitions: exponentially delayed transitions for unpredictable events like order arrivals and material defects, and deterministic transitions for fixed-time production phases such as cutting, molding, and coating. For a detailed description of the process consult [22].

The production process faced a bottleneck where coated windows that required a hardware fix accumulated over time in the place called *Coated*. Indeed, using TAPAAL SMC engine we estimated that with probability 0.61 ± 0.01 and 95% confidence, the weekly production process generates more than 20 unprocessed coated windows. A solution to the bottleneck was suggested in [22] and required the hiring of an additional worker to reduce the hardware fixing operation from 20 minutes to 13 minutes. After this adjustment, the probability of storing 4 or more unprocessed windows dropped to 0.011. In Figure 9 (left) we depict, based on 50 simulations of the workflow for a period of one year (2008 hours), the number of accepted orders and the average number of unprocessed coated windows for the situation before the workflow adjustment. After the adjustment, the number of unprocessed coated windows stays on average slightly above 1 with a maximum between 3 to 5 windows, as shown in Figure 9 (right).

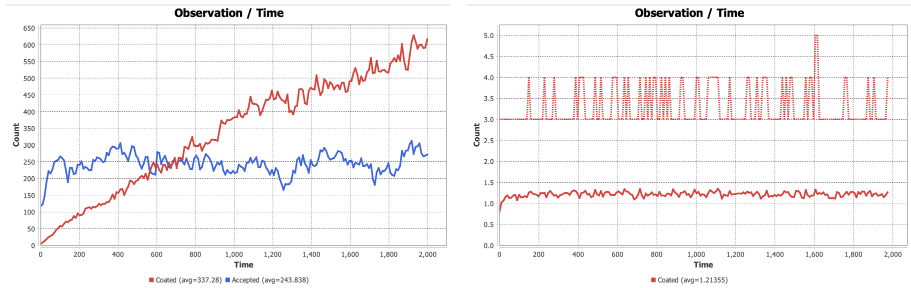


Fig. 9: Average number of orders and unprocessed coated windows over a year

This allowed the company to produce on average about 260 additional windows per year, increasing the productivity by almost 10%.

6 Conclusion

We proposed, to the best of our knowledge, the first stochastic semantics to the popular timed-arc Petri net model. The semantics is weak and race-based, implying that an enabled transition may not necessarily fire within its firing interval but can instead delay out of its enabledness zone, based on the firing date sampled from any probabilistic distribution (we currently support 7 continuous distributions and two discrete ones, but the approach is not limited only to these). This allows us to model a wide range of stochastic systems, as demonstrated in our case studies.

We discussed the design and implementation of the statistical model checking algorithm and argued that it is well-defined. All algorithms are implemented in the open-source model checker TAPAAL, including numerous performance optimizations (e.g. parallel execution) as well as a support for the visualization of the statistical model checking results. The initial experiments indicate that the formalism is applicable to a broad range of problems and allows us to reason about complex stochastic behaviours in an intuitive way.

In the future work, we would like to extend the SMC reachability formulae to a more powerful logic (like e.g. LTL) that will allow us to reason about probability of runs that satisfy additional requirements. An optimization of our SMC algorithm for handling rare events is another line of research.

Acknowledgements We thank Mikkel Tygesen for his help with TAPAAL GUI and in particular the visualization of the SMC results. This paper was funded by the Villum Investigator project S4OS and the ANR project BisoUS ANR-22-CE48-0012.

