Quantitative Verification: Models, Techniques and Tools[*]

Marta Kwiatkowska
Oxford University Computing Laboratory
Parks Road, Oxford OX1 3QD, UK
mzk@comlab.ox.ac.uk

ABSTRACT

Automated verification is a technique for establishing if certain properties, usually expressed in temporal logic, hold for a system model. The model can be defined using a high-level formalism or extracted directly from software using methods such as abstract interpretation. The verification proceeds through exhaustive exploration of the state-transition graph of the model and is therefore more powerful than testing. *Quantitative verification* is an analogous technique for establishing quantitative properties of a system model, such as the probability of battery power dropping below minimum, the expected time for message delivery and the expected number of messages lost before protocol termination. Models analysed through this method are typically variants of Markov chains, annotated with costs and rewards that describe resources and their usage during execution. Properties are expressed in temporal logic extended with probabilistic and reward operators. Quantitative verification involves a combination of a traversal of the state-transition graph of the model and numerical computation.

This paper gives a brief overview of current research in quantitative verification, concentrating on the potential of the method and outlining future challenges. The modelling approach is described and the usefulness of the methodology illustrated with an example of a real-world protocol standard – Bluetooth device discovery – that has been analysed using the PRISM model checker (`www.prismmodelchecker.org`).

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods, model checking, statistical methods*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs, mechanical verification*

[*]Supported in part by EPSRC grants EP/D076625, EP/D07956X and EP/F001096.

General Terms

Theory; Verification; Performance; Reliability

Keywords

Markov models; Probabilistic temporal logics; Probabilistic model checking; Quantitative model checking

1. INTRODUCTION

Automated verification techniques have made steady progress in the past few years, and are now frequently incorporated into the software engineering process for use in addition to established methods such as testing. An important advantage of automated verification is its ability to establish, through exhaustive traversal of the underlying model, that the system is error-free, or else produce a diagnostic trace. As is well known, automated verification via model checking suffers from the state-space explosion problem, which limits the size of the models that can be verified. Recent progress with techniques such as counter-example guided abstraction refinement has considerably enhanced the applicability of model checking. Examples of widely-used software tools include e.g. LTSA and FDR, which analyse labelled transition system models described in a process algebra notation, and SLAM and Bandera, which are applied directly to software in C or Java.

The vast majority of research into model checking has concentrated on developing methods for analysing functional, *qualitative* properties of system/software models, for example, whether the executions never violate a safety property, or the program eventually terminates. Many programs, however, contain random assignment and/or real-time delays, which necessitates *quantitative* analysis in order to establish properties such as the probability of termination within a given time limit. Examples of real-world protocols that involve such timing delays and randomisation are root contention in IEEE 1394 FireWire and random back-off schemes in e.g. IEEE 802.11 and Bluetooth. Probability is also used to quantify unreliable or unpredictable behaviour, for example in fault-tolerant systems and computer networks, where properties such as component failure and packet loss can be described probabilistically.

Traditionally, quantitative evaluation of systems has been performed through performance analysis, where a probabilistic model of the system is derived, typically a continuous time Markov chain, on which analytical, simulation-based or numerical calculations are performed to obtain the desired quantitative measures [34]. In AI, a different probabilistic model (Markov decision processes) is used for planning and control problems solvable via Bellmann equations, for example through value or policy iteration [7]. In recent years, a complementary technique of *probabilistic model checking*, an automated verification technique for probabilistic models, has been developed [35, 11, 15, 8, 6, 4, 32]. The models are similar to those used in performance analysis and planning, i.e. variants of Markov chains or Markov decision processes, in the sense that they encode the *probability* or *rate* of making a transition between states. This induces a probability space on the system behaviours and enables the calculation of the *likelihood* of the occurrence of certain events during the execution of the system rather than the usual existential or universal quantification over execution paths. Based on techniques formulated for qualitative (i.e. with probability 1 or 0) temporal logic model checking for Markov chains [35], a *quantitative* variant [11, 6] has been developed with which one can express the following *quantitative, probabilistic* statements:

- for a system which can suffer failures: "what is the *probability* of a critical fault occurring during execution?"
- for a multimedia protocol: "what is the *maximum* probability of delivering a packet in the next 5ms?"

When the models are additionally annotated with costs and rewards that specify resource usage during execution, the method also enables more general *quantitative expected reward* statements:

- for a wireless communication protocol: "what is the *worst case* expected time for delivering a data packet?"
- for a battery-powered device: "what is the *expected* power consumption during the first 20s of operation?".

Quantitative verification generalises conventional temporal logic model checking by the addition of probabilistic and reward operators to the logic. Its main advantage is that the analysis is exhaustive. In that, probabilistic model checking draws on conventional model checking, by performing traversal of the underlying state-transition system graph; this is achieved through a judicious combination with numerical and other algorithmic techniques often drawn from performance analysis and planning, for example, uniformisation and value iteration. Therefore, quantitative verification offers the full benefit of temporal logic model checking, in addition to performing quantitative evaluation.

Quantitative and probabilistic verification have been applied in a multitude of domains: distributed coordination algorithms, wireless communication protocols, security and anonymity protocols, nanotechnology designs, power management and biological modelling; for more information, see [30, 20, 32, 22, 23]. In this paper we give an overview of formalisms and techniques employed in quantitative model checking. We describe the well-known probabilistic model of discrete time Markov chains and its reward extension, with examples of how reward structures can be used to describe resources. Next, we summarise the probabilistic temporal logic PCTL, including the probabilistic and reward operators, and the corresponding model checking methods. A brief overview of the features and model checking algorithms of three other model types is also given. The models and specification formalisms introduced here are supported by the probabilistic model checker PRISM [30, 16], which is briefly introduced. We illustrate the capabilities and limitations of the quantitative verification techniques with an example of a protocol standard analysed with PRISM: the Bluetooth device discovery [13]. We conclude by outlining the challenges that remain in the area.

2. THE MODELS

Probabilistic models used in quantitative verification generalise labelled state-transition systems by the addition of *probabilistic* transitions. Probabilistic transitions can replace the usual nondeterminism or the latter can be partly retained in the model to describe aspects such as scheduling, where probability of taking a transition is not known. To illustrate the methodology, we present in some detail the simple yet expressive Markov chain model, together with a reward extension. This model admits only probabilistic choice between transitions. The exposition is illustrated with a running example to explain how probability and reward structures are used in quantitative verification. This is followed by a summary of the main features of three other important probabilistic models, each loosely based on Markov chains.

