Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Automatic Verification of the IEEE-1394 Root Contention Protocol with KRONOS and PRISM ⋆

Conrado Daws[1], Marta Kwiatkowska[2], Gethin Norman[2]

[1] University of Twente, P.O. Box 217 7500 AE Enschede, The Netherlands e-mail: daws@cs.utwente.nl
[2] University of Birmingham, Birmingham B15 2TT, United Kingdom e-mail: {M.Z.Kwiatkowska,G.Norman}@cs.bham.ac.uk

The date of receipt and acceptance will be inserted by the editor

Abstract. We report on the automatic verification of timed probabilistic properties of the IEEE 1394 root contention protocol combining two existing tools: the real-time model-checker KRONOS and the probabilistic model-checker PRISM. The system is modelled as a probabilistic timed automaton. We first use KRONOS to perform a symbolic forward reachability analysis to generate the set of states that are reachable with non-zero probability from the initial state, and before the deadline expires. We then encode this information as a Markov decision process to be analyzed with PRISM. We apply this technique to compute the minimal probability of a leader being elected before a deadline, for different deadlines, and study how this minimal probability is influenced by using a biased coin and considering different wire lengths.

1 Introduction

The design and analysis of many hardware and software systems, such as embedded systems and monitoring equipment, requires detailed knowledge of their real-time aspects, in addition to the functional requirements. Typically, this is expressed in terms of *hard* real-time constraints, e.g. "after a fatal error, the system will be shut down in 45 seconds". In the case of safety-critical systems, it is essential to ensure that such constraints are *never* invalidated.

However, in other cases, for example multimedia protocols that perform in the presence of lossy media, such *hard deadlines* can be too restrictive. *Soft deadlines* are then a viable alternative in these cases. For example, a soft deadline of a multimedia system could be that "*with probability at least* 0.96, video frames arrive *within 80 to 100 ms* after being sent". Soft deadlines can also specify *fault-tolerance* and *reliability* properties such as "deadlock will not occur with probability 1", or "the message may be lost with probability at most 0.01".

Recent research [28] has set a theoretical framework for the specification and verification of timed probabilistic systems. Inspired by the success of real-time model-checkers such as Kronos [14] and Uppaal [30], the direction taken is that of automatic verification through *model checking*, adapting the formalisms and algorithms for model-checking of classical (non-probabilistic) timed systems [2] to the case of timed probabilistic systems. Within this approach[1], timed probabilistic systems are modelled as *probabilistic timed automata* [1,28], i.e. timed automata with discrete probability distributions associated with the edges, and properties are specified in the logic PTCTL, which extends the quantitative branching temporal logic TCTL with a probabilistic operator. Due to the denseness of time, model checking algorithms rely on the construction of a finite quotient of the state space of the system, namely the region graph or the forward reachability graph [28]. By adding the corresponding probability distributions to the transitions of the graph we obtain a Markov decision process (MDP). The probability with which a state of this MDP satisfies a property can then be calculated by solving an appropriate linear programming problem [9,7].

In this work we show how, based on these ideas, the real-time model-checker KRONOS [14,25] and the probabilistic model-checker PRISM [26,32] can be combined for the *automatic* verification of the root contention protocol of IEEE 1394, a timed and probabilistic protocol to resolve conflicts between two nodes competing in a leader election process. The property of interest is the minimal probability for electing a leader before a dead-

⋆ Supported in part by the EPSRC grant GR/N22960 and by the European Community Project IST-2001-35304 AMETIST

[1] We consider in this work systems where only discrete probabilities arise.

line. We first use KRONOS[2] to perform a symbolic forward reachability analysis to generate the set of states that are reachable with non-zero probability from the initial state, and before the deadline expires. We then encode this information as a Markov decision process in the PRISM input language. Finally, we compute with PRISM the minimal probability of a leader being elected before a deadline, for different deadlines. Moreover, we investigate the influence of using a biased coin, and wires of different length, on this minimal probability.

This article proceeds as follows. Section 2 introduces probabilistic timed automata and defines probabilistic reachability of a set of states. In section 3 we describe the features of KRONOS and PRISM used in our verification approach. The encoding of the reachability graph in PRISM input language is explained in section 4. Section 5 illustrates this approach with the verification of the root contention protocol of the IEEE 1394 standard. We conclude with Section 6.

2 Probabilistic Timed Automata

A timed automaton [3] is an automaton extended with *clocks*, variables with positive real values which increase uniformly with time. Clocks may be compared to positive integer time bounds to form *clock constraints* such as $(x \geq 2) \wedge (x \leq 5)$. There are two types of clock constraints: *invariants* labelling nodes, and *guards* labelling edges. The automaton may only stay in a node, letting time pass, if the clocks satisfy the invariant. When a guard is satisfied, the corresponding edge can be taken. Transitions are instantaneous, and can be labelled with *clock resets* of the form $x := 0$ meaning that upon entering the target node the value of clock x is set to 0. Probabilistic automata have probability distributions added to the edges, which model the likelihood of the action happening.

Example 1. The probabilistic timed automaton PTA_1 of Figure 1 models a process which repeatedly tries to send a packet after waiting between 4 and 5 ms, and if successful waits for 3 ms before trying to send another packet. The packet is sent with probability 0.99 and lost with probability 0.01 because of an error. Notice that edges belonging to a same distribution must be labelled with the same guard.

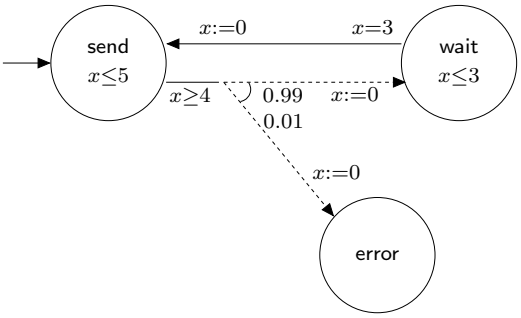

Fig. 1. An example of a probabilistic timed automaton PTA_1.

2.1 Syntax

Clocks and valuations. Let $\mathcal{X}$ be a finite set of variables called *clocks* which take values from the time domain $\mathbf{R}_+$. A *clock valuation* is a point $v \in \mathbf{R}_+^{|\mathcal{X}|}$. The clock valuation $\mathbf{0} \in \mathbf{R}_+^{|\mathcal{X}|}$ assigns 0 to all clocks in $\mathcal{X}$. Let $v \in \mathbf{R}_+^{|\mathcal{X}|}$ be a clock valuation, $t \in \mathbf{R}_+$ be a time duration, and $X \subseteq \mathcal{X}$ a subset of clocks. Then $v+t$ denotes the *time increment* for v and t, and $v[X := 0]$ denotes the clock valuation obtained from v by resetting all of the clocks in X to 0 and leaving the values of all other clocks unchanged.

Zones. Let Z be the set of *zones* over $\mathcal{X}$, which are conjunctions of atomic constraints of the form $x \sim c$ and $x - y \sim c$, with $x, y \in \mathcal{X}$, $\sim \ \in \{<, \leq, \geq, >\}$, and $c \in \mathbf{N}$. A clock valuation v *satisfies* the zone ζ, written $v \models \zeta$, if and only if ζ resolves to true after substituting each clock $x \in \mathcal{X}$ with the corresponding clock value $v(x)$. Let ζ be a zone and $X \subseteq \mathcal{X}$ be a subset of clocks. Then $\nearrow\zeta$ is the zone representing the set of clock valuations $v + t$ such that $v \models \zeta$ and $t \geq 0$, and $\zeta[X := 0]$ is the zone representing the set of clock valuations $v[X := 0]$ such that $v \models \zeta$.

Probability distributions. A *discrete probability distribution (subdistribution)* over a finite set Q is a function $\mu : Q \to [0, 1]$ such that $\sum_{q \in Q} \mu(q) = 1$ ($\sum_{q \in Q} \mu(q) \leq 1$). Let $\mathsf{Dist}(Q)$ ($\mathsf{SDist}(Q)$) be the set of distributions (subdistributions) over subsets of Q.

Definition 1. A *probabilistic timed automaton* is a tuple $\mathsf{PTA} = (L, \bar{l}, \mathcal{X}, \Sigma, I, \mathrm{P})$ where:

- L is a finite set of *locations*;
- $\bar{l} \in L$ is the *initial location*;
- Σ is a finite set of *labels*;
- the function $I : L \to \mathrm{Z}$ is the *invariant condition*; and the finite set $\mathrm{P} \subseteq L \times \mathrm{Z} \times \Sigma \times \mathsf{Dist}(2^{\mathcal{X}} \times L)$ is the *probabilistic edge relation*.

An *edge* takes the form of a tuple (l, g, X, l'), where l is its source location, g is its *enabling condition*, X is the set of resetting clocks and l' is the destination location, such that $(l, g, \sigma, p) \in \mathrm{P}$ and $p(X, l') > 0$.

