Electronic Notes in Theoretical Computer Science 66 No. 2 (2002)
URL: `http://www.elsevier.nl/locate/entcs/volume66.html` 16 pages

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

Conrado Daws [a,1], Marta Kwiatkowska [b], Gethin Norman [b]

[a] *CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands*
`Conrado.Daws@cwi.nl`

[b] *University of Birmingham, Birmingham B15 2TT, United Kingdom*
`{M.Z.Kwiatkowska,G.Norman}@cs.bham.ac.uk`

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 probabiliy of a leader being elected before a deadline, for different deadlines, and study the influence of using a biased coin on this minimal probability.

Key words: model checking, soft deadlines, probabilistic timed automata, IEEE 1394, root contention protocol

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.

⋆ Supported in part by the EPSRC grant GR/N22960.
1 Supported by an ERCIM Fellowship.

However, in other cases like 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 [16,17] 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 [7] and Uppaal [18], the direction taken is that of automatic verification through *model checking*, adapting the formalisms and algorithms [1] for model-checking of timed systems to the case of timed probabilistic systems. Within this approach[2], timed probabilistic systems are modelled as *probabilistic timed automata* [16], 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 [16] or the forward reachability graph [17]. 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 [6,5].

In this work we show how, based on these ideas, the real-time model-checker Kronos [7,12] and the probabilistic model-checker Prism [13,19] 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 deadline. We first use Kronos[3] 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, and investigate the influence of using a biased coin 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

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

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

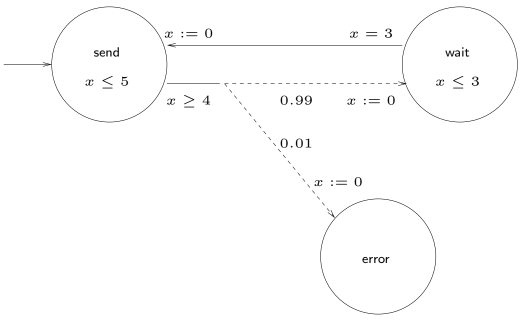

Fig. 1. An example of a probabilistic timed automaton PTA_1.

contention protocol of the IEEE 1394 standard. We conclude with Section 6.

2 Probabilistic Timed Automata

A timed automaton [2] 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 2.1 *The probabilistic timed automaton* PTA_1 *of Figure 1 models a process which repeatedly tries to* send *a packet after between 4 and 5 ms, and if successful* wait*s 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.*

2.1 Syntax

Clocks and valuations. Let the set $\mathcal{X}$ of *clocks* be a set of variables taking 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 $v \in \mathbf{R}_+^{|\mathcal{X}|}$ 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 $\overrightarrow{\zeta}$ 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* over a finite set Q is a function $\mu : Q \to [0,1]$ such that $\sum_{q \in Q} \mu(q) = 1$. Let $\mathsf{Dist}(Q)$ be the set of distributions over subsets of Q.

Definition 2.2 (Probabilistic timed automata.) *A* probabilistic timed automaton *is a tuple* $\mathsf{PTA} = (L, \mathcal{X}, \Sigma, I, \mathrm{P})$ *where: L is a finite set of* locations[4]*;* 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* $p(X, l') > 0$.

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 $v \models I(l)$. 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* $\mathsf{PS} = (S, \mathit{Act}, \mathit{Steps})$ consists of a set S of *states*, a set Act of *actions*, and a *probabilistic transition relation* $\mathit{Steps} \subseteq S \times \mathit{Act} \times \mathsf{Dist}(S)$. 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 $(s, a, \mu) \in \mathit{Steps}$, and then by making a probabilistic choice of target state s' according to μ, such that $\mu(s') > 0$.