2.1 Preliminaries

Let Ω be a *sample set*, the set of possible outcomes of an experiment. A subset of Ω is called an *event*. $(\Omega, \mathcal{F})$ is said to be a *sample space* if $\mathcal{F}$ is a σ-field of subsets, often built from *basic cylinders/cones* by closing w.r.t. countable unions and complement. $(\Omega, \mathcal{F}, \mu)$ is a *probability space* if μ is a probability measure, i.e. $0 \leq \mu[A] \leq 1$ for all $A \in \mathcal{F}$; $\mu[\emptyset] = 0$, $\mu[\Omega] = 1$, and $\mu[\bigcup_{k=1}^{\infty} A_k] = \Sigma_{k=1}^{\infty} \Pr[A_k]$, A_k disjoint.

For a finite set S, a *probability distribution* on S is a function $\mu : S \rightarrow [0,1]$ such that $\sum_{t \in S} \mu(t) = 1$. Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. A function $X : \Omega \rightarrow \mathbb{R}_{\geq 0}$ is said to be a *random variable*. Given a random variable $X : \Omega \rightarrow \mathbb{R}_{\geq 0}$ and the probability space $(\Omega, \mathcal{F}, \mu)$ the expectation or average value with respect to the measure μ is given by the following integral:

$$E[X] \stackrel{\text{def}}{=} \int_{\omega \in \Omega} X(\omega)\, \mathrm{d}\mu \,.$$

We assume a fixed set AP of atomic propositions.

2.2 Discrete Time Markov Chains

A discrete time Markov chain consists of discrete states, representing the configurations of the system, and has transitions governed by a (discrete) probability distribution on the target states. Formally, a (labelled) *discrete time Markov chain* (DTMC) $\mathcal{D}$ is a tuple $(S, \bar{s}, \mathbf{P}, L)$ where

- S is a finite set of *states*
- $\bar{s}$ is an *initial state*
- $\mathbf{P} : S \times S \rightarrow [0,1]$ is the *transition probability matrix* where $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$, and
- $L : S \rightarrow 2^{AP}$ is a *labelling* with atomic propositions that are true in s.

Each element $\mathbf{P}(s, s')$ of the matrix gives the probability of taking a transition from s to s'. Each transition is assumed to take a discrete time-step. Terminating states are

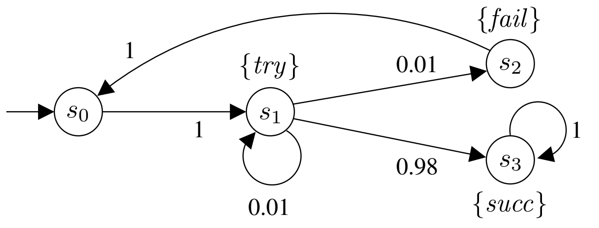


Figure 1: A simple DTMC $\mathcal{D}$

modelled with a self-loop. Note also that there is no notion of real time, though reasoning about discrete time is possible through state variables keeping track of time and 'counting' transition steps.

The behaviour of a DTMC model is represented as a set of paths. A *path* through a DTMC is a non-empty (finite or infinite) sequence of states $\omega = s_0\, s_1\, s_2 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. The probability matrix $\mathbf{P}$ induces a probability space on the set of infinite paths $Path_s$, which start in the state s, using the cylinder construction [19] as follows. An observation of a finite path determines a basic event (cylinder). Let $s = s_0$. For $\omega = s_0 s_1 \ldots s_n$, we define the probability measure $\mathrm{Pr}_s^{\mathrm{fin}}$ for the ω-cylinder by putting $\mathrm{Pr}_s^{\mathrm{fin}} = 1$ if ω consists of a single state, and $\mathrm{Pr}_s^{\mathrm{fin}} = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1}, s_n)$ otherwise. This extends to a unique measure Pr_s on the infinite paths $Path_s$ by a well-known theorem.

EXAMPLE 1. *Figure 1 shows an example of a DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ modelling a simple communication protocol where $AP = \{try, fail, succ\}$ and $\overline{s} = s_0$. The transition probability matrix $\mathbf{P}$ is given by:*

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In the state s_1, the protocol tries *to send a message, and it either* succeeds *with probability* 0.98*,* fails *with probability* 0.01*, or, with with probability* 0.01*, waits another time-step and restarts. This example DTMC will serve as the running example throughout this section.*

2.2.1 The Logic PCTL

Specifications for DTMC models can be written in PCTL (Probabilistic Computation Tree Logic) [15], a probabilistic extension of the temporal logic CTL. The syntax of PCTL is as follows:

$$\begin{aligned} \Phi &::= \mathtt{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathtt{P}_{\sim p}[\alpha] \\ \alpha &::= \mathtt{X}\ \Phi \mid \Phi\ \mathtt{U}^{\leq k}\ \Phi \end{aligned}$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ and $k \in \mathbb{N} \cup \{\infty\}$. PCTL formulae are interpreted over the states of a DTMC. Instead of the existential and universal quantification of CTL, PCTL has the *probabilistic operator* $\mathtt{P}_{\sim p}[\cdot]$ where $p \in [0,1]$ is a *probability bound* or *threshold*. Path formulae can occur only within the scope of the probabilistic operator. Intuitively, a state s of $\mathcal{D}$ satisfies $\mathtt{P}_{\sim p}[\alpha]$ if the probability of taking a path from s satisfying α is in the interval specified by $\sim p$. Formally, the meaning of the probabilistic operator is:

$$s \models \mathtt{P}_{\sim p}[\alpha] \quad \text{iff} \quad \mathrm{Pr}_s\{\omega \in Path_s \mid \omega \models \alpha\} \sim p.$$

Two forms of path formulae αareallowed, $\mathtt{X}\ \Phi$ and $\Phi\ \mathtt{U}^{\leq k}\ \Psi$, the next state and bounded until. When $k = \infty$ we abbreviate $\Phi\ \mathtt{U}^{\leq k}\ \Psi$ by $\Phi\ \mathtt{U}\ \Psi$ (unbounded until). As is standard in temporal logic, $\mathtt{X}\ \Phi$ is true for a path $\omega \in Path_s$ if Φ is satisfied in the next state, and $\Phi\ \mathtt{U}^{\leq k}\ \Psi$ is true if Ψ is satisfied within k time-steps and Φ is true up until that point. The intuition is that the probability measure of the set of α-paths is calculated and compared to the probability bound, yielding true or false respectively (that this set is measurable was shown in [35]). The *qualitative* PCTL fragment are the formulae where p is equal to 0 or 1. Note that while $\mathtt{P}_{>0}[\Phi_1\ \mathtt{U}\ \Phi_2]$ equates to existential quantification, $\mathtt{P}_{\geq 1}[\Phi_1\ \mathtt{U}\ \Phi_2]$ is a weaker analogue of universal quantification.

Each PCTL formula evaluates to a Boolean by virtue of the comparison of the actual probability of the set of α-paths with the probability bound p. If the outermost operator of a PCTL formula is $\mathtt{P}_{\sim p}[\cdot]$, we can omit the bound $\sim p$ and simply compute the probability instead. The PCTL model checking algorithm computes the actual probability anyway, so no extra cost is incurred. It is also often useful to study a range of such values by varying one or more parameters, either of the model or of the property, see the notion of experiment in Section 4.1.