To aid higher-level modelling, systems can be defined as the *parallel composition* of a number of interacting components. For example, in the case of the IEEE 1394 root contention protocol, it suffices to construct models

[2] We have used an experimental version, not distributed yet, that has been adapted to deal with probability distributions and generates the corresponding output.

for each of the two contending nodes, and the two wires along which they communicate. Based on the theory of (untimed) probabilistic systems [33] and classical timed automata [3], the parallel composition of two probabilistic timed automata, which interact by synchronizing on common events, is defined in the following way.

Definition 2. The *parallel composition* of two probabilistic timed automata $\mathsf{PTA}_1 = (L_1, \bar{l}_1, \mathcal{X}_1, \Sigma_1, I_1, \mathrm{P}_1)$ and $\mathsf{PTA}_2 = (L_2, \bar{l}_2, \mathcal{X}_2, \Sigma_2, I_2, \mathrm{P}_2)$ such that $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, is the probabilistic timed automaton

$$\mathsf{PTA}_1 \| \mathsf{PTA}_2 = (L_1 \times L_2, (\bar{l}_2, \bar{l}_2), \mathcal{X}_1 \cup \mathcal{X}_2, \Sigma_1 \cup \Sigma_2, I, \mathrm{P})$$

where $I(l, l') = I_1(l) \wedge I_2(l')$ for all $(l, l') \in L_1 \times L_2$ and $((l_1, l_2), g, \sigma, p) \in \mathrm{P}$ if and only if one of the following conditions holds:

- $\sigma \in \Sigma_1 \setminus \Sigma_2$ and there exists $(l_1, g, \sigma, p_1) \in \mathrm{P}_1$ such that $p = p_1 \otimes \mu_{(\emptyset, l_2)}$;
- $\sigma \in \Sigma_2 \setminus \Sigma_1$ and there exists $(l_2, g, \sigma, p_2) \in \mathrm{P}_2$ such that $p = \mu_{(\emptyset, l_1)} \otimes p_2$;
- $\sigma \in \Sigma_1 \cap \Sigma_2$ and there exists $(l_1, g_1, \sigma, p_1) \in \mathrm{P}_1$ and $(l_2, g_2, \sigma, p_2) \in \mathrm{P}_2$ such that $g = g_1 \wedge g_2$ and $p = p_1 \otimes p_2$

where for any $l_1 \in L_1$, $l_2 \in L_2$, $X_1 \subseteq \mathcal{X}_1$ and $X_2 \subseteq \mathcal{X}_2$, we let $p_1 \otimes p_2(X_1 \cup X_2, (l_1, l_2)) = p_1(X_1, l_1) \cdot p_2(X_2, l_2)$.

Furthermore, it is often convenient to designate certain locations as being *urgent*; once an urgent location is entered, it must be left immediately, without time passing. The notion of urgency for locations is closely related to the concept of urgent transitions [22,16] (an urgent location is a location for which all outgoing discrete transitions are urgent). Urgent locations can be represented syntactically using the framework given in Definition 1 using an additional clock, combined with additional clock resets and invariant conditions.

2.2 Semantics

A state of a probabilistic timed automaton PTA is a pair (l, v) where $l \in L$ and $v \in \mathbf{R}_+^{|\mathcal{X}|}$ such that $v \models I(l)$, and the automaton starts in the state $(\bar{l}, \mathbf{0})$, that is, in the initial location $\bar{l}$ with all clocks set to 0. If the current state is (l, v), there is a nondeterministic choice of either letting *time pass* while satisfying the invariant condition $I(l)$, or making a *discrete* transition according to any probabilistic edge in P with source location l and whose enabling condition g is satisfied. If the probabilistic edge (l, g, σ, p) is chosen, then the probability of moving to the location l' and resetting to 0 all clocks in X is given by $p(X, l')$.

The semantics of probabilistic timed automata is defined in terms of transition systems exhibiting both nondeterministic and probabilistic choice, called probabilistic systems, which are essentially equivalent to Markov decision processes.

2.2.1 Probabilistic systems.

A *probabilistic system* $\mathsf{PS} = (S, \bar{s}, Act, Steps)$ consists of a set S of *states*, an initial state $\bar{s} \in S$, a set Act of *actions*, and a *probabilistic transition relation* $Steps \subseteq S \times Act \times \mathsf{SDist}(S)$. A *probabilistic transition* $s \xrightarrow{a,\mu} s'$ is made from a state $s \in S$ by first nondeterministically selecting an action-distribution pair (a, μ) such that $(s, a, \mu) \in Steps$, and then by making a probabilistic choice of target state s' according to μ, such that $\mu(s') > 0$.

Definition 3. Given a probabilistic timed automaton $\mathsf{PTA} = (L, \bar{l}, \mathcal{X}, \Sigma, I, \mathrm{P})$, the *semantics* of PTA is the probabilistic system $[\![\mathsf{PTA}]\!] = (S, \bar{s}, Act, Steps)$ defined by the following.

States. Let $S \subseteq L \times \mathbf{R}_+^{|\mathcal{X}|}$ such that $(l, v) \in S$ if and only if $v \models I(l)$ and $\bar{s} = (\bar{l}, \mathbf{0})$.
Actions. Let $Act = \mathbf{R}_+ \cup \Sigma$.
Transitions. Let $Steps$ be the least set of probabilistic transitions containing, for each $(l, v) \in S$:
- for each $t \in \mathbf{R}_+$, let $((l, v), t, \mu) \in Steps$ if and only if $\mu(l, v + t) = 1$ and $v + t' \models I(l)$ for all $0 \le t' \le t$.
- for each $(l, g, \sigma, p) \in \mathrm{P}$, let $((l, v), \sigma, \mu) \in Steps$ if and only if $v \models g$ and for each $(l', v') \in S$:

$$\mu(l', v') = \sum_{X \subseteq \mathcal{X} \,\&\, v' = v[X:=0]} p(X, l') \; .$$

2.3 Probabilistic Reachability

The behaviour of a probabilistic timed automaton PTA is described in terms of the behaviour of its semantics, that is, the behaviour of the probabilistic system $[\![\mathsf{PTA}]\!]$.

Paths. A *path* of a probabilistic system PS is a nonempty finite or infinite sequence of transitions

$$\omega = \bar{s} \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} s_2 \xrightarrow{a_2,\mu_2} \cdots \; .$$

For a path ω and $i \in \mathbf{N}$, we denote by $\omega(i)$ the $(i+1)$th state of ω, and by $last(\omega)$ the last state of ω if ω is finite.

Adversaries. An *adversary* is a function A mapping every finite path ω to a pair $(a, \mu) \in Act \times \mathsf{Dist}(S)$ such that $(last(\omega), a, \mu) \in Steps$ [36]. Let Adv_{PS} be the set of adversaries of PS. For any $A \in Adv_{\mathsf{PS}}$, let $Path_{fin}^A$ and $Path_{ful}^A$ denote the set of finite and infinite paths associated with A. A probability measure $Prob^A$ over $Path_{fin}^A$ can then be defined following [24].

Definition 4. Let $\mathsf{PS} = (S, \bar{s}, Act, Steps)$ be a probabilistic system. Then the *reachability probability* with which a set $F \subseteq S$ of target states can be reached from the initial state $\bar{s}$, for an adversary $A \in Adv_{\mathsf{PS}}$, is:

$$ProbReach^A(F) \stackrel{\text{def}}{=} Prob^A\{\omega \in Path_{ful}^A \mid \exists i \in \mathbf{N} \,.\, \omega(i) \in F\} \; .$$

Furthermore, the *maximal* and *minimal* reachability probabilities are defined respectively as

$$MaxProbReach_{\mathsf{PS}}(F) \stackrel{\text{def}}{=} \sup_{A \in Adv_{\mathsf{PS}}} ProbReach^A(F)$$
$$MinProbReach_{\mathsf{PS}}(F) \stackrel{\text{def}}{=} \inf_{A \in Adv_{\mathsf{PS}}} ProbReach^A(F)$$

2.4 Probabilistic Bisimulation

Definition 5 ([31]). A *probabilistic bisimulation* on a probabilistic system $(S, \bar{s}, Act, Steps)$ is an equivalence relation R on S such that, for all sRs', if $s \xrightarrow{a} \mu$ then there exists a transition $s' \xrightarrow{a} \mu'$ such that for all equivalence classes $C \in [S]_R$:

$$\sum_{t \in C} \mu(t) = \sum_{t \in C} \mu'(t).$$

Two states s_1 and s_2 are called probabilistically bisimilar, denoted by $s_1 \sim s_2$, if and only if there exists a probabilistic bisimulation which contains (s_1, s_2).

A probabilistic system $P = (S, \bar{s}, Act, Steps)$ can be reduced, by quotienting with respect to probabilistic bisimulation, giving an *equivalent* probabilistic system $P_\sim = ([S]_\sim, [\bar{s}], Act, Steps')$, where

$$Steps' \subseteq [S]_\sim \times Act \times \mathsf{Dist}([S]_\sim)$$

is such that $([s], a, \mu') \in Steps'$ if for some $r \in [s]$, $(r, a, \mu) \in Steps$ and for all $C \in [S]_\sim$ we have $\mu'(C) = \sum_{t \in C} \mu(t)$.

