

Practical Applications of Probabilistic Model Checking to Communication Protocols

Marie Duflot¹, Marta Kwiatkowska², Gethin Norman², David Parker², Sylvain Peyronnet³, Claudine Picaronny⁴, and Jeremy Sproston⁵

¹ LACL, Faculté des Sciences et Technologie, Université Paris XII, 61 Avenue du Général de Gaulle, 94010 Créteil, France

`duflot@univ-paris12.fr`

² University of Birmingham, Birmingham B15 2TT, United Kingdom
{M.Z.Kwiatkowska,G.Norman,D.A.Parker}@cs.bham.ac.uk

³ EPITA Research and Development Laboratory (LRDE), 14-16 rue Voltaire, F-94276 Le Kremlin-Bicêtre, France

`syp@lrde.epita.fr`

⁴ LSV, CNRS & ENS de Cachan, 61 av. du Pres. Wilson, 94235 Cachan, France
`picaro@lsv.ens-cachan.fr`

⁵ Dipartimento di Informatica, Università di Torino, 10149 Torino, Italy
`sproston@di.unito.it`

Abstract. Probabilistic model checking is a formal verification technique for the analysis of systems which exhibit stochastic behaviour. It has been successfully employed in an extremely wide array of application domains including, for example, communication and multimedia protocols, security and power management. In this chapter we focus on the applicability of these techniques to the analysis of communication protocols. An analysis of the performance of such systems must successfully incorporate several crucial aspects, including concurrency between multiple components, real-time constraints and randomisation. Probabilistic model checking, in particular using probabilistic timed automata, is well suited to such an analysis. We provide an overview of this area, with emphasis on an industrially relevant case study: the IEEE 802.3 (CSMA/CD) protocol. We also discuss two contrasting approaches to the implementation of probabilistic model checking, namely those based on numerical computation and those based on discrete-event simulation. Using results from the two tools PRISM and APMC, we summarise the advantages, disadvantages and trade-offs associated with these techniques.

1 Introduction

Computer-controlled devices are today ubiquitous in many areas of industry, including business- and safety-critical domains. For this reason, the use of communication protocols, devised to govern the interactions between such devices, is extremely widespread. Recent years have also seen considerable growth in the development and deployment of formal verification methods, used to establish

the correctness of and identify faults in a wide array of real-life systems, a fact evidenced by the breadth of topics covered in this volume. In this chapter, we describe work which has been carried out to apply formal verification techniques to communication protocols. Such systems represent a particularly challenging case in this respect because, in order to successfully model and analyse them, several key aspects must be incorporated: *concurrency* between multiple components, potentially operating in different locations and at unknown speeds; precise, *real-time* constraints, imposed by the protocols and the mediums under which they are designed to operate; and *randomisation*, which is often used to break symmetry between devices communicating using the same protocol.

Probabilistic model checking is a formal verification technique for the analysis of systems which exhibit stochastic behaviour. It involves the construction of a probabilistic model of some real-life system followed by a mathematical analysis of this model in order to determine useful properties of the original system. Unlike conventional verification techniques, probabilistic model checking can be used to ascertain not only correctness, but also quantitative measures such as performance and reliability. These techniques can be applied to a range of probabilistic models, typically variants of Markov chains, but of particular relevance here are their application to *probabilistic timed automata* (PTAs), a model which can ably express nondeterministic, real-time and probabilistic behaviour. In this chapter, we give an overview of probabilistic timed automata, probabilistic model checking, and the corresponding implementation techniques and tools. Furthermore, we illustrate their usefulness in the domain of communication protocols through a case study: the IEEE 802.3 (CSMA/CD) protocol, as used for example in networking over Ethernet.

2 Probabilistic timed automata

Probabilistic timed automata (PTAs) are a formalism for modelling systems whose behaviour exhibits nondeterministic, real-time and probabilistic characteristics [19]. PTAs are an extension of timed automata [3], one of the most prominent formalisms for the formal verification of real-time systems (see Chapter ? for more on this topic).

In this section, we illustrate a number of basic concepts of (probabilistic) timed automata using the example in Figure 1. The figure shows an automaton modelling a simple communication protocol, in which a station attempts to transmit onto a bus. If the station's transmission is interrupted by a transmission from another station (a collision), then the station suspends its activity, waiting a randomly chosen amount of time, before starting to send its message again. The *control states* of the model, TRANSMIT, WAIT, and DONE, are shown by the nodes of the graph, and the possible transitions between the control states are indicated by the graph's edges. In the initial state, TRANSMIT, denoted by the double border, a transmission is being made by the system. When 5 time units have elapsed from the start of transmission, the message has been sent