EXAMPLE 2. *Below are examples of PCTL formulae for the DTMC example in Figure 1:*

- $\mathtt{P}_{<0.1}[true\ \mathtt{U}^{\leq 3}\ fail]$ - *the probability that failure occurs within 3 time-steps is strictly less than* 0.1*;*
- $\mathtt{P}_{=?}[try\ \mathtt{U}\ succ]$ - *what is the probability of sending a message successfully without failure?*

2.2.2 Model Checking for PCTL over DTMCs

The PCTL model checking algorithm [11, 15, 12] takes as inputs a labelled DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ and a PCTL formula Φ. The algorithm proceeds, as for CTL [10], by bottom-up traversal of the parse tree for Φ, recursively computing the set $Sat(\Psi) = \{s \in S \mid s \models \Psi\}$ of states satisfying each subformula Ψ. Therefore, the algorithm will compute the set of *all* states satisfying Φ.

All cases except the probabilistic operator $\mathtt{P}_{\sim p}[\alpha]$ are the same as for CTL. Here, it is helpful to view the DTMC as the matrix $\mathbf{P}$ and $Sat(\Phi)$ as a *column vector* $\underline{\Phi} : S \longrightarrow \{0,1\}$ given by $\underline{\Phi}_s = 1$ if $s \models \Phi$ and 0 otherwise. We have:

$$Sat(\mathtt{P}_{\sim p}[\alpha]) \quad = \quad \{s \in S \mid Prob_s(\alpha) {\sim} p\}.$$

where $Prob_s(\alpha)$ is the probability of satisfying α in s. Then, in vector notation, $\underline{Prob}(\mathtt{X}\ \Phi) = \mathbf{P} \cdot \underline{\Phi}$, i.e. a single matrix by vector multiplication suffices.

Next, we consider bounded until formulae $\mathtt{P}_{\sim p}[\Phi\ \mathtt{U}^{\leq k}\ \Psi]$. For such formulae we need to determine the probabilities $Prob_s(\Phi\ \mathtt{U}^{\leq k}\ \Psi)$ for all states s where $k \in \mathbb{N} \cup \{\infty\}$. When $k \in \mathbb{N}$, the corresponding vector $\underline{Prob}(\Phi\ \mathtt{U}^{\leq k}\ \Psi)$ can be expressed in terms of the transient probabilities of a DTMC [24]. Firstly, for any DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ and PCTL formula Φ, let $\mathcal{D}[\Phi] = (S, \overline{s}, \mathbf{P}[\Phi], L)$ be the DTMC $\mathcal{D}$ modified as follows: if $s \not\models \Phi$, then $\mathbf{P}[\Phi](s, s') = \mathbf{P}(s, s')$ for all $s' \in S$, and if $s \models \Phi$, then $\mathbf{P}[\Phi](s, s) = 1$ and $\mathbf{P}[\Phi](s, s') = 0$

for all $s' \neq s$. Intuitively, $\mathcal{D}[\Phi]$ is obtained from $\mathcal{D}$ by making all states in Φ absorbing (the only transitions in those states are self-loops). To obtain the solution for bounded until, we transform the DTMC to $\mathcal{D}[\neg\Phi \vee \Psi] = \mathcal{D}[\neg(\Phi \wedge \Psi)][\Psi]$. States in $Sat(\neg(\Phi \wedge \Psi))$ are made absorbing because $Prob_s(\Phi \ \mathtt{U}^{\leq k} \ \Psi)$ is trivially 0, and likewise for states in $Sat(\Psi)$ because $Prob_s(\Phi \ \mathtt{U}^{\leq k} \ \Psi)$ is trivially 1. It can be shown [24] that the vector of probabilities $\underline{Prob}(\Phi \ \mathtt{U}^{\leq k} \ \Psi)$ can then be computed using the following matrix and vector multiplications:

$$\underline{Prob}(\Phi \ \mathtt{U}^{\leq k} \ \Psi) = (\mathbf{P}[\neg\Phi \vee \Psi])^k \cdot \underline{\Psi}\,.$$

This product is typically computed in an iterative fashion:

$$\mathbf{P}[\neg\Phi \vee \Psi] \cdot (\cdots (\mathbf{P}[\neg\Phi \vee \Psi] \cdot \underline{\Psi}) \cdots)$$

which requires k matrix-vector multiplications.

EXAMPLE 3. *Consider the DTMC $\mathcal{D}$ in Figure 1 and the PCTL formula $\mathtt{P}_{>0.9}[\mathtt{true} \ \mathtt{U}^{\leq 1} \ succ]$. We have $Sat(\mathtt{true}) = \{s_0, s_1, s_2, s_3\}$ and $Sat(succ) = \{s_3\}$. The matrix $\mathbf{P}[\neg\mathtt{true} \vee succ]$ is identical to $\mathbf{P}$, and we have that:*

$$\begin{array}{rcl} \underline{Prob}(\Phi \ \mathtt{U}^{\leq 0} \ \Psi) & = & [0,0,0,1] \\ \underline{Prob}(\Phi \ \mathtt{U}^{\leq 1} \ \Psi) & = & [0,0.98,0,1]\,. \end{array}$$

Hence, $Sat(\mathtt{P}_{>0.9}[\mathtt{true} \ \mathtt{U}^{\leq 1} \ succ]) = \{s_1, s_3\}$.

Finally, when $k = \infty$, i.e. we have the formula $\Phi \ \mathtt{U} \ \Psi$, the probabilities $Prob_s(\Phi \ \mathtt{U} \ \Psi)$ are obtained as the unique solution of the *linear equation system* [11] in variables $\{x_s \mid s \in S\}$:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot x_{s'} & \text{if } s \in S^{?} \end{cases}$$

where $S^{no} = Sat(\mathtt{P}_{\leq 0}[\Phi \ \mathtt{U} \ \Psi])$ and $S^{yes} = Sat(\mathtt{P}_{\geq 1}[\Phi \ \mathtt{U} \ \Psi])$ denote, respectively, the sets of *all* states that satisfy $\Phi \ \mathtt{U} \ \Psi$ with probability exactly 0 (resp. 1), and $S^{?} = S \setminus (S^{no} \cup S^{yes})$. The sets S^{no} and S^{yes} are precomputed using conventional fixed point computation. Since the values for these states are known (0 or 1), the solution of the resulting linear equation system in $|S^{?}|$ variables can be obtained by any direct method (e.g. Gaussian elimination) or iterative method (e.g. Jacobi, Gauss-Seidel). For qualitative PCTL properties, it suffices to use these precomputation algorithms alone. For quantitative properties with an arbitrary bound p, numerical computation is also usually required. Note that the precomputation algorithms determine the exact probability in case it is 0 or 1, thus avoiding the problem of round-off errors that are typical for numerical computation.