References

1. Ash, R.B., Doleans-Dade, C.A.: Probability and Measure Theory. Academic Press (2000)
2. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Probabilistic and Topological Semantics for Timed Automata. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science. pp. 179–191. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77050-3_15
3. Balbo, G.: Introduction to Generalized Stochastic Petri Nets, pp. 83–131. Springer (2007). https://doi.org/10.1007/978-3-540-72522-0_3
4. Bause, F., Kritzinger, P.: Stochastic Petri Nets: An Introduction to the Theory. AMS (Nov 2013)
5. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of different semantics for time Petri nets. In: Peled, D.A., Tsay, Y.K. (eds.) Automated Technology for Verification and Analysis. pp. 293–307. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
6. Bilgram, A., Jensen, P., Pedersen, T., Srba, J., Taankvist, P.: Methods for efficient unfolding of colored Petri nets. Fundamenta Informaticae **189**(3–4), 297–320 (2023)
7. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed lotos. Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol pp. 395–408 (01 1990)
8. Boyer, M., Roux, O.H.: Comparison of the Expressiveness of Arc, Place and Transition Time Petri Nets. In: Kleijn, J., Yakovlev, A. (eds.) Petri Nets and Other Models of Concurrency – ICATPN 2007. pp. 63–82. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73094-1_7
9. Boyer, M., Roux, O.H.: On the compared expressiveness of arc, place and transition time Petri nets. Fundam. Inf. **88**(3), 225–249 (2008)
10. Chatterjee, B.C., Oki, E.: Performance evaluation of spectrum allocation policies for elastic optical networks. In: 2015 17th International Conference on Transparent Optical Networks (ICTON). pp. 1–4 (2015). https://doi.org/10.1109/ICTON.2015.7193485
11. Chatterjee, S., Pawlowski, S.: All-optical networks. Commun. ACM **42**(6), 74–83 (1999). https://doi.org/10.1145/303849.303865
12. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Transactions on Computers **42**(11), 1343–1360 (1993). https://doi.org/10.1109/12.247838
13. Christesen, N., Glavind, M., Schmid, S., Srba, J.: Latte: Improving the latency of transiently consistent network update schedules. In: Proceedings of 38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (PERFORMANCE'20). ACM SIGMETRICS Performance Evaluation Review, vol. 48, no. 3, pp. 14–26. ACM (2020). https://doi.org/10.1145/3453953.3453957
14. Cicirelli, F., Nigro, C., Nigro, L.: Qualitative and quantitative evaluation of stochastic Time Petri Nets. In: 2015 Federated Conference on Computer Science and Information Systems (FedCSIS). pp. 763–772 (Sep 2015). https://doi.org/10.15439/2015F69, https://ieeexplore.ieee.org/abstract/document/7321519

15. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12). LNCS, vol. 7214, pp. 492–497. Springer-Verlag (2012)
16. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical Model Checking for Stochastic Hybrid Systems. Electronic Proceedings in Theoretical Computer Science **92**, 122–136 (Aug 2012). https://doi.org/10.4204/EPTCS.92.9, http://arxiv.org/abs/1208.3856, arXiv:1208.3856 [cs]
17. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Stochastic Semantics and Statistical Model Checking for Networks of Priced Timed Automata (Nov 2014). https://doi.org/10.48550/arXiv.1106.3961, http://arxiv.org/abs/1106.3961, arXiv:1106.3961 [cs] version: 2
18. Florin, G., Fraize, C., Natkin, S.: Stochastic Petri nets: Properties, applications and tools. Microelectronics Reliability **31**(4), 669–697 (Jan 1991). https://doi.org/10.1016/0026-2714(91)90009-V
19. Gardiner, C.: Stochastic Methods: A Handbook for the Natural and Social Sciences. Springer Series in Synergetics, Springer Berlin Heidelberg (2010)
20. Hai, D.T., Morvan, M., Gravey, P.: Combining heuristic and exact approaches for solving the routing and spectrum assignment problem. IET Optoelectronics **12**(2), 65–72 (2018). https://doi.org/https://doi.org/10.1049/iet-opt.2017.0013
21. Hanisch, H.M.: Analysis of place/transition nets with timed arcs and its application to batch process control. In: Application and Theory of Petri Nets 1993. pp. 282–299. Springer (1993)
22. Horváth, Á.: Usability of deterministic and stochastic Petri nets in the wood industry: A case study. In: Advanced Computational Methods for Knowledge Engineering. pp. 119–127. Springer, Cham (2014)
23. Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of Timed-Arc Petri Nets. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Královič, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011: Theory and Practice of Computer Science. pp. 46–72. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18381-2_4
24. Jegourel, C., Legay, A., Sedwards, S.: Importance Splitting for Statistical Model Checking Rare Properties. p. 576 (Jul 2013). https://doi.org/10.1007/978-3-642-39799-8_38, https://inria.hal.science/hal-01087826
25. Jegourel, C., Legay, A., Sedwards, S.: Command-based importance sampling for statistical model checking. Theoretical Computer Science **649**, 1–24 (Oct 2016). https://doi.org/10.1016/j.tcs.2016.08.009, https://www.sciencedirect.com/science/article/pii/S0304397516303966
26. Jensen, P., Larsen, K., Srba, J., Sørensen, M., Taankvist, J.: Memory efficient data structures for explicit verification of timed systems. In: Proceedings of the 6th NASA Formal Methods Symposium (NFM'14). LNCS, vol. 8430, pp. 307–312. Springer-Verlag (2014). https://doi.org/10.1007/978-3-319-06200-6_26
27. Kawaguchi, K., Seki, Y., Tanigawa, Y., Hirota, Y., Tode, H.: Proactive modulation format allocation method with selective downgrading to enhance inter-core crosstalk tolerance in sdm-eons. In: 2024 15th International Conference on Network of the Future (NoF). pp. 159–163 (2024). https://doi.org/10.1109/NoF62948.2024.10741410
28. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking: Advances and applications. In: Formal System Verification. pp. 73–121. Springer (2017)

29. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification. pp. 585–591. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
30. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical Model Checking, pp. 478–504. Springer International Publishing (2019). https://doi.org/10.1007/978-3-319-91908-9_23
31. Marsan, M.A., Chiola, G.: On Petri nets with deterministic and exponentially distributed firing times. In: Rozenberg, G. (ed.) Advances in Petri Nets 1987. pp. 132–145. Springer, Berlin, Heidelberg (1987). https://doi.org/10.1007/3-540-18086-9_23
32. McCrea Madeline, E.B., E., R.J.: A model for the collective synchronization of flashing in photinus carolinus. J. of Royal Soc. Interface (19) (2022)
33. Paolieri, M., Biagi, M., Carnevali, L., Vicario, E.: The ORIS Tool: Quantitative Evaluation of Non-Markovian Systems. IEEE Transactions on Software Engineering **47**(6), 1211–1225 (Jun 2021). https://doi.org/10.1109/TSE.2019.2917202, https://ieeexplore.ieee.org/document/8719961, conference Name: IEEE Transactions on Software Engineering
34. Pilch, C., Edenfeld, F., Remke, A.: HYPEG: Statistical Model Checking for hybrid Petri nets: Tool Paper. In: Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools. pp. 186–191. VALUETOOLS 2017, Association for Computing Machinery, New York, NY, USA (Dec 2017). https://doi.org/10.1145/3150928.3150956, https://doi.org/10.1145/3150928.3150956
35. Pilch, C., Remke, A.: Statistical Model Checking for Hybrid Petri Nets with Multiple General Transitions. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 475–486 (Jun 2017). https://doi.org/10.1109/DSN.2017.41, https://ieeexplore.ieee.org/abstract/document/8023146, iSSN: 2158-3927
36. Ramírez-Ávila, G.M., Kurths, J., Depickère, S., Deneubourg, J.L.: Modeling Fireflies Synchronization, pp. 131–156. Springer (2019). https://doi.org/10.1007/978-3-319-78512-7_8
37. Srba, J.: Comparing the expressiveness of timed automata and timed extensions of Petri nets. In: Proceedings of the 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'08). LNCS, vol. 5215, pp. 15–32. Springer-Verlag (2008)
38. Wikipedia: Photinus carolinus — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Photinus%20carolinus&oldid=1237034011 (2025), [Online; accessed 09-January-2025]
39. Zimmermann, A., Knoke, M.: Timenet 4.0: A software tool for the performability evaluation with stochastic and colored petri nets. In: User manual (2007)

A Implementation Details

Most parts of the SMC algorithm for run generation are written in a pseudocode that is easy to implement, however, we shall now discuss in more details the check (ii) on line 7 and the condition on line 29 in Algorithm 1 that are harder to compute. We cannot iterate through infinitely many small epsilons in order to detect if a time point is an upper bound of an enabledness zone, and we cannot store every possible firing date. Instead, as we only allow closed intervals for arcs, the firing intervals for transitions will be unions of closed intervals. Internally, given a marking M, for each transition t we compute a list of non-intersecting intervals $F(t, M) = ([x_0, y_0], \ldots, [x_n, y_n])$ such that:

- $\forall 0 \leq i < j \leq n.\ y_i < x_j$
- $\delta \in \bigcup_{0 \leq i \leq n} [x_i, y_i] \iff M \xrightarrow{\delta}$ and $t \in \mathsf{en}(M[\delta])$.