Probabilistic bisimulation preserves the behaviour of the systems, that is, bisimilar probabilistic systems exhibit the same behaviour. For instance, in the case of finite state probabilistic systems, PCTL formulas are preserved by probabilistic bisimulation [33]. Here we are interested, in particular, in the preservation of maximal and minimal probabilities for reaching an equivalence class [11]. That is, for any finite state probabilistic system P and $F \in [S]_\sim$:

$$MaxProbReach_P(F) = MaxProbReach_{P_\sim}(F)$$
$$MinProbReach_P(F) = MinProbReach_{P_\sim}(F)$$

3 Verification with KRONOS and PRISM

Due to the denseness of time, the underlying semantic model of a (probabilistic) timed automaton is infinite, and hence effective decision procedures rely on building a finite quotient of the state space, e.g. the region graph or the forward reachability graph. This section describes the verification technique based on the generation of the forward reachability graph with KRONOS, and model checking the obtained graph encoded as a Markov decision process with PRISM.

3.1 Forward Reachability with KRONOS

The forward reachability algorithm of KRONOS proceeds by a graph-theoretic traversal of the reachable state space using a symbolic representation of sets of states, called *symbolic states* [15]. A symbolic state is a pair of the form $\langle l, \zeta \rangle$, with $l \in L$ and $\zeta \in \mathrm{Z}$, such that $\zeta \subseteq I(l)$; it represents all states (l, v) such that $v \models \zeta$. The traversal is based on the iteration of a *successor* operator in two alternating steps: first the computation of the *edge*-successors and then the computation of the *time*-successors of a symbolic state.

Edge Successors. The edge-successor of $\langle l, \zeta \rangle$ with respect to an edge $e = (l, g, X, l')$, such that $(l, g, \alpha, \mu) \in \mathrm{P}$ with $\alpha \in \Sigma$ and $\mu(X, l') > 0$ is [3]

$$\mathsf{edge\_succ}(\langle l, \zeta \rangle, e) = \langle l', (\zeta \wedge g)[X := 0] \wedge I(l') \rangle .$$

Time Successors. The time-successor of $\langle l, \zeta \rangle$ is defined as

$$\mathsf{time\_succ}(\langle l, \zeta \rangle) = \langle l, \nearrow \zeta \wedge I(l) \rangle .$$

Figure 2 shows the reachability graph obtained for the probabilistic timed automaton PTA_1 for a deadline of 15 ms, measured with an extra clock y. Since y is never reset, its value would increase indefinitely. To obtain a finite reachability graph, we need to apply the extrapolation abstraction of [15], which abstracts away the exact value of y when $y > 15$. Notice that this abstraction is exact with respect to reachability properties.

3.2 Model Checking Reachability Properties with PRISM

PRISM [26,32] is a model checker designed to verify different types of probabilistic models: discrete-time Markov chains (DTMCs), Markov decision processes (MDPs) and continuous-time Markov chains (CTMCs). Properties to be checked are specified in probabilistic temporal logics, namely PCTL [9,7] if the model is a DTMC or an MDP, and CSL [6] in the case of a CTMC. We focus on the model checking of reachability properties on MDPs, since a (nondeterministic) probabilistic reachability graph belongs to this class of model, and deadline properties are specified as time bounded reachability properties.

3.2.1 Model Checking MDPs.

Model checking of Markov decision processes is based on the computation of the minimal probability $\mathsf{p}(\mathsf{s}, \Diamond \phi)$, or

[3] Notice the use of the same non probabilistic edge-successor operator than for timed automata. The probabilistic information is however kept in the transitions between symbolic states and used later to generate the MDP.

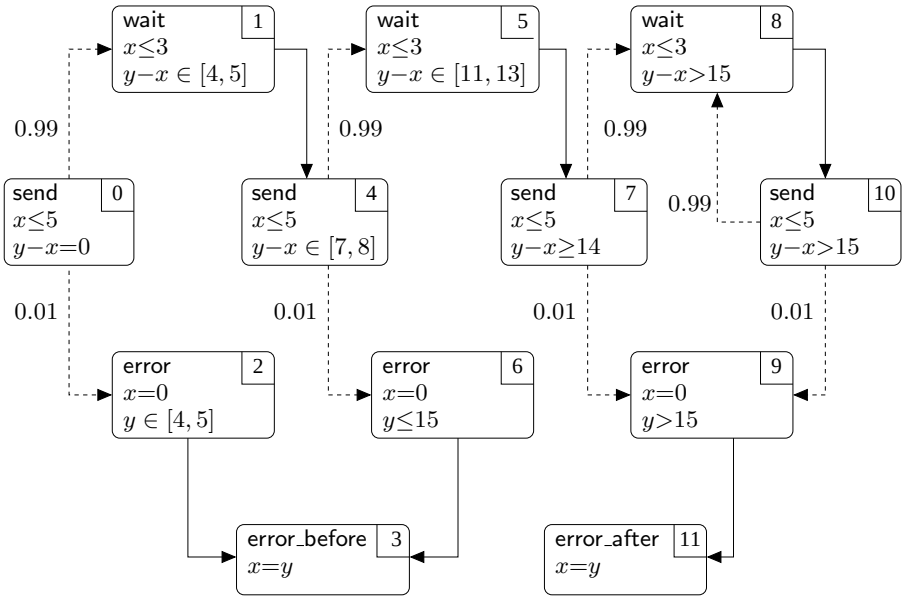

Fig. 2. Reachability graph of PTA_1

the maximal probability $\mathsf{P}(\mathsf{s}, \Diamond\, \phi)$, with which a state s satisfies a reachability formula $\Diamond\, \phi$. Then, a state s satisfies the PTCL formula $\mathcal{P}_{\leq\lambda}(\Diamond\phi)$ if and only if $\mathsf{P}(\mathsf{s}, \Diamond\, \phi) \leq \lambda$. Maximal and minimal probabilities are computed by solving a linear programming problem [9,17]. The *iterative* algorithms implemented in PRISM to solve this problem can combine different numerical computation methods with different data structures [18,27].

3.2.2 Model Checking PTAs.

We verify a PTA by model checking its probabilistic reachability graph using the following result [28]: the maximal probability computed on the reachability graph is an *upper bound* to the maximal probability defined on the semantic model of the probabilistic timed automaton. That is,

$$MaxProbReach_{\mathsf{PS}}(s, F) \leq \mathsf{P}(\mathsf{s}, \Diamond\, \phi_{\mathsf{F}}),$$

where ϕ_F is a formula characterizing the set of states F.

4 Encoding of a Reachability Graph in PRISM

The reachability graph obtained with KRONOS is a list of symbolic states and transitions between them. In order to model-check probabilistic properties we must encode it as a Markov decision process using PRISM's description language, a simple, state-based language similar to Reactive Modules [4].

The behaviour of a Markov decision process is described in this language by a set of *guarded commands* of the form:

```
[] <guard> -> <command>;
```

where *guard* is a predicate over variables of the system, and *command* describes a transition that the system can make when the guard is true. A transition updates the value of the variables, by giving their new *primed* value with respect to their old *unprimed* value. We consider two types of encoding of a reachability graph in this language.

4.1 Explicit Encoding

The first solution is to use an *explicit encoding* of the reachability graph with a single variable s whose value is the index of the symbolic state of the reachability graph. Transitions are simply encoded by guarded commands such that the guard tests the value of s and the command updates it according to the transition relation of the reachability graph.

For example, the encoding of the outgoing probabilistic transitions from symbolic states $0, 4$ and 7, corresponding to location send in the reachability graph of Figure 2, is:

```
[] (s=0) -> 0.99:(s'=1) + 0.01:(s'=2);
[] (s=4) -> 0.99:(s'=5) + 0.01:(s'=6);
[] (s=7) -> 0.99:(s'=8) + 0.01:(s'=9);
```

and the incoming transitions to symbolic state 3, corresponding to location error_before are encoded as:

```
[] (s=2) -> 1:(s'=3);
[] (s=6) -> 1:(s'=3);
```

This encoding generates a description file whose size is, in number of command lines, the size of the transition relation of the reachability graph, which can grow drastically as the complexity of the system increases. PRISM involves a model construction phase, during which the

system description is parsed and an MTBDD [10,5] representing the transition relation is built. When the input file is not a modular description of a system, such as the file generated with the explicit encoding, this phase can be extremely time consuming. An encoding allowing for a more compact description of the system is needed.

4.2 Instances Encoding

Symbolic states in the reachability graph correspond to several *instances* of locations of the timed automaton from which it was generated, with different time constraints. We can then encode them with two variables: a *location* variable l and an *instance* variable n describing to which instance of the location it corresponds.