EXAMPLE 4. *Consider again the DTMC $\mathcal{D}$ in Figure 1 and the PCTL formula $\mathtt{P}_{=?}[try \ \mathtt{U} \ succ]$ which queries the probability of successfully sending a message without failure. We have $Sat(try) = \{s_1\}$ and $Sat(succ) = \{s_3\}$. We calculate that $Sat(\mathtt{P}_{\leq 0}[try \ \mathtt{U} \ succ]) = \{s_0, s_2\}$ and $Sat(\mathtt{P}_{\geq 1}[try \ \mathtt{U} \ succ]) = \{s_3\}$. The resulting linear equation system is:*

$$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 0.01 \cdot x_1 + 0.01 \cdot x_2 + 0.98 \cdot x_3 \\ x_2 & = & 0 \\ x_3 & = & 1\,. \end{array}$$

This yields the solution $(0, \frac{98}{99}, 0, 1)$ and we see that the formula $\mathtt{P}_{=?}[try \ \mathtt{U} \ succ]$ returns $\frac{98}{99}$ in state s_1.

2.2.3 Extending DTMCs and PCTL with Rewards

In this section we introduce rewards which can be used to annotate DTMCs with information about resources and their usage, for example the power consumption or the number of lost messages. Let $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ be a DTMC, which we endow with a reward structure $(\underline{\rho}, \boldsymbol{\iota})$. A *state* reward is a function $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$; intuitively, $\underline{\rho}(s)$ is the reward acquired in state s per time-step. A *transition* reward is a function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geq 0}$. This type of reward is acquired each time a transition between states s and s' occurs.

EXAMPLE 5. *Consider the DTMC $\mathcal{D}$ in Figure 1 endowed with the reward structure $(\underline{\rho}, \mathbf{0})$, where $\underline{\rho}(s) = 1$ if $s = s_1$ and equals 0 otherwise. This reward counts the number of time-steps spent in state s_1 and can be used to calculate, for example, energy usage while attempting to send the message.*

The logic PCTL is extended to allow for the reward properties by means of the following state formulae:

$$\mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}] \ \big| \ \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] \ \big| \ \mathtt{R}_{\sim r}[\mathtt{F} \ \Phi]$$

where $\sim \in \{<, \leq, \geq, >\}$, $r \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$ and Φ is a PCTL state formula.

Intuitively, a state s satisfies $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}]$ (*cumulative* reward) if, from state s, the expected reward cumulated after k time-steps satisfies $\sim r$; $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$ (*instantaneous*) is true if from state s the expected state reward at time-step k meets the bound $\sim r$; and $\mathtt{R}_{\sim r}[\mathtt{F} \ \Phi]$ (*reachability*) is true if from state s the expected reward cumulated before a state satisfying Φ is reached meets the bound $\sim r$. Formally, for a DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$, the semantics of the reward operator is defined using expectation of an appropriate random variable:

$$\begin{array}{lcl} s \models \mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}] & \Leftrightarrow & Exp_s(X_{\mathtt{C}^{\leq k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{I}^{=k}] & \Leftrightarrow & Exp_s(X_{\mathtt{I}^{=k}}) \sim r \\ s \models \mathtt{R}_{\sim r}[\mathtt{F} \ \Phi] & \Leftrightarrow & Exp_s(X_{\mathtt{F}\Phi}) \sim r \end{array}$$

for any $s \in S$, $k \in \mathbb{N}$, $r \in \mathbb{R}_{\geq 0}$ and PCTL formula Φ. In the above, $Exp_s(X)$ denotes the expectation of the random variable $X : Paths_s \to \mathbb{R}_{\geq 0}$ with respect to the probability measure $\Pr_s$ on the set of paths $Paths_s$ of $\mathcal{D}$. The random variables corresponding to the three forms of the reward operator are defined, for any path $\omega = s_0 s_1 s_2 \cdots \in Paths_s$, as follows:

$$X_{\mathtt{C}^{\leq k}}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{\mathtt{I}^{=k}}(\omega) \stackrel{\text{def}}{=} \underline{\rho}(s_k)$$

$$X_{\mathtt{F}\Phi}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}.\, s_i \not\models \Phi \\ \sum_{i=0}^{\min\{j \mid s_j \models \Phi\}-1} \underline{\rho}(s_i) + \boldsymbol{\iota}(s_i, s_{i+1}) & \\ & \text{otherwise.} \end{cases}$$

Similarly to the probabilistic operator, if the outermost operator of a PCTL formula is $\mathtt{R}_{\sim r}[\cdot]$, we can omit the bound $\sim r$ and compute the expected value instead. This also enables a range of such values to be obtained by varying one or more parameters, either of the model or of the property.

EXAMPLE 6. *Below are some examples of reward based specifications:*

- $\mathtt{R}_{\leq 5.5}[\mathtt{C}^{\leq 100}]$ - *the expected power consumption within the first 100 time-steps of operation is less than or equal to 5.5;*

- $\mathtt{R}_{=?}[\mathtt{I}^{=10}]$ - *what is the expected number of messages still to be delivered after 10 time-steps have passed?*
- $\mathtt{R}_{\geq 5}[\mathtt{F}\ succ]$ - *the expected number of correctly delivered messages is at least 5.*

2.2.4 Model Checking for DTMCs with Rewards

We now consider model checking of reward formulae. The input for the algorithm is a DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$, together with a reward structure $(\underline{\rho}, \boldsymbol{\iota})$, and a PCTL reward formula Φ. Since the PCTL model checking algorithm proceeds by bottom-up traversal of the parse tree of the formula, model checking for the reward operator reduces to the computation of the expected values for each of the corresponding random variables every time a reward subformula $\mathtt{R}_{\sim r}[\cdot]$ is encountered. Below we consider each type of the reward formula in turn.

The reward formula $\mathtt{R}_{\sim r}[\mathtt{C}^{\leq k}]$. In this case the the vector of expected values can be computed iteratively by means of the following matrix-vector operations:

$$\underline{Exp}(X_{\mathtt{C}^{\leq k}}) = \begin{cases} \underline{0} & \text{if } k=0 \\ \underline{\rho} + (\mathbf{P} \bullet \boldsymbol{\iota})\cdot\underline{1} + \mathbf{P}\cdot\underline{Exp}(X_{\mathtt{C}^{\leq k-1}}) & \text{otherwise} \end{cases}$$

with $\bullet$ denoting the Schur or entry-wise multiplication of matrices and $\underline{1}$ a vector with all entries equal to 1.