Definition 2.3 (Semantics of probabilistic timed automata.) *Given a probabilistic timed automaton* $\mathsf{PTA} = (L, \mathcal{X}, \Sigma, I, \mathrm{P})$*, the* semantics *of* PTA *is the probabilistic system* $[\![\mathsf{PTA}]\!] = (S, \mathit{Act}, \mathit{Steps})$ *defined by the following.*

[4] We sometimes identify an *initial location* $\bar{l} \in L$ represented graphically by an incoming arrow. In this case, the model starts in $\bar{l}$ with all clocks set to 0.

(States) *Let* $S \subseteq L \times \mathbf{R}_+^{|\mathcal{X}|}$ *such that* $(l, v) \in S$ *if and only if* $v \models I(l)$. (Actions) *Let* $Act = \mathbf{R}_+ \cup \Sigma$. (Probabilistic transitions) *Let Steps be the least set of probabilistic transitions containing, for each state* $(l, v) \in S$*:*

Time transitions. *For each duration* $t \in \mathbf{R}_+$*, let* $((l, v), t, \mu) \in Steps$ *if and only if (1)* $\mu(l, v + t) = 1$*, and (2)* $v + t' \models I(l)$ *for all* $0 \leq t' \leq t$*.*

Discrete transitions. *For each probabilistic edge* $(l, g, \sigma, p) \in \mathrm{P}$*, let* $((l, v), \sigma, \mu) \in Steps$ *if and only if (1)* $v \models g$*, and (2) for each state* $(l', v') \in S$*:*

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

2.3 Probabilistic Reachability

The behaviour of a probabilistic timed automaton is described in terms of the behaviour of its semantics, that is, the behaviour of a probabilistic system.

Paths. A *path* of a probabilistic system PS is a non-empty finite or infinite sequence of transitions $\omega = s_0 \xrightarrow{a_0, \mu_0} s_1 \xrightarrow{a_1, \mu_1} \cdots$. 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 $(last(\omega), a, \mu) \in Steps$ [22]. Let Adv_{PS} be the set of adversaries of PS. For any $A \in Adv_{\mathsf{PS}}$, let $Path^A_{fin}$ and $Path^A_{ful}$ denote the set of finite and infinite paths associated with A. A probability measure $Prob^A$ over $Path^A_{fin}$ can then be defined following [11].

Definition 2.4 *Let* $\mathsf{PS} = (S, Act, Steps)$ *be a probabilistic system. Then the* reachability probability *with which a set* $F \subseteq S$ *of target states, can be reached from a state* $s \in S$*, for an adversary* $A \in Adv_{\mathsf{PS}}$*, is:*

$$ProbReach^A(s, F) \stackrel{\text{def}}{=} Prob^A\{\omega \in Path^A_{fin} \mid \omega(0) = s \,\&\, \exists i \in \mathbf{N} \,.\, \omega(i) \in F\} \; .$$

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

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

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

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* [8]. 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.

3.1.1 Edge Successors.

The edge-successor of $\langle l, \zeta \rangle$ with respect to an edge $e = (l, g, X, l')$ is

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

3.1.2 Time Successors.

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

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

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 [8], 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 [19] 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 [6,5] if the model is a DTMC or an MDP, and CSL [4] in the case of a CTMC. We focus on the model checking of reachability properties on MDPs, since a (non-deterministic) 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 the maximal probability $\mathsf{P}(\mathsf{s}, \Diamond \phi)$ with