Let $l = 0$, 1, 2 and 3 be the values corresponding to locations send, wait, error, and error_before in Figure 2, respectively. Then, symbolic states 0, 4 and 7 correspond to three different instances of location send, say $n = 0$, 1 and 2. The outgoing probabilistic transitions from these states can be specified by the guarded commands:

```
[] (l=0)&(n=0) -> 0.99:(l'=1)&(n'=0)
                + 0.01:(l'=2)&(n'=0);
[] (l=0)&(n=1) -> 0.99:(l'=1)&(n'=1)
                + 0.01:(l'=2)&(n'=1);
[] (l=0)&(n=2) -> 0.99:(l'=1)&(n'=2)
                + 0.01:(l'=2)&(n'=2);
```

Similarly, symbolic state 3 is the unique instance of location error_before, encoded as $l = 3$ and $n = 0$. The incoming transitions to this state are described by:

```
[] (l=2)&(n=0) -> 1:(l'=3)&(n'=0)
[] (l=2)&(n=1) -> 1:(l'=3)&(n'=0)
```

Instances are computed by a breadth-first traversal of the reachability graph.

4.2.1 Relative compaction

Note that, in the commands representing the outgoing transitions from location send, the instance variable n is left unchanged, meaning that the transition only affects the location variable for instances 0, 1 and 2. This is equivalent to write that $n' = n$, which can be omitted since, by default, a non updated variable takes its old value. Thus, the transitions above have the same update command, and can then be described more compactly in a single command line:

```
[] (l=0)&(n=0|n=1|n=2) -> 0.99:(l'=1)
                        + 0.01:(l'=2);
```

In a reachability graph, a transition between two given locations is usually repeated several times for different instances of the locations. This encoding allows us to specify them all in a single command line.

We will refer to this as the *relative compaction*, because it is based on specifying the updated value n' relative to its old value n.

4.2.2 Absolute compaction

The previous compaction does not apply in the case of the two incoming transitions to error_before. However, if we specify the updated value n' with its absolute value, the update command of both transitions is the same. Thus, they can both be described more compactly in a single command line:

```
[] (l=2)&(n=0|n=1) -> 1:(l'=3)&(n'=0)
```

In a reachability graph, we encounter states which are the destination of many different transitions, such as the state encoded by $l = 3$ and $n = 0$ in the example of Figure 2. This encoding allows us to specify them all in a single command line.

We will refer to this as the *absolute compaction*, because it is based on specifying the absolute value of n'. Note that this compaction could also be applied to the explicit encoding. However, since in practice the relative compaction leads to a more compact description, compaction algorithms have only been implemented in the case of the instances encoding. Absolute compaction is especially interesting when used in combination with the relative one.

4.2.3 Combination

In order to obtain a further reduction, we can combine both compactions. Since in practice there are potentially more transitions to be compacted with the relative encoding than with the absolute one, the heuristic implemented consists in first applying the relative compaction and then, for those transitions that weren't compacted, i.e. those whose guard correspond to a unique source state, to change the command updates for the instance variable n from relative to absolute, and then apply the absolute compaction.

4.2.4 Algorithms

The compaction algorithms are based on a traversal of the set of transitions of the reachability graph in order to find those which correspond to the same update command, and then describe them in a single line as a transition from multiple sates.

Roughly speaking, the algorithm keeps a set of update commands as pairs (d, k), with $d \in \mathbf{N}$ and $k \in \mathbf{Z}$, corresponding to an update $(l' = d)\&(n' = n+k)$ in the case of a relative encoding, and to $(l' = d)\&(n' = k)$ in the case of an absolute encoding. It then associates a list of source states to every update command.

Furthermore, in order to improve the model building phase, the algorithm detects when different source states correspond to the same location and successive numbers of instance. The corresponding guard is then a constraint specifying that the value of n is between two bounds. The set of transitions considered in the examples above are then specified by:

```
[] (l=0)&(n=0..2) -> 0.99:(l'=1)
                   + 0.01:(l'=2);
[] (l=2)&(n=0..1) -> 1:(l'=3)&(n'=0);
```

5 Verification of the Root Contention Protocol

The IEEE 1394 High Performance serial bus is used to transport digitized video and audio signals within a network of multimedia systems and devices, such as TVs, PCs and VCRs. It has a scalable architecture, and it is hot-pluggable, meaning that devices can be added or removed from the network at any time, supports both isochronous and asynchronous communication and allows quick, reliable and inexpensive data transfer. It is currently one of the standard protocols for interconnecting multimedia equipment. The system uses a number of different protocols for different tasks, including a leader election protocol, called *tree identify protocol*.

The tree identify protocol is a leader election protocol which takes place after a bus reset in the network, i.e. when a node (device or peripheral) is added to, or removed from, the network. After a bus reset, all nodes in the network have equal status, and know only to which nodes they are directly connected, so a leader must then be chosen. The aim of this protocol is to check whether the network topology is a tree and, if so, to construct a spanning tree over the network whose root is the leader elected by the protocol.

In order to elect a leader, nodes exchange "be my parent" requests with its neighbours. However, *contention* may arise when two nodes simultaneously send "be my parent" requests to each other. The solution adopted by the standard to overcome this conflict, called *root contention*, is both probabilistic and timed: each node will flip a coin in order to decide whether to wait for a short or for a long time for a request. The property of interest of the protocol is whether a leader is elected before a certain deadline, with a certain probability or greater.

5.1 The Probabilistic Timed Automata Models

The models presented here are based on the classical timed automata models of [34]. Figure 3 shows $\mathtt{Node}_i^{\mathtt{P}}$, the probabilistic timed automaton for a contending node of the network involved in the root contention protocol.

The probabilistic timed automaton $\mathtt{Node}_i^{\mathtt{P}}$ is a probabilistic extension of the classical timed automaton node model from [34]. The usual conventions for the graphical representation of classical timed automata are used. The edges leaving the locations root_contention (the initial location, as denoted by the bold node) and rec_idle, correspond to probabilistic transitions. For example, the left-hand edges leaving root_contention correspond to a probabilistic choice of taking a transition to either of the target locations, rec_req_fast and rec_req_slow, each with probability 0.5, while resetting the clock x_i. For simplicity, we omit the probability labels from edges corresponding to probability 1. Urgent locations are indicated by the dashed locations. The communication medium between the nodes, which assumes that signals are driven continuously across wires which comprise of two-place buffers, is then represented by the models $\mathtt{Wire}_i$, for $i \in \{1, 2\}$ (Figure 4), adopted directly from [34]. The full model of the protocol is defined as the parallel composition

$$\mathtt{Impl}_1^p = \mathtt{Node}_1^{\mathtt{P}} \| \mathtt{Wire}_1 \| \mathtt{Wire}_2 \| \mathtt{Node}_2^{\mathtt{P}}$$

using Definition 2.

We also study the abstract probabilistic timed automaton $\mathtt{I}_1^p$ of the root contention protocol given in Figure 5. It is a probabilistic extension of the classical timed automaton $\mathtt{I}_1$ of [34] where each instance of bifurcating edges corresponds to a coin being flipped. For example, in the initial location start_start, there is a nondeterministic choice corresponding to node 1 (resp. node 2) starting the root contention protocol and flipping its coin, leading with probability 0.5 to each of slow_start and fast_start (resp. start_slow and start_fast). For simplicity, probability labels are omitted from the figure and probabilistic edges are represented by dashed arrows.

The probabilistic timed automaton $\mathtt{I}_1^p$ represents an *abstraction* of the root contention protocol, in the sense that it may exhibit a superset of adversaries of the more refined protocol model $\mathtt{Impl}_1^p$. However, similarly to the results presented in [29], the probabilities computed for the verification of $\mathtt{I}_1^p$ and $\mathtt{Impl}_1^p$ agree for all the deadlines considered.

The timing constraints are derived from those given in the IEEE 1394a standard when the communication delay between the nodes is set to 360 ns, which represents the assumption that the contending nodes are separated by a distance close to the maximum required for the correctness of the protocol (from the analysis of [34]). Note that in the abstract model (Figure 5) the timing constant for the enabling condition of the edge from fast_fast to done is obtained from 760 ns, the minimal waiting time if the "fast" side of the coin is obtained, minus 360 ns, the wire propagation delay; similarly, the enabling conditions of the other edges to done are obtained from 1590 ns, the minimal waiting time if the "slow" side of the coin is obtained, minus 360 ns. In Section 5.5 we will investigate the effect of changing the communication delay.