EXAMPLE 7. *Consider the DTMC $\mathcal{D}$ in Figure 1 with the reward structure of Example 7. The PCTL formula $\mathtt{R}_{=?}[\mathtt{C}^{\leq k}]$ queries the expected number of time steps spent in each state after k time steps. We calculate:*

$$\begin{aligned} \underline{Exp}(X_{\mathtt{C}^{\leq 0}}) &= [0,0,0,0] \\ \underline{Exp}(X_{\mathtt{C}^{\leq 1}}) &= [0,1,0,0] + \mathbf{P}\cdot[0,0,0,0] \\ &= [0,1,0,0] \\ \underline{Exp}(X_{\mathtt{C}^{\leq 2}}) &= [0,1,0,0] + \mathbf{P}\cdot[0,1,0,0] \\ &= [1,1.01,0,0] \\ \underline{Exp}(X_{\mathtt{C}^{\leq 3}}) &= \cdots \end{aligned}$$

and hence $\mathtt{R}_{=?}[\mathtt{C}^{\leq 2}]$ returns 1.01 in state s_1.

The reward formula $\mathtt{R}_{\sim r}[\mathtt{I}^{=k}]$. In this case the the vector of expected values can be computed by means of the following matrix-vector operations:

$$\underline{Exp}(X_{\mathtt{I}^{=k}}) = \begin{cases} \underline{\rho} & \text{if } k=0 \\ \mathbf{P}\cdot\underline{Exp}(X_{\mathtt{I}^{=k-1}}) & \text{otherwise.} \end{cases}$$

EXAMPLE 8. *For the running example DTMC consider the PCTL formula $\mathtt{R}_{>0}[\mathtt{I}^{=k}]$ which specifies that, at time-step k, the expectation of being in state s_1 is greater than 0. We have:*

$$\begin{aligned} \underline{Exp}(X_{\mathtt{I}^{=0}}) &= [0,1,0,0] \\ \underline{Exp}(X_{\mathtt{I}^{=1}}) &= \mathbf{P}\cdot[0,1,0,0] = [1,0.01,0,0] \\ \underline{Exp}(X_{\mathtt{I}^{=2}}) &= \mathbf{P}\cdot[1,0.01,0,0] = [0.01,0.0001,1,0]\,. \end{aligned}$$

Hence, the states s_0, s_1 and s_2 satisfy the formula $\mathtt{R}_{>0}[\mathtt{I}^{=2}]$.

The reward formula $\mathtt{R}_{\sim r}[\mathtt{F}\Phi]$. It can be shown [24] that the expectations in this case are calculated by solving a system of linear equations obtained as follows. Firstly, we identify the set of states s for which $Exp_s(X_{\mathtt{F}\Phi})$ equals ∞. This includes the states for which the probability of reaching a Φ state is less than 1, that is, the set $Sat(\mathtt{P}_{<1}[\mathtt{true}\ \mathtt{U}\ \Phi])$. This set can be computed from the equivalence $\mathtt{P}_{<1}[\mathtt{true}\ \mathtt{U}\ \Phi] \equiv \neg\mathtt{P}_{\geq 1}[\mathtt{true}\ \mathtt{U}\ \Phi]$. Let $S^{\Phi} = Sat(\Phi)$, $S^{\infty} = Sat(\mathtt{P}_{<1}[\mathtt{true}\ \mathtt{U}\ \Phi])$. One can then compute $Exp(s, X_{\mathtt{F}\Phi})$ as the unique solution of the following linear equation system in variables $\{x_s \mid s \in S\}$:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{\Phi} \\ \infty & \text{if } s \in S^{\infty} \\ \underline{\rho}(s) + \sum_{s' \in S} \mathbf{P}(s,s')\cdot\big(\boldsymbol{\iota}(s,s') + x_{s'}\big) & \\ & \text{otherwise.} \end{cases}$$

Similarly to probabilistic until formulae, this can be solved using any standard direct or iterative method.

EXAMPLE 9. *Consider again the running example. The PCTL formula, $\mathtt{R}_{<1}[\mathtt{F}\ succ]$, in this case, asserts that the expected number of times state s_1 is entered before reaching a state satisfying succ is less than 1. We compute:*

$$\begin{aligned} Sat(succ) &= \{s_3\} \\ Sat(\mathtt{P}_{<1}[\mathtt{true}\ \mathtt{U}\ succ]) &= Sat(\neg\mathtt{P}_{\geq 1}[\mathtt{true}\ \mathtt{U}\ succ]) \\ &= S \setminus Sat(\mathtt{P}_{\geq 1}[\mathtt{true}\ \mathtt{U}\ succ]) \\ &= S \setminus \{s_0, s_1, s_2, s_3\} = \emptyset. \end{aligned}$$

This leads to the linear equation system:

$$\begin{aligned} x_0 &= 0{+}1.00\cdot(0{+}x_1) \\ x_1 &= 1{+}0.01\cdot(0{+}x_1){+}0.01\cdot(0{+}x_2) \\ x_2 &= 0{+}1.00\cdot(0{+}x_0) \\ x_3 &= 0 \end{aligned}$$

which has the solution $\left(\frac{100}{98}, \frac{100}{98}, \frac{100}{98}, 0\right)$, and hence it follows that $Sat(\mathtt{R}_{<1}[\mathtt{F}\ succ]) = \{s_3\}$.

2.2.5 Complexity of PCTL Model Checking

The time complexity for PCTL model checking over a DTMC $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ is linear in the size of the formula $|\Phi|$ (number of logical connectives and temporal operators) and polynomial in $|S|$ [15].

2.3 Other Model Types

The discrete time Markov chain model, sometimes referred to as fully probabilistic, admits only probabilistic choice between transitions and does not faithfully model real-time passage. This limitation is not present in other model types, each admitting a reward extension: Markov decision processes (which feature both nondeterministic as well as probabilistic choice), continuous time Markov chains (which model continuous time, but no nondeterminism) and probabilistic timed automata (which admit dense time, probabilistic choice and nondeterminism).

2.3.1 Markov Decision Processes

Markov decision processes (MDPs) [7] (also known as concurrent Markov chains [35]) generalise discrete time Mar-kov chains by allowing in each state a nondeterministic choice between a number of probability distributions over target states. This nondeterminism is a result of scheduling of parallel processes or underspecification, where the probabilities of taking a particular transition are not known. In a Markov decision process both states and time are discrete. It is assumed that this choice is made by an adversary, also known as a policy, which maps a finite execution path to a probability distribution. Once an adversary is known, the behaviour of a Markov decision process is reduced to a Markov chain. This type of model is particularly well suited to communication and distributed coordination protocols.