We now show how to compute F given a marking M. First, we compute the maximum possible delay:

$$\mathsf{MaxDelay}(M) = \min\{b - x \mid (p, x) \in M, \mathsf{Inv}(p) = [0, b]\} \ .$$

Then, for each input arc (p, I, w, t), we compute the dates in which a set of tokens satisfies its guard as follows:

$$\mathsf{ArcDates}(p, I, w, M) = \bigcup_{(x_1, \ldots, x_w) \in M(p)} \left(\bigcap_{i=1}^{w} (I - x_i) \right) \ .$$

Note than here $M(p)$ is a multiset, therefore it can contain several occurrences of the same age and $[a, b] - x$ is the interval $[a - x, b - x]$. In case the computed interval contains negative numbers, these will be removed later on in the definition of $F(t, M)$. This set can be computed by implementing the set operations $\{\cup, \cap\}$ for list of intervals.

We can then apply it directly to input arcs, but also to transport arcs by intersecting the destination invariant of the place p' with the input interval I: $\mathsf{ArcDates}(p, I \cap \mathsf{Inv}(p'), w, M)$. We denote by $\mathsf{InputDates}(t, M)$ the intersection of those dates for input arcs, and $\mathsf{TransportDates}(t, M)$ the intersection for transport arcs. Therefore, if a transition is not inhibited in a marking M, its firing dates are:

$$F(t, M) = \mathsf{InputDates}(t, M) \cap \mathsf{TransportDates}(t, M) \cap [0, \mathsf{MaxDelay}(M)] \ .$$

Being able to compute $F(t, M)$ for any transition t and marking M, we can replace the computation of d at line 7 of Algorithm 1 with:

$$d = \min_{t \in T}(\{x_i \mid [x_i, y_i] \in F(t, M), x_i > 0\} \cup \{y_i \mid [x_i, y_i] \in F(t, M), y_i > 0\}) \ .$$

Given a transition t and a marking M, if $F(t, M) = ([x_0, y_0], \ldots)$, we can replace the condition in line 29 with:

$$\mathsf{scheduled}(t) > 0 \land y_0 = 0 \ .$$

In fact, given a marking M, a transition t, and the list $F(t, M)$ as defined above, the dates when t becomes enabled are exactly $(x_i)_{0\leq i\leq n}$, and the dates after which any delay disables t are $(y_i)_{0\leq i<n}$.

B Induced Probabilistic Semantics

In order to avoid overcomplicated formulas, we assume that densities are all continuous, meaning that no firing date collision is possible, to avoid writing the probability mass weighting among the set of transitions sharing the same date, and to avoid the special case where a transition would be enabled over only a time singleton but would fire at that exact instant.

To define the probability measure, a state in a stochastic TAPN can no longer be reduced to a marking M. A state of a sTAPN is now a tuple $(M, \mathsf{scheduled})$ where $\mathsf{scheduled} : T \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ is a function mapping a transition to a scheduled firing date as used in Algorithm 1.

For a given marking M, we define the sets of persistent and newly enabled transitions:

$$\begin{aligned}\mathsf{pers} &= \{t \in T \mid t \in \mathsf{en}(M) \wedge \exists\delta > 0.\forall\varepsilon < \delta.t \in \mathsf{en}(M')\}\\ \mathsf{newen} &= \{t \in T \mid t \notin \mathsf{en}(M) \wedge \exists\delta > 0.\forall\varepsilon < \delta.t \in \mathsf{en}(M')\}\end{aligned}$$

Given these sets of transitions pers and newen, a previous scheduling function $\mathsf{scheduled}$, and a delay $d \in \mathbb{R}_{\geq 0}$, we define in Equation 1 the set of scheduling functions $\mathsf{SCH}(d)$.