5.2 Verification Method

In this section we outline our approach of using KRONOS and PRISM to verify deadline properties of the tree identify protocol of the IEEE 1394 High Performance serial bus. In particular we calculate the probability of electing a leader before a certain deadline D for both the full

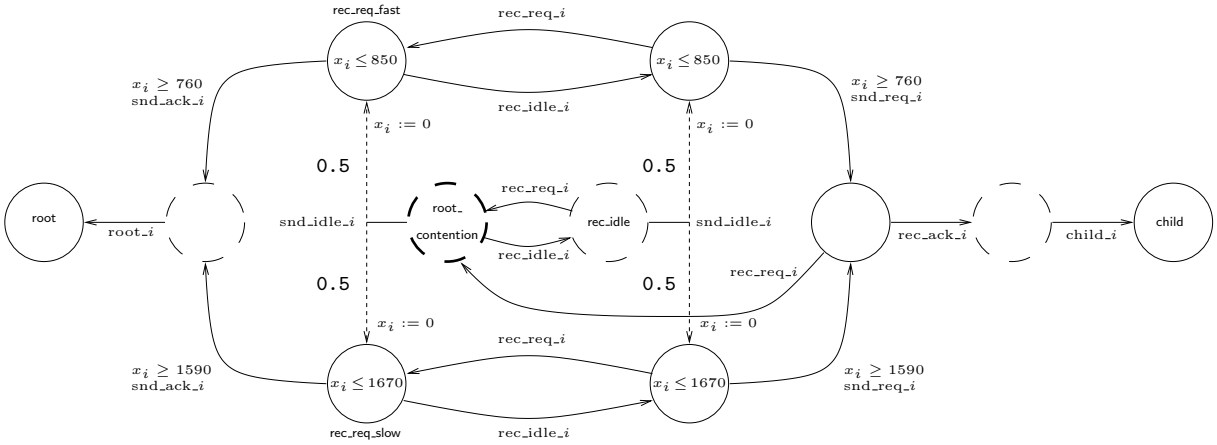

Fig. 3. The probabilistic timed automaton $\texttt{Node}_i^{\texttt{p}}$.

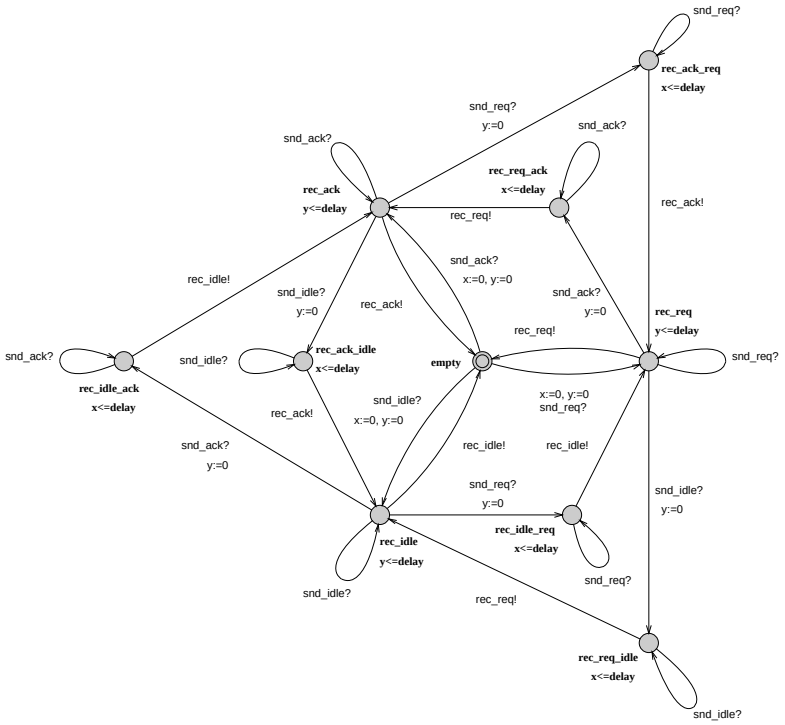

Fig. 4. The timed automaton template for $\texttt{Wire}_i$.

model $\texttt{Impl}_1^p$ and the abstract model $\texttt{I}_1^p$. For both models the first step is to construct the reachability graph of the probabilistic timed automaton $\texttt{I}_1^p$ until a deadline D is exceeded. To do this, we add an additional clock y, which measures the time elapsed since the beginning of the execution, and, upon entering a location where a leader is elected, we immediately check whether the clock y satisfies the deadline and then force the system to move to distinct locations depending on whether the deadline is satisfied by y or not.

For example, in the case of the abstract model $\texttt{I}_1^p$, upon entering the location done, we test in *time zero*, by adding an invariant $x = 0$ to this location and resetting the clock x on all incoming edges (this invariant then forces the system to leave the location immediately) whether the clock y exceeds this deadline or not, by adding two outgoing edges from done, one with the guard $y \geq D$ leading to a location done_after and one with the guard $y < D$ leading to a location done_before.

Next, we specify the property of the root contention protocol we are interested in, namely, that a leader is elected *before* the deadline D with *at least* a given probability λ. The PCTL formula that specifies this property is of the form $\mathcal{P}_{\geq\lambda}(\Diamond\,(\mathsf{leader\_elected} \wedge y < D))$, which

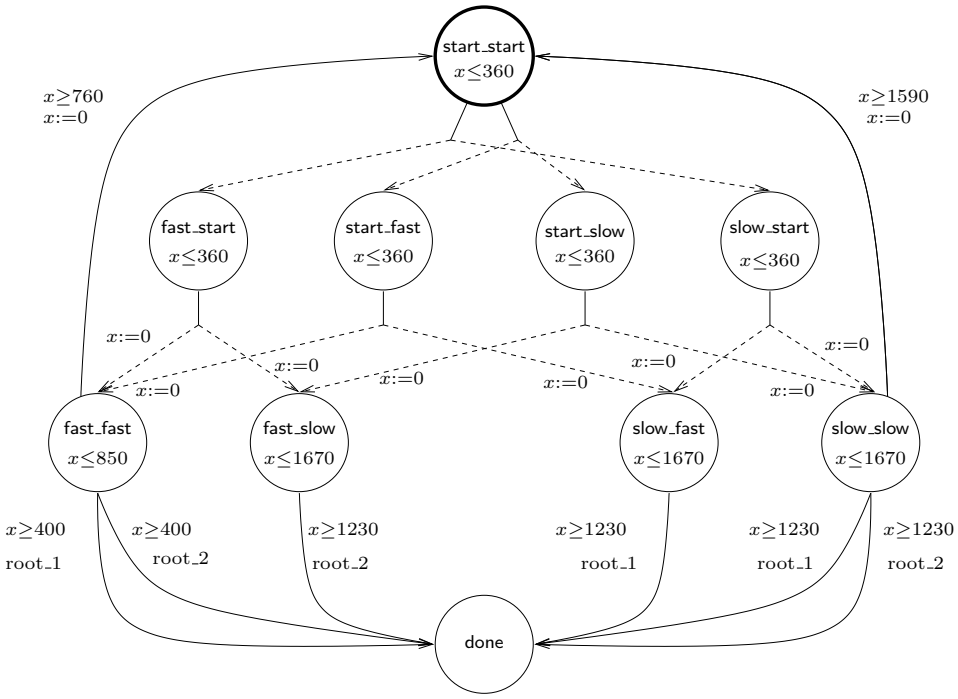

Fig. 5. The probabilistic timed automaton $\mathtt{I}_1^p$.

cannot be verified with our technique because the probabilistic quantifier $\mathcal{P}_{\geq\lambda}$ is not of the correct form. However, it can be shown [29] that it is equivalent to the formula $\mathcal{P}_{<1-\lambda}(\Diamond\,(\mathsf{leader\_elected} \wedge y \geq D)))$ which can actually be verified on the reachability graph. For example, in the case of the abstract model this corresponds to checking the formula $\mathcal{P}_{<1-\lambda}(\Diamond\,\mathsf{done\_after})$.

More precisely, we compute $\mathsf{P}(\mathsf{s}, \Diamond\,(\mathsf{leader\_elected} \wedge \mathsf{y} \geq \mathsf{D}))$, the *maximal* probability for electing a leader *after* the deadline on the forwards reachability graph, and then, from above, $\mathsf{p}(\mathsf{s}, \Diamond\,(\mathsf{leader\_elected} \wedge \mathsf{y} < \mathsf{D}))$, the *minimal* probability of electing a leader *before* the deadline, equals 1 minus this computed probability. Note that, as explained in Section 2.3, since using forwards reachability yields only upper bounds on the actual maximal reachability probability, the computed minimal reachability probability is a lower bound on the actual minimal reachability probability. However, the results generated with this forwards reachability approach for both the full model $\mathtt{Impl}_1^p$ and abstract model $\mathtt{I}_1^p$ agree with the exact results presented in [29].

5.3 Experimental Results

The deadlines D we consider range up to $10^5 ns$ and, unless otherwise stated, the wire delay is set to 360 ns. These experiments were performed on a PC running Linux, with a 1400 MHz processor and 512 MB of RAM. PRISM was used with its default options. Additional information can be found in [32].

Table 1 and Table 2 show the results concerning the generation with KRONOS of the reachability graph and

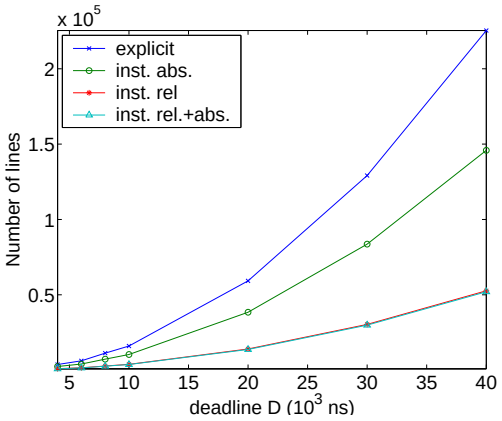

Fig. 6. Number of lines of the MDP for the full protocol $\mathtt{Impl}_1^p$

of its encoding as an MDP for the full model $\mathtt{Impl}_1^p$ and abstract model $\mathtt{I}_1^p$. In both tables, the first two columns give information about the generation of the reachability graph, its size in terms of the number of *states* and the *time* in seconds needed to generate it. The remaining columns show the size, in number of lines (i.e. transitions), of the MDP file generated by KRONOS, for the different encodings we considered: explicit, instances with either absolute or relative compaction, and with both of them.