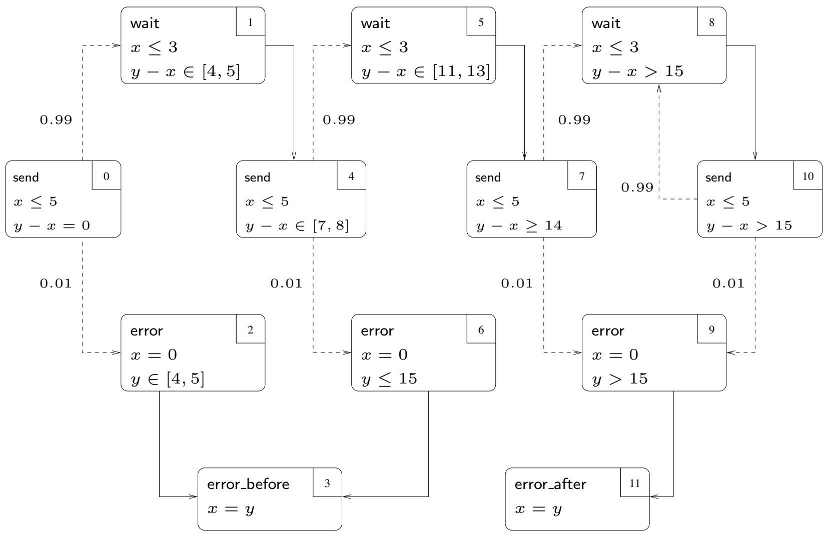

Fig. 2. Reachability graph of PTA_1

which a state s satisfies a reachability formula $\Diamond\,\phi$. Then, a state s satisfies the PTCL formula $\mathcal{P}_{\leq\lambda}(\Diamond\phi)$ iff $\mathsf{P}(\mathsf{s}, \Diamond\,\phi) \leq \lambda$. Maximal and minimal probabilities are computed by solving a linear programming problem [6,9]. The *iterative* algorithms implemented in Prism to solve this problem can combine different numerical computation methods with different data structures [13,14].

3.2.2 Model Checking PTAs.

We verify a PTA by model checking its probabilistic reachability graph using the following result [17]: 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 decission process using Prism's description language, a simple, state-based language, similar to Reactive Modules [3].

The behaviour of a system is described by a set of *guarded commands* of the form `[] <guard> -> <command>`. A *guard* is a predicate over variables of the system. A *command* describes a transition which the system can make if the guard is true, by giving a new value to *primed* variables as a function on the old values of *unprimed* variables. We consider two types of encoding of a reachability graph in this language.

4.1 Explicit Encoding

The first solution is a direct explicit encoding of the reachability graph using a single variable s whose value is the index of the state of the reachability graph. Transitions of the system are 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 transitions from 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)
```

This encoding generates a description the size of the reachability graph, which can grow drastically with the value of the deadline. Prism involves a model construction phase, during which the system description is parsed and converted into an MTBDD representation for further analysis. This phase can be extremely time consuming when the input file does not correspond to a modular and structured description of a system, such as with the explicit encoding. Thus, an encoding allowing for a more compact description of the system is needed.

4.2 Instances Encoding

States of a reachability graph correspond to several instances of locations of the timed automaton from which it was generated. 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. For example, let $l = 0$ be the value of the location variable corresponding to send. Then, states $0, 4$ and 7 correspond to three different instances of this location, say $n = 0$, $n = 1$ and $n = 2$. Then, the outgoing transitions from states corresponding to send can be specified by:

```
[] (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)
```

4.2.1 Relative compaction

The instance variable n is left unchanged by the command, 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 can be described more compactly in a single line:

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

Since in a reachability graph a transition between two given locations can be repeated several times for different instances, this encoding allows us to specify them in a more compact manner. 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.

The compaction algorithm is 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. Moreover, we combine different source states corresponding to the same location and successive numbers of instance, in a simple constraint where n is between two bounds, as in the example above.

4.2.2 Absolute compaction

In a reachability graph, we can encounter states which are the destination of many different transitions, such as state error_before ($l = 3$, $n = 0$) in the example of Figure 2. In this case, if we specify the updated value n' with its absolute value, an algorithm similar to the one above will allow us to describe all the incoming transitions in a single line. For example, the two transtions to error_before can be described by:

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

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 practise 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

The heuristic implemented consists in first applying the relative compaction and then, for those transitions that couldn't be compacted, change the way the command updates the value of n from relative to absolute, and apply the absolute compaction.

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 Model

The probabilistic timed automaton $\mathtt{I}_1^p$ in figure 3 is the abstract model of the root contention protocol considered in [15]. It is a probabilistic extension of the timed automaton $\mathtt{I}_1$ of [20] 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 timing constraints in $\mathtt{I}_1^p$ correspond to those specified in the updated standard IEEE 1394a. For instance, 360 ns corresponds to the transmission delay in the network if nodes are connected using a long wire. Other types of wire will have a different transmission delay, and hence can be verified by changing this value and re-running the experiments. Naturally, a lower delay results in a greater or equal probability of electing a leader before a deadline.

5.2 Verification

We first generate the reachability graph of the probabilistic timed automaton $\mathtt{I}_1^p$ 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. Upon entering the location done, we test in *time zero* (clock x is reset in all incoming edges, and an invariant $x = 0$ forces the system to leave the location immediately) whether the deadline is exceeded or not, by adding two outgoing edges from done, one with the guard $y > D$ leading to a location done_after and one with the guard $y \leq D$ leading to a location done_before.

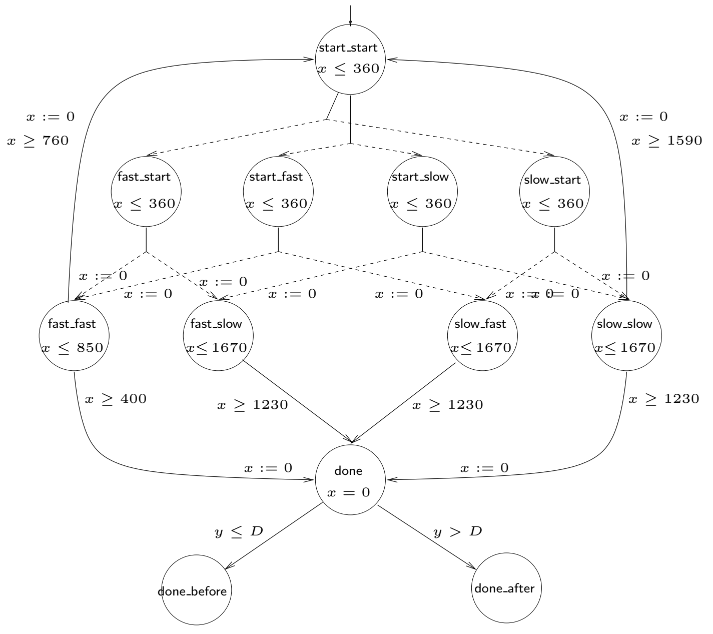

Fig. 3. Probabilistic timed automaton $\mathtt{I}_1^p$ modelling the root contention protocol.

Then, we specify the property of the root contention protocol we are interested in, namely, that a leader is elected *before* the deadline with *at least* a given probability. The PCTL formula that specifies this property is $\mathcal{P}_{\geq\lambda}(\Diamond\,(\mathsf{done} \wedge y \leq D))$, which 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 [15] that it is equivalent to the formula $\mathcal{P}_{<1-\lambda}(\Diamond\,\mathsf{done\_after})$ which can actually be verified on the reachability graph.

5.3 Experimental Results