Following the stochastic behaviour defined in Algorithm 1, given a state $(M, \mathsf{scheduled})$, there is at most one unique possible delay d:

$$d = \min(\min_{t_i \in \mathsf{en}(M)} \mathsf{scheduled}(t_i), \inf\{d \in \mathbb{R}_{\geq 0} \mid \mathsf{en}(M) \neq \mathsf{en}(M[d])\}\ .)$$

If such d does not exist then we reached a deadlock. In the event that d exists, we write the transition relation:

$$(M, \mathsf{scheduled}) \xrightarrow{d} (M', \mathsf{scheduled}')$$

where M' is defined as:

$$M' = \{(p, x + d) \mid (p, x) \in M\}$$

and clearly $\mathsf{scheduled}' \in \mathsf{SCH}(d)$.

There can be multiple successive delays as long as transitions continue changing enabledness, however as there is a finite amount of tokens, there would be a finite amount of such successive delays before a deadlock or a transition firing.

In a similar way as for delays let t_f be a (unique) transition scheduled for firing such that $\mathsf{scheduled}(t_f) = 0$. Then we write

$$(M, \mathsf{scheduled}) \xrightarrow{t_f} (M', \mathsf{scheduled}')$$

where M' is generated from M according to $\mathsf{mode}(t_f)$. Here for the sake of simplicity we assume a deterministic firing mode, but a stochastic firing mode will only add probability masses to all available M'. Given such two markings Ma and M' as defined above, we update the definition of persistent and newly enabled transitions:

$$\begin{aligned}\mathsf{pers} &= \{t \in T \mid t \in \mathsf{en}(M) \setminus \{t_f\} \wedge \exists\delta > 0.\forall\varepsilon < \delta.t \in \mathsf{en}(M')\}\\ \mathsf{newen} &= \{t \in T \mid t \notin (\mathsf{en}(M) \setminus \{t_f\}) \wedge \exists\delta > 0.\forall\varepsilon < \delta.t \in \mathsf{en}(M')\}\end{aligned}$$

so that t_f is not persistent any more and it can become newly enabled in case it is still enabled after its firing. Clearly, we get $\mathsf{scheduled}' \in \mathsf{SCH}(0)$.

Let us recall the density function D defined in Equation 2. Let $\pi = I_0.t_0 \dots I_n.t_n$ where $t_i \in T$ and $I_i \in \mathcal{I}$ for all i, $0 \leq i \leq n$ be a cylinder. For any state $(M, \mathsf{scheduled})$ in a net $\mathcal{N}$ we define $\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \varepsilon) = 1$ in case that the cylinder consists of the empty sequence. Using an induction on $|\pi|$, we then define the probability measure of a cylinder. There are three cases, either the next delay d results in the firing of a transition t_f, in that case:

$$\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), I_0.t_0.\pi') = \mathbb{1}_{d \in I_0} \cdot \mathbb{1}_{t_f = t_0} \cdot \left(\int_{s \in \mathsf{SCH}(d)} D(s) \cdot \mathbb{P}((M', s), \pi') \,\mathrm{d}s\right) .$$

Alternatively, the next delay d is due to an enabledness change and no transition fires, and in that case:

$$\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), I_0.\pi') = \int_{s \in \mathsf{SCH}(d)} D(s) \cdot \mathbb{P}((M', s), (I_0 - d).\pi') \,\mathrm{d}s .$$

Note that this formula will lead to an inner-induction as the length of the prefix describing the cylinder has not reduced however this case may only be chained a finite amount of times, this will be discussed in the proof of Theorem 1.

Finally, there may not exist any next delay d. In that case, we reached a deadlock and $\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \pi) = 0$.

Remark 1. Note that in order to integrate over $\mathsf{SCH}(d)$, we represent scheduling functions as vectors from $\mathbb{R}_{\geq 0}^{\mathsf{newen} \cup \mathsf{pers}}$.