Figure 6 and Figure 7 show the evolution of the number of command lines of the generated file for different values of the deadline for the full protocol $\mathtt{Impl}_1^p$ and abstract model $\mathtt{I}_1^p$ respectively. The graphs demonstrate that the instances encoding allows for compactions which reduce drastically the number of lines of the MDP file.

deadline (10^3 ns)	forw. reach. states	forw. reach. time (s)	explicit	instances abs	instances rel	instances rel+abs
4	2599	0.940	3716	2424	974	894
6	4337	1.64	6202	4050	1545	1473
8	7831	2.93	11262	7320	2748	2622
10	11119	4.27	15986	10398	3864	3710
20	41017	18.7	59254	38406	14062	13730
30	89283	56.1	129154	83634	30349	29843
40	155675	129	225420	145854	52681	52019

Table 1. Generation and encoding of the reachability graph for the full model $\texttt{Impl}_1^p$

deadline (10^3 ns)	forw. reach. states	forw. reach. time (s)	explicit	instances abs	instances rel	instances rel+abs
4	131	0.00	174	104	42	26
6	216	0.01	290	173	64	27
8	372	0.02	499	297	91	36
10	526	0.03	709	421	126	39
20	1876	0.09	2531	1501	368	72
30	4049	0.20	5466	3240	734	100
40	7034	0.46	9499	5629	1223	126
50	10865	1.23	14674	8694	1842	159
60	15511	2.74	20952	12412	2586	186
80	27296	8.94	36868	21841	4437	243
100	42401	22.29	57274	33926	6797	303

Table 2. Generation and encoding of the reachability graph for the abstract model $\texttt{I}_1^p$

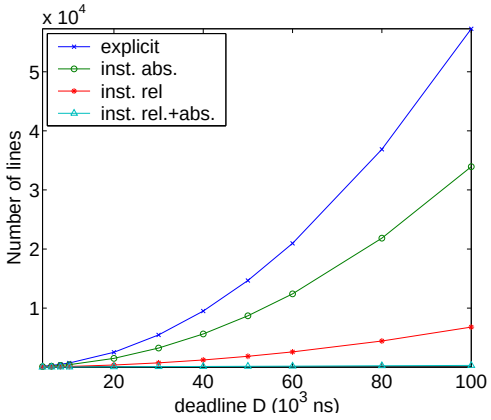

Fig. 7. Number of lines of the MDP for the abstract model $\texttt{I}_1^p$

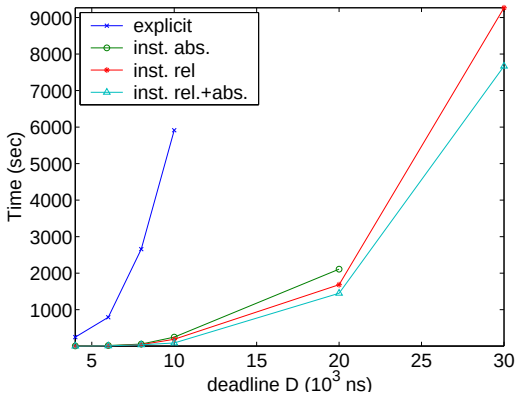

Fig. 8. Time to build the full protocol $\texttt{Impl}_1^p$

In the case of the abstract model $\texttt{I}_1^p$, when both relative and absolute compactions are considered, the number of lines grows less than linearly on the value of the deadline. However, there is not such a drastic improvement in the case of the more complex model $\texttt{Impl}_1^p$. Currently, the compaction algorithms do not take into account the fact that this model is built through the parallel composition of subcomponents which may explain why the compaction methods are not as efficient in this case.

The experimental results concerning the verification with PRISM for the full protocol $\texttt{Impl}_1^p$ and abstract model $\texttt{I}_1^p$ are shown in Table 3 and Table 4 respectively. The left-most column shows the deadline used in the property, and the right-most column shows the minimal probability with which the system has reached a state where a leader is elected before the deadline. The results reflect the obvious fact that increasing the deadline increases the probability of a leader being elected. Notice that the probabilities is computed for deadlines of more than 80000 ns, have value 1, the meaning that the probabilities equal one when rounded up to 8 decimal places *not* that the probability equals 1.

The remaining columns give information on the time performance of PRISM in seconds, to build the model (columns labelled "model") and to compute the probability (columns labelled "verif"), using the *explicit* encoding and the *instances* encoding with relative compaction (inst+rel) and with relative and absolute compaction (inst+rel+abs) .

Compared to the previous attempt of verifying the root contention protocol using forwards reachability [29], an approach which uses HYTECH [23], the generation of the reachability graph is no longer a problem, since it only took about 20 seconds to generate the forwards reachability graph of the abstract model $\texttt{I}_1^p$ for a deadline of 100000 ns, whilst it took approximately 24 hours to generate it with HYTECH for a deadline of 6,000 ns. Moreover, model checking of the probabilistic property in the case of the abstract model took less than one second in the worst case.

deadline (10^3 ns)	explicit		inst+rel		inst+rel+abs		probability
	model (s)	verif (s)	model (s)	verif (s)	model (s)	verif (s)	
4	249	0.461	5.52	0.109	4.76	0.111	0.62500000
6	792	0.709	13.1	0.217	10.8	0.158	0.85156250
8	2657	1.31	43.4	0.272	37.9	0.229	0.93945313
10	5915	9.76	189	0.719	89.7	0.364	0.97473145
20	–	–	1684	1.48	1450	1.39	0.99962956
30	–	–	9268	5.07	7669	4.29	0.99999445
40	–	–	30977	14.9	27870	20.3	0.99999991

Table 3. Time performances for model building and verification of the full model $\texttt{Impl}_1^p$

deadline (10^3 ns)	explicit		inst+rel		inst+rel+abs		probability
	model (s)	verif (s)	model (s)	verif (s)	model (s)	verif (s)	
4	0.497	0.020	0.062	0.010	0.051	0.001	0.625000000
6	1.21	0.025	0.094	0.017	0.058	0.018	0.851562500
8	4.00	0.035	0.157	0.019	0.091	0.023	0.939453135
10	9.31	0.051	0.244	0.020	0.108	0.020	0.974731455
20	131	0.158	2.01	0.042	0.466	0.043	0.999629565
30	778	0.383	9.06	0.088	1.35	0.089	0.999994454
40	2445	0.554	28.9	0.151	3.30	0.151	0.999999919
50	–	–	78.2	0.239	7.32	0.231	0.999999998
60	–	–	151	0.334	13.9	0.343	0.999999999
80	–	–	555	0.604	37.3	0.606	1.000000000
100	–	–	1449	1.00	90.6	0.963	1.000000000

Table 4. Time performances for model building and verification of the abstract model $\texttt{I}_1^p$

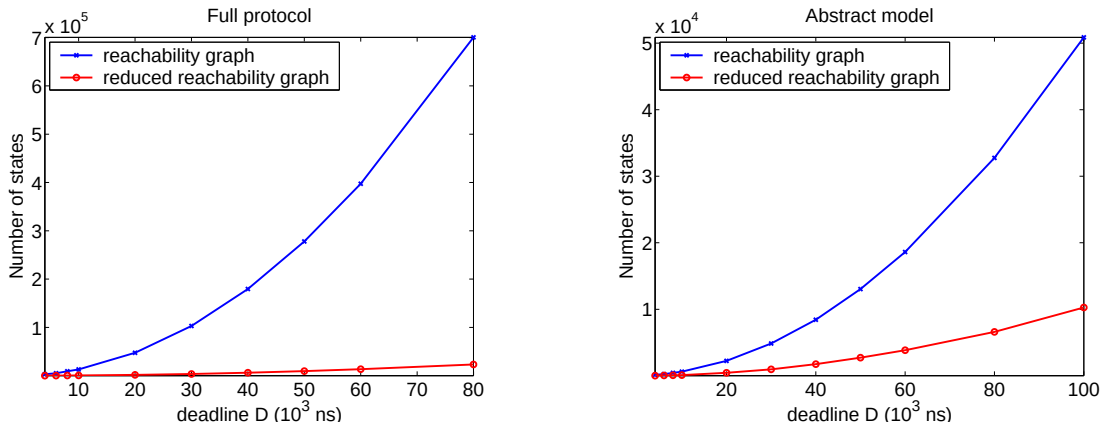

Fig. 10. State space reduction through bisimulation quotient for both models

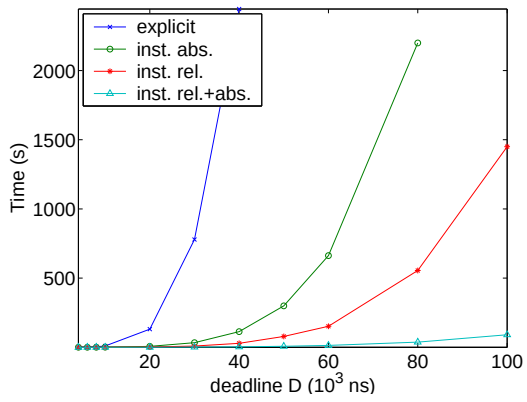

Fig. 9. Time to build the abstract model $\texttt{I}_1^p$

Figure 8 and Figure 9 shows the evolution of the time needed to build the model for different deadlines using different encodings for the full protocol and abstract model respectively. We can see that the compactions also improve the time required to build the model in PRISM. We note the improvements in the case of the abstract model $\texttt{I}_1^p$ over the result presented in [13] (the full model $\texttt{Impl}_1^p$ was not considered there), where the time to build the model in PRISM grew drastically as the value of the deadline increased, even though the size of the input file grows linearly. This was due to the complexity of the guards after compaction. However, the model building phase of PRISM has since been optimized, leading to an order of magnitude decrease in construction times.