In order to verify the property above, we compute the minimal probability for electing a leader before the deadline, for deadlines ranging from $4.10^3 ns$ to $100.10^3 ns$. 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 [19].

Table 1 shows the results concerning the generation with Kronos of the reachability graph and of its encoding as an MDP. 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 4 shows the evolution of the number of lines of the generated file for different values of the deadline. It demonstrates that the instances encoding allows for compactions which reduce drastically the number of lines of the

Table 1
Generation and encoding of the reachability graph

deadline	forw. reach.		explicit	instances	instances	instances
(10^3 ns)	states	time (s)		abs	rel	abs+rel
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

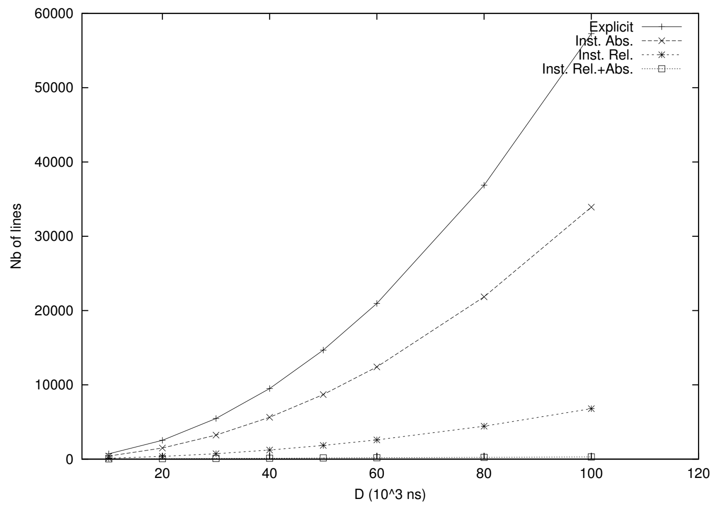

Fig. 4. Number of lines of the MDP

MDP file, and in the case where both relative and absolute compactions are considered, the number of lines grows less than linearly on the value of the deadline.

The experimental results concerning the verification with Prism are shown in Table 2. The left-most column shows the deadline used in the property, and the right-most column shows the minimum 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 same probability is computed for deadlines of more than 40000 ns, meaning that the iterative method has converged, i.e. that the actual probabilities differ by less than $\epsilon = 10^{-6}$.

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*) .

Table 2
Time performances for model building and verification

deadline	explicit		inst+rel		inst+rel+abs		probability
(10^3 ns)	model (s)	verif (s)	model (s)	verif (s)	model (s)	verif (s)	
4	0.626	0.009	0.061	0.006	0.054	0.007	0.625
6	1.588	0.013	0.111	0.007	0.073	0.008	0.851562
8	5.654	0.018	0.195	0.008	0.140	0.008	0.939453
10	13.338	0.029	0.301	0.009	0.196	0.010	0.974731
20	190.971	0.098	3.038	0.025	1.303	0.026	0.999629
30	1037.892	0.309	14.672	0.056	4.969	0.058	0.999993
40	–	–	344.251	0.134	30.147	0.112	0.999998
50	–	–	1119.008	0.349	50.129	0.204	0.999998
60	–	–	3468.310	0.442	233.272	0.351	0.999998
80	–	–	–	–	814.035	0.729	0.999998
100	–	–	–	–	2861.889	1.744	0.999998

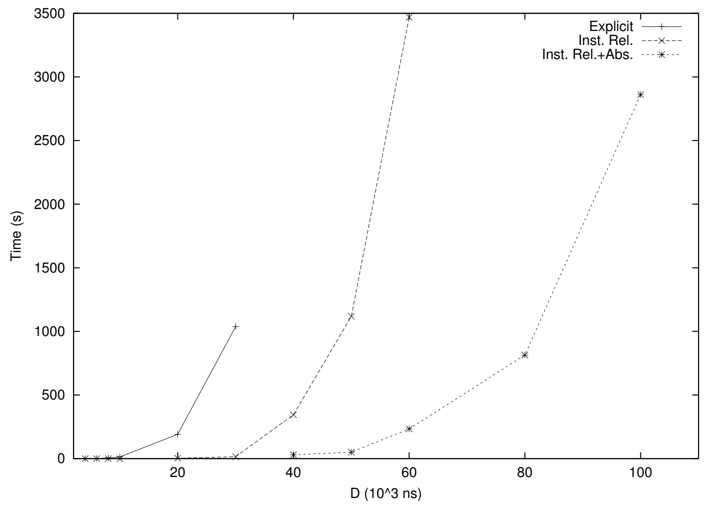

Fig. 5. Time to build the model

Compared to the previous attempt of verification [15] of the root contention protocol using HyTech [10], the generation of the reachability graph is no longer a problem, since it only took about 20 seconds to generate it 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 took less than two seconds in the worst case. It is clear from this results that the bottleneck of this verification approach is now the model building phase of Prism, and the practical success of our verification approach depends on improving either the encoding or the model building algorithms.

Figure 5 shows the evolution of the time needed to build the model for different deadlines using different encodings. We can see that, although compactions improve the time needed to build the model, the latter still grows

Table 3
Probability of leader election with a biased coin.

fast	*slow*	$D = 6000$	$D = 10000$
.01	.99	0.039211	0.076886
.10	.90	0.330534	0.551770
.20	.80	0.554516	0.801000
.30	.70	0.704352	0.910950
.40	.60	0.799150	0.957090
.45	.55	0.830027	0.968230
.50	.50	0.851562	0.974731
.55	.45	0.864616	0.977771
.60	.40	0.869498	0.977795
.65	.35	0.865609	0.974558
.70	.30	0.850898	0.966919
.80	.20	0.768942	0.923030
.90	.10	0.544273	0.746829
.99	.01	0.076872	0.130600

drastically with the value of the deadline, even when the size of the input file grows linearly, because of the complexity of the guards after compaction.

5.4 RCP with a biased coin