At last, we define the probability of the initial state, considering that $\mathsf{newen} = \mathsf{en}(M_0)$ and $\mathsf{pers} = \emptyset$:

$$\mathbb{P}_{\mathcal{N}}(M_0, \pi) = \int_{s \in \mathsf{SCH}(0)} D(s) \cdot \mathbb{P}((M_0, s), \pi) \,\mathrm{d}s .$$

Theorem 1. *Given a stochastic TAPN $\mathcal{N}$, $\mathbb{P}_{\mathcal{N}}$ is a probability measure over the σ-algebra $\mathfrak{S}^{\mathcal{N}}$ generated by the set of cylinders of the form $(\mathcal{I}.T)^*$.*

Proof. Inductively on $|\pi|$, we are going to prove that for all state $(M, \mathsf{scheduled})$ and cylinder π, $\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \pi) \in [0,1]$ and if there is a cylinder $\pi' \subseteq \pi$, then $\mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \pi') \leq \mathbb{P}_{\mathcal{N}}((M, \mathsf{scheduled}), \pi)$.

If $|\pi| = 0$, the property is verified. Then, there are three cases according to the nature of the next delay:

- If the next delay results in a transition firing, then because D generates a density function, using the induction on the tail of π, we prove the induction.
- If the next delay would not result in a transition firing, then for each $s \in \mathsf{SCH}(d)$, there is an inner induction as there could only be a finite number of other steps before a transition firing. Hence we may split the integral over continuous zones sharing the same steps sequences, and conclude the induction by unfolding the inner induction until the next transition firing.
- If there is no next delay, then the probability is zero and the induction is verified.

We have now defined a way to assign a probability to each set of runs described by a cylinder. As there is a countable number of cylinders, there exists an unique way to expand this probability to the σ-algebra of runs generated by such cylinders. $\square$

Theorem 2. *Given a marking property $\varphi \in \Phi$, the formulae $F\varphi$ and $G\varphi$ describe sets of runs belonging to the σ-algebra $\mathfrak{S}^{\mathcal{N}}$, thus having well-defined probabilities $\mathbb{P}_{\mathcal{N}}(F\varphi)$ and $\mathbb{P}_{\mathcal{N}}(G\varphi)$.*

Proof. To define the probability of a marking property φ being finally verified, given a net $\mathcal{N}$, we define the net $\mathcal{N}^*$ in which we add a set of transitions $(t_{\varphi,i})_{1 \leq i \leq n}$ matching the property φ. This is possible, because in our logic, any property can be expressed as a disjunction of conjunctions of $\geq$ and $<$ propositions (respectively matched using input and inhibitor arcs). We also modify and split each other transition so that when φ is verified, then only $t_{\varphi,i}$ transitions are enabled. Now we get

$$\mathbb{P}_{\mathcal{N}}(F\varphi) = \mathbb{P}_{\mathcal{N}^*}\left(M_0, \bigcup_{1 \leq i \leq n} \bigcup_{\sigma \in (\mathcal{I}.T)^*.\mathcal{I}} \sigma.t_{\varphi,i}\right)$$

and we can easily extend the notation to define: $\mathbb{P}_{\mathcal{N}}(G\varphi) = 1 - \mathbb{P}_{\mathcal{N}}(F\neg\varphi)$. $\square$

We can also extend the measure to support bounds on time or number of transition firings, either by adding a place initially marked, with an input arc having a time guard going to every $t_{\varphi,i}$ transition for the time bound; or a place marked initially with $N+1$ tokens with an input arc going to every transition for an upper bound N on steps firing.

Theorem 3. *Let $\mathcal{N}$ be a stochastic TAPN and let $\mathcal{N}_d$ be a normal TAPN net obtained by removing all stochastic features in $\mathcal{N}$. Let φ be a marking property.*

(i) If there is no reachable marking in $\mathcal{N}_d$ that satisfies φ then $\mathbb{P}_{\mathcal{N}}(F\varphi) = 0$.

(ii) If all reachable markings in $\mathcal{N}_d$ satisfy φ then $\mathbb{P}_{\mathcal{N}}(G\varphi) = 1$.

Proof. For the case (i), as φ never holds in any reachable marking and the SMC algorithm is a conservative extension of reachability on classical TAPNs, there is no SMC run that satisfies $F\varphi$, implying that the probability $\mathbb{P}_{\mathcal{N}}(F\varphi)$ corresponds to the measure of the empty set, which is clearly equal to 0. Similarly, in the case (ii), any marking obtained during the SMC run always satisfies φ and hence $\mathbb{P}_{\mathcal{N}}(G\varphi)$ is the measure of any possible run, which is clearly equal to 1. $\square$