5.4 Probabilistic Bisimulation

The results presented in Figure 8 and Table 3 demonstrate that, even after applying the compaction techniques of Section 4, the main obstacle to verifying the full protocol $\texttt{Impl}_1^p$ against large deadlines is the time required by PRISM to build the model. In this section we consider an approach to overcome this problem, by first

reducing the forwards reachability graph to its (strong) probabilistic bisimulation quotient and then constructing the PRISM model. This is similar to the approach in [11,12], where probabilistic systems are reduced with respect to a probabilistic simulation relation, which is refined until the probabilities are accurate enough, yielding in the worst case the probabilistic bisimulation quotient.

We use the CADP (*Ceasar/Aldebaran Development Package*) [20] a tool set for the design and verification of complex systems, which has recently be extended to allow for performance evaluation [19]. In particular, we use the BCG_MIN tool [8] which supports the minimization of probabilistic systems with respect to probabilistic bisimulation. The main steps in this approach are: represent the reachability graph in a format suitable for the CADP toolset, use CADP (in particular BCG_MIN) to construct the bisimulation quotient, and finally translate the CADP output into the PRISM language.

To construct the input to CADP required only a straightforward modification [4] of the output from KRONOS. For the quotient system to preserve the maximal reachability probability of interest, we must identify the states where a leader has been elected after the deadline has passed. For simplicity, since these are the only states that we need to identify, we label all the transitions leaving such states with one action and then label all remaining transitions with another distinct action.

Using this version of the reachability graph, we then use the toolset CADP to construct the quotient under probabilistic bisimulation which preserves the maximal probability of electing a leader after the deadline has passed.

To translate the CADP output into the PRISM language we wrote a simple translator which takes as input the probabilistic system representing the quotient system (the output from CADP) and translates this into the PRISM language. We note that at this stage we have lost all information concerning which locations and zones correspond to which states, and hence we are restricted to an explicit encoding of the quotient system.

The results obtained with CADP for both the full protocol $\texttt{Impl}_1^p$ and abstract model $\texttt{I}_1^p$ are given in Figure 10. As can be seen from the graphs of Figure 10, there is a significant state-space reduction in both cases, in particular when considering the full protocol. The improved reduction for the full protocol can be expected, since the model $\texttt{I}_1^p$ is already an abstraction, and hence there is less reduction possible by applying the bisimulation quotient.

In Table 5 we give the experimental results concerning the verification with PRISM on the reduced reachability graph for different deadlines and for both the full protocol $\texttt{Impl}_1^p$ and abstract model $\texttt{I}_1^p$. The results demonstrate that, similarly to the compaction based approach, the majority of the time required by PRISM is with respect to model construction and not verification. Note that, as expected, the probabilities match those obtained through the compaction approach. In Figure 11 we compare the time required to construct the models in PRISM with the time taken when using the most efficient compaction technique (instances encoding with both relative and absolute compactions). In the case of the full protocol $\texttt{Impl}_1^p$, because there is such a significant reduction in the state space, even though we can only use an explicit encoding, the time taken to construct the model in PRISM is faster than when applying any of the compaction algorithms. On the other hand, in the case of the abstract model $\texttt{I}_1^p$, the results show that using the compaction algorithms on the (unreduced) reachability graph is more efficient than using the explicit encoding of the reduced reachability graph. This can be seen as both a result of the efficiency of the compaction algorithms in this case and the less signification reduction in the state space when using probabilistic bisimulation in comparison with the reduction for the full protocol.

5.5 RCP under wires of different lengths

In this section we report on the results obtained as the communication delay between the nodes varies, which corresponds to running the protocol with wires of different lengths connecting the nodes. The results obtained are presented in Figure 12. As expected, as the communication delay between the nodes increases, the probability of electing the root before a deadline decreases.

The statistics obtained when considering different wire delays, for both the full protocol and abstract model, with regards to both the compaction algorithms and the reduction by probabilistic bisimulation follow a similar pattern as those presented above (where the communication delay equals 360ns). In particular, the most efficient approach for the full protocol is to use the reduced reachability graph with the explicit encoding, and, in the case of the abstract protocol, to apply the instances encoding with both the relative and absolute compaction on the unreduced reachability graph.

5.6 RCP with a biased coin

We now study the influence of using a biased coin on the performance of the protocol. As conjectured in [35], a curious property of the protocol is that the probability for electing a leader before a deadline can be slightly increased if the probability to choose fast timing increases for both nodes.

Note that we do not need to recompute the forward reachability graph in each case. Instead, since probabilities for choosing a fast or slow timings can be given as parameters in PRISM description language, the same input file is used to perform probabilistic model checking, and only the actual values of the probabilities change.

[4] The main step is converting the reachability graph to the alternating model [21].

deadline (10^3 ns)	full protocol $\texttt{Impl}_1^p$ model (s)	verif (s)	abstract model $\texttt{I}_1^p$ model (s)	verif (s)	probability
4	0.268	0.054	0.092	0.039	0.625000000
6	0.711	0.076	0.167	0.048	0.851562500
8	1.85	0.124	0.208	0.045	0.939453135
10	5.08	0.180	0.339	0.093	0.974731455
20	81.8	0.709	5.79	0.129	0.999629565
30	389	1.94	27.7	0.333	0.999994454
40	1305	4.11	111	0.655	0.999999919
50	3451	6.17	284	1.087	0.999999998
60	7172	13.0	554	1.379	0.999999999
80	23320	30.5	1802	2.771	1.000000000
100	–	–	5181	6.05	1.000000000

Table 5. Time performances for model building and verification for both models after reduction

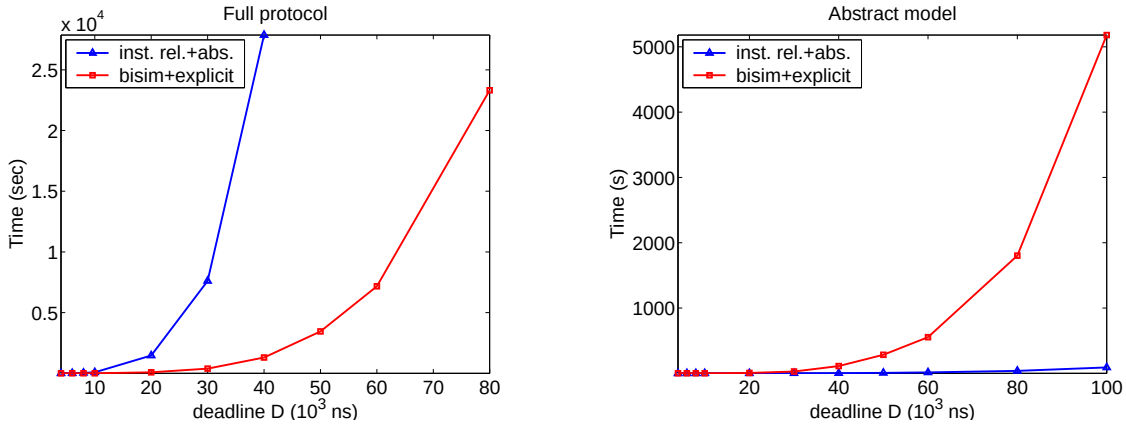

Fig. 11. Comparison of construction time for compaction based and bisimulation based approaches

fast	*slow*	$D = 3000$	$D = 4000$	$D = 6000$	$D = 8000$	$D = 10000$
0.01	0.99	0.019800	0.019803	0.039211	0.058237	0.076886
0.10	0.90	0.180000	0.181800	0.330534	0.452219	0.551777
0.20	0.80	0.320000	0.332800	0.554516	0.702353	0.801006
0.30	0.70	0.420000	0.457800	0.704352	0.838050	0.910958
0.40	0.60	0.480000	0.556800	0.799150	0.907635	0.957090
0.45	0.55	0.495000	0.595238	0.830027	0.927066	0.968234
0.50	0.50	0.500000	0.625000	0.851562	0.939453	0.974731
0.55	0.45	0.495000	0.644738	0.864616	0.946095	0.977772
0.60	0.40	0.480000	0.652800	0.869498	0.947313	0.977795
0.65	0.35	0.455000	0.647238	0.865609	0.942253	0.974559
0.70	0.30	0.420000	0.625800	0.850898	0.928530	0.966912
0.80	0.20	0.320000	0.524800	0.768942	0.853275	0.923035
0.90	0.10	0.180000	0.325800	0.544273	0.629189	0.746829
0.99	0.01	0.019800	0.039206	0.076872	0.095156	0.130622

Table 6. Probability of leader election with a biased coin.