We now study the influence of using a biased coin on the performance of the protocol. As noted in [21], 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. This is because, although the protocol will require more rounds to elect a leader, the time per round is lower when both processes select fast timing.

Table 3 gives the probability for electing a leader before 6000ns or 10000 ns, for different values of the probability of choosing *fast* or *slow* timing for both nodes. The results presentend correspond to model checking the same property as before using the optimized instances encoding. Note that we don't need to compute the forward reachability for 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.

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 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 compact and structured manner. An explicit encoding of the reachability graph using a single variable to encode a state turned out to be infeasible even for small values of the deadline. The instances encoding using two variables, one corresponding to the location of the timed automaton, the other to the instance of this location in the reachability graph, allowed us to apply compaction techniques that helped overcoming this problem. However, it is not clear how general a solution this is. Finding a good encoding is then crucial.

Naturally, we need to validate this approach by applying it to other systems or protocols where timing and probabilistic aspects arise. In order to do this, a better integration of both tools is needed. A first step in this direction would be to implement the parallel composition of probabilistic timed automata so that we are able to model complex systems in a compositional way.

Acknowledgement

We thank Sergio Yovine for making Kronos' libraries available to us.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV'00*, LNCS, 2000.

[5] C. Baier and M. Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[6] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. S. Thiagarajan, editor, *Proceedings of FSTTCS'95*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.

[7] 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.

[8] C. Daws and S. Tripakis. Model–checking of real–time reachability properties using abstractions. In B. Steffen, editor, *Proc. TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer-Verlag, 1998.

[9] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In J. Baeten and S. Mauw, editors, *Proceedings of CONCUR'99*, volume 1664 of *LNCS*, pages 66–81. Springer Verlag, 1999.

[10] T. A. 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.

[11] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains.* Graduate Texts in Mathematics. Springer, 2nd edition, 1976.

[12] KRONOS web page. `http://www-verimag.imag.fr/TEMPORISE/kronos/`.

[13] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In J. B. T. Field, P. Harrison and U. Harder, editors, *Proc. TOOLS 2002*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[14] 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. TACAS 2002*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.

[15] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 firewire root contention protocol. In *Proc. Int. Workshop on Application of Formal Methods to IEEE 1394 Standard*, 2001.

[16] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. volume 1601 of *LNCS*, pages 75–95. Springer-Verlag, 1999.

[17] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 286, 2001. To appear.

[18] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.

[19] PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism/`.

[20] 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*, 2001. To appear.

[21] M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems.* PhD thesis, Univerisity of Nijmegen, 2002.

[22] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings FOCS'85.* IEEE Computer Society Press, 1985.