Similarly to DTMCs, MDPs can also be enhanced with state and transition rewards. The logic PCTL [8], including both the probabilistic and reward operators, can also be defined over Markov decision processes, the only difference being that the semantics of the probabilistic and reward operator must now include quantification over all adversaries (or adversaries of a certain type, such as the fair adversaries [6]). Model checking for PCTL over MDPs reduces to the calculation of the *minimum* or *maximum* probability of satisfying the path formula, and likewise for the reward operator. The model checking algorithm is by induction on syntax as for DTMCs, except that the probabilistic operator induces a *linear programming* problem (a simple case of a stochastic shortest path problem). The solution can be obtained using either direct methods such as Simplex, or iterative methods such as value or policy iteration [31]. The complexity for PCTL model checking over a MDP is linear in the size of the formula and polynomial in $|S|$ [8], which follows from the existence of polynomial LP solvers. For more detail see e.g. [21].

2.3.2 Continuous Time Markov Chains

Continuous time Markov chains (CTMCs) are well known in performance modelling [34] to model systems which have discrete states, but where time progresses continuously. A CTMC is represented as a *transition rate* matrix, where a positive rate λ between two states s and s' denotes that the probability of the transition from s and s' being triggered after t time units is given by a negative exponential distribution. CTMCs do not admit nondeterminism, and instead model scheduling via the *race condition.* CTMCs can be unfolded into execution paths, which are alternating sequences of states and real numbers (amount of time spent in each state). The probability measure construction over paths [19] can be generalised to this case. The reward structure is similar to the DTMC case, except that the reward can be acquired in proportion to time t spent in a state. This type of model is well suited to reliability, performance and dependability modelling.

The logic CSL (Continuous Stochastic Logic) [3, 5] has been defined over CTMCs. It is based on PCTL and contains the probabilistic and reward operators of PCTL evaluated with respect to path-based probability measure. Additionally, there is also a *steady-state* operator. The path formulae of CSL include a time-bounded until, which is similar to the bounded until of PCTL except that the time bound can be real-valued, e.g. $\mathtt{P}_{=?}[\mathtt{true}\ \mathtt{U}^{<4.25}\ \Phi]$ expresses the probability that Φ becomes true before 4.25 units of time. The untimed properties of CSL can be reduced to PCTL model checking on the embedded DTMC. Model checking for the time-bounded until operator of CSL has been shown in [5] to reduce to transient analysis and can be computed on the uniformised DTMC. The steady-state operator is verified through identification of bottom strongly connected components and solving a linear equation, and the reward operator is model checked similarly [24]. The complexity of CSL model checking for CTMCs is linear in the size of the formula, polynomial in the state space, linear in the maximum time bound in the formula, and linear in the largest number contained in the generator matrix. For more information see e.g. [5, 32, 24].

2.3.3 Probabilistic Timed Automata

Continuous time Markov chains do not allow nondeterminism which often features in real-world distributed protocols, for example random back-off schemes. Probabilistic timed automata (PTAs) [28] extend the timed automata formalism [2] with probabilistic choice over transitions. Similarly to timed automata, PTAs contain *clocks*, positive real-valued variables which increase uniformly with time, which can be referred to in its invariants and guards. A probabilistic timed automaton therefore models dense real-time passage, nondeterminism arising from scheduling and discrete probabilistic choice between transitions. This type of model frequently arises in randomised distributed coordination protocols, for example random back-off.

The presence of nondeterminism means that the semantics of a PTA is a Markov decision process, and consequently the notion of adversary is required. A probability measure over paths of a given adversary can be defined following [19] as for Markov decision processes. The logic for expressing properties of PTAs is PTCTL (Probabilistic Timed CTL) [28], derived from TCTL and PCTL, with which one can query the probability of reaching a state within a real-valued time bound. Model checking for this logic is obtained by adapting the techniques established for timed automata to the probabilistic case. Observe that, since clocks are real-valued, the state space of a probabilistic timed automaton is infinite but can be partitioned into a *finite* set of symbolic states (regions or zones). The methods so far developed for PTAs include those based on the region graph construction [28], forward [28] and backward [29] zone graph exploration and the digital clocks approach [25]. The latter admits an extension with rewards, for example expected reachability properties, a generalisation of uniformly priced timed automata. Model checking for PTAs is very expensive and experimental results are still limited with the exception of the digital clocks approach for which MDP techniques are sufficient. For more detail see [26, 29, 25]. An extension of probabilistic timed automata with *continuous* probability distributions and spaces are respectively described in [27, 9].

3. TECHNIQUES

As can be seen from the above overview of quantitative verification, a great variety of analysis techniques are required, ranging from graph-theoretical analysis of the underlying transition system graph, symbolic techniques for state-space reduction or to ensure a finite quotient, to matrix algorithms and the numerical solution techniques. Below we describe the main numerical methods used and the issues that may arise in quantitative verification.

3.0.4 Exact methods

Quantitative model checking typically proceeds by first constructing a representation of the full probabilistic model, or its part relevant for the property being verified, and then, based on the model and the property, formulating a family of numerical problems which will yield the desired quantities. For the former, an analysis of the underlying graph of the probabilistic model suffices, which can e.g. be performed via conventional fixed point computation. The latter requires the solution of linear equation or linear optimisation problem, often integrated with the representation of the state-transition graph of the model, and can be a bottleneck since the size of the problem is in the worst-case as large as the state-space. Such problems can be solved exactly with *direct methods* (e.g. Gaussian elimination or Simplex, respectively), but these are usually impractical for large systems. In suchcases, *iterative methods* (e.g. Gauss-Seidel or dynamic programming, respectively) are typically used. These converge towards the correct solution with each iteration and are terminated when convergence indicates that the desired precision has been reached. See e.g. [32] for more details.

3.0.5 Approximate methods

Instead of constructing the full state-transition graph of the model, it has been proposed [37] to use a combination of discrete event simulation and Monte Carlo methods to estimate the probability of satisfying a path formula. This is done by generating random paths of a fixed depth k and applying statistical analysis of thus obtained paths to compute the value of a random variable which *estimates* the probability that a given formula is satisfied on paths of depth at most k, for specified confidence and error bounds. This method avoids the (potentially very costly) construction of the model and its state space, and results in complexity dependent on e.g. the length of the path and confidence bound, not the size of the state space. The method is suited to DTMCs and CTMCs and can be applied to a wider range of properties, for example LTL. Known approaches based on this idea include one based on a randomised approximation scheme [17], statistical model checking [33] and statistical hypothesis testing [36].

An alternative technique is to invoke directed model checking [14] which performs a partial exploration of the state-space, extended to the probabilistic case in [1].

4. SOFTWARE TOOLS

A number of probabilistic and quantitative verification software tools exist. Our main focus is the PRISM model checker which was used to perform the case study discussed in the next section.

4.1 The Probabilistic Model Checker PRISM

PRISM [18, 30] accepts probabilistic models described in a simple, high-level modelling language. Three types of probabilistic models are supported directly; these are discrete-time Markov chains, Markov decision processes, and continuous-time Markov chains. Additionally, probabilistic timed automata are partially supported, with the subset of diagonal-free PTAs supported directly via *digital clocks* [25]. Properties are specified using PCTL for DTMCs and MDPs, and CSL for CTMCs, extended with the reward operator.