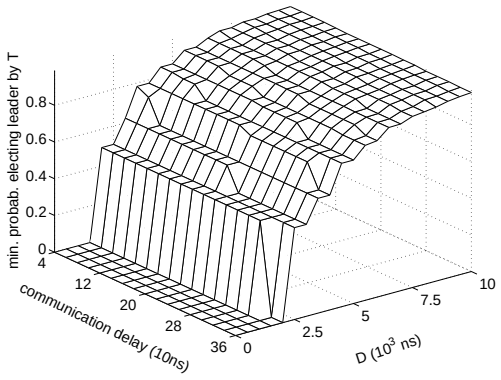

Fig. 12. Verification results as communication delay varies.

Table 6 gives the probability for electing a leader for deadlines between 3000ns and 10000ns, when using different biased coins (coins which return different probability of choosing the *fast* and *slow* timing). We suppose that the nodes in contention have the same biased coins. Although it is possible to improve the performance further by supposing that nodes have different biased coins (one node has a coin biased towards fast while the other's coin is biased towards slow), this is not feasible in practice as each node follows the same procedure and it is not known in advance which nodes of the network will take part in the root contention protocol. Furthermore, to decide before entering the protocol, which node should flip what sort of coin is equivalent to electing a root.

The results demonstrate that the (timing) performance of the root contention protocol can be improved

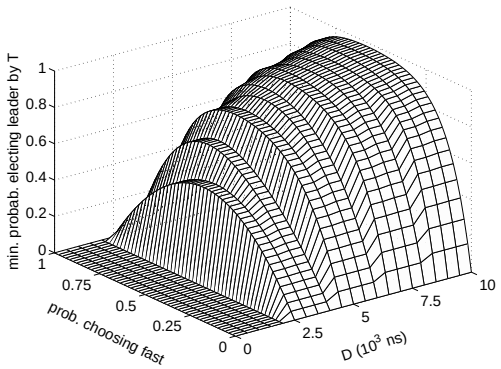

Fig. 13. Verification results with a biased coin.

using a biased coin which has a higher probability of flipping "fast". This curious result is possible because, although using such a biased coin decreases the likelihood of the nodes flipping different values, when nodes flip the same values there is a greater chance (i.e. when both flip "fast") that less time elapses before they flip again. There is a compromise here though: as the coin becomes more biased towards "fast", the probability of the nodes actually flipping different values (which is required for a leader to be elected) decreases, even though the delay between coin flips will on average decrease. This decrease in probability is demonstrated in Table 6 and Figure 13, where it is shown that increasing the probability of flipping "fast" eventually leads to a decrease in the probability of electing a leader by any given deadline.

We also considered the effect of using a biased coin as the communication delay between the nodes varies. The results showed that, for shorter communication delays (wire lengths), there is a greater advantage in using a biased coin (for shorter wire length the maximum probability for a fixed deadline occurs for a coin which has a greater bias towards "fast"). The reason this happens is that, for the short wire length, there is a greater saving in time when both nodes flip "fast" than for a longer wire length, since the time required when both nodes flip "fast" is a constant delay given in the protocol plus a delay which is dependent on the wire length. For details of computed probabilities in this case see the PRISM web page [32].

6 Conclusions

We have presented an approach to the automatic verification of soft deadlines for timed probabilistic systems modelled as probabilistic timed automata. We use KRONOS to generate the probabilistic reachability graph with respect to the deadline and encode it in the PRISM input language. A probabilistic reachability property is then verified with PRISM. We have successfully applied this verification technique to the timed and probabilistic root contention protocol of the IEEE 1394. We have computed the minimal probability of electing a leader before different deadlines, and studied the influence of using a biased coin, and of varying the wire length, on this minimal probability.

The main obstacle we had to face was the encoding of the reachability graph in the PRISM input language. The model checking algorithms of PRISM are based on (MT)BDDs, so its input needs to be specified in a modular way. An explicit encoding of the reachability graph using a single variable to encode a state turned out to be inadequate even for small values of the deadline. The instances encoding using two variables, one corresponding to the location of the timed automaton, and the other to the instance of this location in the reachability graph, allowed us to apply compaction techniques that helped overcoming this problem in the case of the abstract model.

However, verification of the full model showed that compaction algorithms do not always lead to a substantial reduction. Taking these results into account, a better encoding allowing for a better compaction is under study. Reduction by strong bisimulation proved to be very useful in this case, although it was limited to the use of the explicit encoding. It will be interesting to investigate the use of the instances encoding and the compaction algorithms on the reduced model to obtain a further compaction, and lower the time needed to build the model. These different aspects should be studied by applying this approach to other systems where timing and probabilistic aspects arise.

Acknowledgments

We would like to thank Sergio Yovine for providing us the libraries of KRONOS, Dave Parker for improving the model-building algorithms of PRISM and the referees for their useful suggestions.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In J. L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *ICALP'91*, volume 510 of *Lecture Notes in Computer Science*. Springer, 1991.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
5. I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E.Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams

and their applications. In *Proc. International Conference on Computer-Aided Design (ICCAD'93)*, pages 188–191, 1993. Also available in *Formal Methods in System Design*, 10(2/3):171–206, 1997.

6. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In E. Emerson and A. Sistla, editors, *Proc. 12th Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *LNCS*, pages 358–372. Springer, 2000.
7. C. Baier and M. Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
8. BCG_MIN manual page. `http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html`.
9. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.
10. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. International Workshop on Logic Synthesis (IWLS'93)*, pages 1–15, 1993. Also available in *Formal Methods in System Design*, 10(2/3):149–169, 1997.
11. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proceedings of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2001,* Aachen, Germany, volume 2165 of *Lecture Notes in Computer Science*, pages 29–56. Springer-Verlag, 2001.
12. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reduction and refinement strategies for probabilistic analysis. In H. Hermanns and R. Segala, editors, *Proceedings of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2002,* Copenhagen, Denmark, Lecture Notes in Computer Science. Springer-Verlag, 2002.
13. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. In *Proc. 7th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'02)*, volume 66(2) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.
14. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer-Verlag, 1996.
15. C. Daws and S. Tripakis. Model–checking of real–time reachability properties using abstractions. In B. Steffen, editor, *Proc. Tools and Algorithms for Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 313–329. Springer-Verlag, 1998.
16. C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In A. Burns, Y.-H. Lee, and K. Ramamritham, editors, *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 66–75. IEEE Computer Society Press, 1995.
17. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In J. Baeten and S. Mauw, editors, *Proc. CONCUR '99: Concurrency Theory*, volume 1664 of *LNCS*, pages 66–81. Springer Verlag, 1999.
18. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.
19. H. Garavel and H. Hermanns. On combining functional verification and performance evaluation using CADP. In L. Eriksson and P. Lindsay, editors, *FME 2002: International Symposium of Formal Methods Europe*, volume 2391 of *LNCS*, pages 410–429. Springer, 2002.
20. H. Garavel, F. Lang, and R. Mateescu. An overview of CADP. Technical Report RT-254, INRIA, 2001.
21. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
22. T. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HYTECH. In E. Brinksma, W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, volume 1019 of *LNCS*, pages 41–71. Springer-Verlag, 1995.
23. T. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1+2):110–122, 1997.
24. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Graduate Texts in Mathematics. Springer, 2nd edition, 1976.
25. KRONOS web page. `http://www-verimag.imag.fr/TEMPORISE/kronos/`.
26. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In J. B. T. Field, P. Harrison and U. Harder, editors, *Proc. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.
27. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, *Proc. Tools and Algorithms for Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.
28. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002. A preliminary version of this paper appeared in *Proc. ARTS'99*, volume 1601 of LNCS, pages 75–95, 1999.
29. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Special Issue of Formal Aspects of Computing*, 2002. To appear.

30. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.
31. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991. Preliminary version of this paper appeared in Proc. 16th Annual ACM Symposium on Principles of Programming Languages, pages 134-352, 1989.
32. PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism/`.
33. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
34. D. Simons and M. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k. *Springer International Journal of Software Tools for Technology Transfer*, 3(4):469–485, 2001.
35. M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, Univerisity of Nijmegen, 2002.
36. M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. Symposium on Foundations of Computer Science (FOCS'85)*. IEEE Computer Society Press, 1985.