PRISM first builds a symbolic, representation of the reachable state-space of the probabilistic model as a *multi-terminal* binary decision diagram (MTBDD) [21]. The model checking proceeds as for CTL, by induction over syntax. PRISM handles both the *probability/reward bound* properties, reporting true/false outcomes when the probability is above or below the threshold, as well as the *quantitative* outcomes, reporting, for example, the actual probability of a certain event or the expectation for a reward formula. In addition to the numerical solution, PRISM also supports *sampling-based, approximate* quantitative verification based on Monte Carlo simulation [16]. Further, PRISM supports the notion of *experiments*, which is a way of automating multiple instances of model checking. This allows the user to plot the outcome of one or more properties as functions of model and property parameters.

The model size capacity of PRISM is approx. 10^7 – 10^8 for CTMCs and higher for other types of models. The underlying computation in PRISM involves a combination of *graph-theoretical algorithms*, for reachability analysis, conventional temporal logic model checking and *qualitative* probabilistic formulae; and *numerical computation*, for *quantitative* probabilistic model checking, e.g. solution of linear equation systems. Graph-theoretical algorithms are performed in PRISM using BDDs. For numerical computation, PRISM supports three numerical engines and uses iterative methods: Jacobi, Gauss-Seidel and SOR for the solution of linear equation systems, and value iteration for linear programming problems. Finally, for transient analysis of CTMCs, PRISM incorporates another iterative numerical method known as uniformisation.

PRISM is a *free*, *open source* application. It presently operates on Linux, Unix, Windows and Macintosh operating systems. The reader is invited to consult [22, 23] and the 'Case Studies' section of the PRISM website [30] for many examples of quantitative verification applied to several domains, including communication and security protocols, reliability, dependability, performance and biological systems.

4.1.1 Other Tools

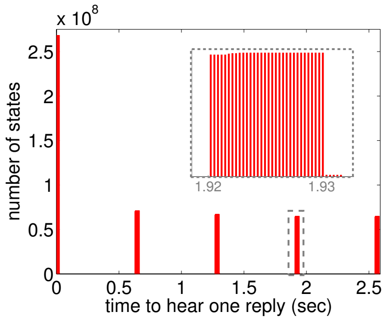
Several tools support quantitative modelling, see e.g. Mobius and PEPA. Probabilistic model checkers for MDPs include RAPTURE and ProbMela/LiQuor for LTL properties, and those tailored to DTMCs and CTMCs are MRMC and SMART. Approximate model checking tools include APMC, Ymer and VESTA. Links and references can be found at [30].

5. CASE STUDY: DEVICE DISCOVERY IN BLUETOOTH

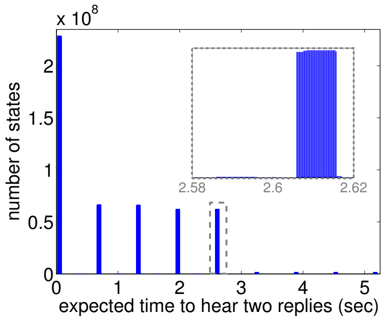
We illustrate the quantitative verification methodology by briefly describing the outcome of a modelling case study of the Bluetooth device discovery protocol standard performed with PRISM [18]. The model simultaneously serves as an illustration of the potential as well as the current limitations of the techniques described in this paper: the model derived is a DTMC and the analysis involved both timing and resource specifications. On the positive side, the model is extremely large and we were able to calculate, for the first time, the worst case expected time to receive a message. On the negative side, the complexity of the protocol is so great that the largest configuration we studied included two devices and two messages only.

5.1 Modelling Bluetooth Device Discovery

Bluetooth is a short-range, low-power, open standard for implementing wireless personal area networks which uses a frequency hopping scheme, where devices alternate rapidly among the 79 available frequencies in a pseudo-random fashion, to avoid interference from other devices, such as microwave ovens and other phones, operating in the same frequency band.



(a) distribution over initial states (one reply)



(b) distribution over initial states (two replies)

Figure 2: Expected time for the sender to hear one or two replies from the receiver

In order to communicate, Bluetooth devices organise themselves into small networks called *piconets*, comprising one master and up to 7 slave devices, in which the frequency hopping sequences are synchronised and controlled by the master. Our study focused on the issue of piconet creation, and specifically the *inquiry* process, and its effect on performance and power consumption.

Each Bluetooth device has a 28 bit free-running clock, which ticks every $312.5\mu s$. An inquiring device attempts to detect potential slaves by broadcasting inquiry packets on a previously agreed sequence of 32 of the 79 available frequencies for two consecutive time slots and scanning for replies on the next two slots.

Bluetooth devices that want to be discovered periodically scan for inquiry packets on the same 32 frequencies that the inquiring device is transmitting on. To ensure that the frequencies used eventually coincide and that messages are successfully received, the hopping rate of scanning devices is much slower than that of the inquiring device, i.e. changes every 1.28s alternating between a sleep and a scan phase.

If the scanning device successfully hears a message, by listening on the right frequency at the right time (when the inquiring device is transmitting a packet), it will enter a reply phase, in which it waits 2 time slots (i.e. $625\mu s$) and then sends a reply on the same frequency. A contention problem arises when two devices in inquiry scan try to reply to the same inquiry packet, thus potentially leading to collision and loss of the replies. To avoid repetition of such a problem, after sending a reply, a device invokes a *random back-off* by drawing a random number $N \in [0, \ldots, 127]$ and waiting for $2{\cdot}N$ time slots before going back to its alternation between sleep and scan states.

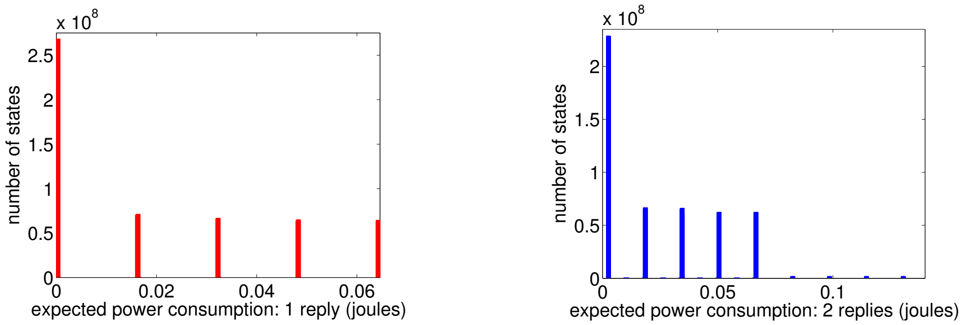
We derived a PRISM model of the above process by considering a single inquiring device and a single scanning device, referred to as the *sender* and *receiver*, respectively. Note that, despite the limitations of this scenario, the randomised back-off procedure must be executed. The clocks of both devices are digital, whose time is incremented in discrete steps, corresponding to $312.5\mu s$ *slots*, and whose drift can be assumed to be negligible during the relative short inquiry process. This is consistent with the constraints imposed upon possible clock drift in the Bluetooth specification.

From the assumptions of digital clocks and additional simplifying assumptions to reduce the state space, we obtain a discrete-time Markov chain model as the synchronised parallel composition of the sender and receiver components. In our model the resources we measure are time and power consumption. For example, in the case of time, we assign a cost of 1 to all time-step transitions and 0 to all others. As an indication of the model size and complexity, the number of possible *initial* states is $17,179,869,184$, with the full state space much larger. The model description can be found at [30] and the approach taken to reduce the model complexity is described in [13].

5.2 Experimental Results

We perform an exhaustive quantitative verification of the resulting DTMC model, extended with rewards, using PRISM. Since we are primarily interested in performance and power consumption of the Bluetooth inquiry process, we compute the expected time for the inquiry process to receive a specified number of replies. We also compute the corresponding expected power consumption. In view of the prohibitive size of the state space, we partition the verification into 32 sets of $536,870,912$ states each, thus reducing the problem to 32 separate instances of model checking, each of which is now feasible on a standard workstation. This still results in models with extremely large state spaces, which is a direct consequence of the complexity of the interaction between the Bluetooth devices attempting to discover each other. Thanks to a degree of regularity in the model, we were able to build and analyse the 32 models using MTBDDs. With an approach based on explicit data structures (e.g. sparse matrices) this would not be feasible.

We computed the expected time to send *one* (and later also *two*) message for all possible initial configurations, expressible as a reachability reward formula $\mathtt{R}_{=?}\ [\ \mathtt{F}\ \Phi\ ]$ which corresponds to the expected cumulated cost (in this case, time) of the system until condition Φ is satisfied. The condition Φ identifies states of the model where a sufficient number of replies (in this case, one) have been received. Using PRISM, we were able to extract, for the first time, information about the best- and worst-case scenarios over *all* initial states of the model. This would not have been possible with simulation. We can also use PRISM to identify precisely



(a) distribution over initial states (one reply)

(b) distribution over initial states (two replies)

Figure 3: Expected power consumed before the sender hears one or two replies from the receiver

the way in which this maximum time arises by automatically generating a path. The 32 models we constructed each contained approximately 3.4×10^9 states, required less than a minute to construct, and took 1–2 seconds to analyse.

Our results show that the minimum time for a single reply is 625μs (2 slots), while the maximum time is 2.5716s (8,229 slots) and is achieved in 860,160 of the possible initial states. In Figure 2(a) we have plotted the time until the sender hears a reply against the number of initial states that result in this time. The discontinuities in the graph are to be expected in view of the sleep periods. The inset in Figure 2(a) illustrates one of these peaks more clearly. The width of each peak is 11.25ms (36 slots).

We also analysed the scenario where the sender waits until *two* replies have been received. In this case, the 32 constructed models each have approximately 5.6×10^{10} states and took roughly 80 minutes to build and 165 minutes to model check. The minimum expected time, over all possible initial configurations, for the sender to hear two replies is 0.0456s (146.0 slots). The maximum is 5.177 seconds (16,565 slots) and 518 of the possible initial states result in this.

Finally, we performed an analysis of the expected power consumption of the Bluetooth device discovery process. To do so, we need only change the costs associated with transitions of the model from the elapsed time to the power consumed. Taking 100mW for active mode in Inquiry-Scan and 50mW for standby, values which are consistent with the standard, the results obtained are presented in Figure 3. These are very similar to the expected time results, with smaller gaps between two peaks (as less power is consumed during sleep).

6. CONCLUSIONS

In this paper we presented an overview of quantitative model checking for models which are variants of Markov chains. Algorithms were given for verifying these models against probabilistic temporal logic PCTL and its extension with the reward operator. Finally, a real-world protocol analysed with the probabilistic model checker PRISM was described.

There are many related topics covered in the literature which we have not been able to cover in this paper: more case studies, more expressive specification formalisms, state-space reduction techniques such as symmetry reduction, partial order reduction and bisimulation quotient, probabilistic verification for mobility, counter-examples and generation of models from design notations such as UML. See [30] for some pointers to these topics.

Finally, we mention some future challenges for quantitative verification: automated extraction of quantitative models from source code, compositional quantitative reasoning, approaches for general probability distributions and efficient parallelisation techniques.

7. REFERENCES

[1] H. Aljazzar and S. Leue. Extended directed search for probabilistic timed reachability. In *FORMATS'06*, volume 4202 of *LNCS*, pages 33–51. Springer, 2006.

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

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Proc. CAV'96*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[4] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proc. ICALP'97*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[5] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Proc. CONCUR'99*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

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

[7] D. Bertsekas. *Dynamic Programming and Optimal Control, Volumes 1 and 2*. Athena Scientific, 1995.

[8] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[9] S. Cattani, R. Segala, M. Kwiatkowska, and G. Norman. Stochastic transition systems for continuous state spaces and non-determinism. In *Proc.*

FOSSACS'05, volume 3441 of *LNCS*, pages 125–139. Springer Verlag, 2005.

[10] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logics. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[11] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. FOCS'88*, pages 338–345. IEEE CS Press, 1988.

[12] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[13] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[14] S. Edelkamp, S. Leue, and A. Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *STTT*, 5(2-3):247–267, 2004.

[15] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[16] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. In C. Priami, editor, *Proc. Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Bioinformatics*, pages 32–47. Springer Verlag, 2006.

[17] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proc. VMCAI'04*, volume 2937 of *LNCS*. Springer, 2004.

[18] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[19] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[20] M. Kwiatkowska. Model checking for probability and time: From theory to practice. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 351–360. IEEE CS Press, 2003. Invited Paper.

[21] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[22] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: Case studies with PRISM. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):16–21, 2005.

[23] M. Kwiatkowska, G. Norman, and D. Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2005.

[24] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[25] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.

[26] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Verification of real-time probabilistic systems. In S. Mertz and N. Navet, editors, *Modelling and Verification of Real-Time Systems*, 2007. To appear.

[27] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In C. Palamidessi, editor, *Proc. CONCUR 2000 - Concurrency Theory*, volume 1877 of *LNCS*, pages 123–137. Springer, 2000.

[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.

[29] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 2007. to appear.

[30] PRISM web site: www.prismmodelchecker.org.

[31] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[32] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, P. Panangaden and F. van Breugel (eds.)*, volume 23 of *CRM Monograph Series*. AMS, 2004.

[33] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *Proc. CAV'05*, volume 3576 of *LNCS*, pages 266–280. Springer, 2005.

[34] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[35] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE CS Press, 1985.

[36] H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

[37] H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. CAV'02*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.