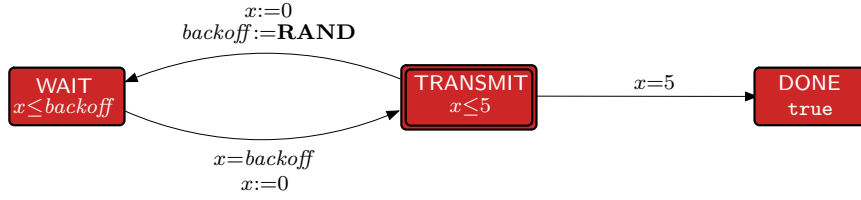


Fig. 1. An example of a probabilistic timed automaton

successfully, and the model moves to state DONE. Dependency on time can be represented by *clocks*, *guards* and *resets* on edges, and *invariants* in states. A clock is a real-valued variable which has the value 0 at the start of execution of the system, and which increases at the same rate as real-time while the model remains within the same control state. The illustrated model has a single clock, x . The guard of the edge from TRANSMIT to DONE is $x = 5$, and specifies that the edge can be traversed when the value of the clock x equals 5. The invariant of the state TRANSMIT is $x \leq 5$, and specifies that control must pass from TRANSMIT before the value of the clock x exceeds 5.

The edge from TRANSMIT to WAIT is not labelled with a guard, which is interpreted as denoting that the edge can always be traversed from TRANSMIT, regardless of the value of the clock x . Whether this edge is taken or not, at any point in time in which the automaton is in the state TRANSMIT, is a *non-deterministic* choice. The traversal of this edge corresponds to interruption of the transmission caused by the simultaneous transmission on the bus by another station in the network. The edge features a clock reset denoted by $x := 0$, which indicates that, on entry to state WAIT, the clock x is reset to the value 0. In addition to clocks, we also allow finite-domain variables to be referred to in resets, guards and invariants; furthermore, we also allow *probabilistic resets* of such variables, in contrast to the case of clock resets, which assign 0 deterministically to clocks. On traversal of the edge from TRANSMIT to WAIT, the finite-domain variable *backoff* is set to a random value according to the assignment $backoff := \mathbf{RAND}$ (where \mathbf{RAND} is a probability distribution over a number of possible values of *backoff*, for example the uniform distribution over the natural numbers between 3 and 10). Furthermore, the value of *backoff* is subsequently used within the invariant of the state WAIT, and the guard of the edge from WAIT to TRANSMIT, in order to ensure that, when the value of the clock x reaches the value of *backoff*, control returns to the state TRANSMIT.

The example of Figure 1 is an example of a PTA. We note that replacing the probabilistic assignment $backoff := \mathbf{RAND}$ by a non-probabilistic assignment results in a standard timed automaton. Although not illustrated by the example, probabilistic timed automata can also be extended with a set of *events*, which label edges. These events can then be used to define the *parallel composition* of a number of PTAs, where each such automaton is regarded as a sub-component

of the overall systems, and where the automata synchronise on shared events. Such events will be used in Section 4 to define the parallel composition of PTAs of a bus and two stations operating according to the CSMA/CD protocol. The parallel composition operator was introduced in Kwiatkowska et al. [21], based on precedents for timed automata [3] and probabilistic transition systems [25].

The semantics of a PTA are formally defined as an infinite-state Markov decision process (MDP), a probabilistic model which supports nondeterministic choice, and in which transitions are made in discrete steps. The MDP will generally comprise infinite states because the clocks of the PTA are real-valued variables. Similarly, the transitions available in a state in which time can elapse will also be infinite in number because these correspond to real-valued durations. Analysis of a PTA, which typically constitutes formal verification of one or more properties specified in temporal logic, is therefore non-trivial because model-checking algorithms usually operate on finite-state models. However, based on analogous work on model checking of (non-probabilistic) timed automata, we can obtain faithful finite-state representations of PTAs, which can be used in the analysis of a wide range of correctness properties or performance indices. One example of such a representation is based on *digital clocks*, in which the clocks of the PTA are interpreted as taking integer values, rather than real values [17]. This approach requires that the comparisons involving clocks in guards and invariants are non-strict; however, this is a common property in PTA models of real-life systems, such as the CSMA/CD protocol considered in this chapter. The digital-clocks approach has been, to date, the most successful in practice of the finite-state representations proposed for PTAs: it has been used to verify the CSMA/CD protocol [7, 22], part of the IEEE 802.11 standard for wireless local area networks [20, 23], the FireWire root contention protocol [21], and the IPv4 Zeroconf protocol [17].

3 Probabilistic model checking

Probabilistic model checking is a formal verification technique for the modelling and analysis of systems which exhibit stochastic behaviour. It can be applied to several different types of probabilistic model. The three most commonly used are: discrete-time Markov chains (DTMCs), in which time is modelled as discrete steps, and randomness as discrete probabilistic choices; Markov decision processes (MDPs), which extend DTMCs with the ability to represent nondeterministic behaviour; and continuous-time Markov chains (CTMCs) which do not permit nondeterminism but allow specification of real (continuous) time behaviour, through the use of exponential distributions. See Chapter ? for information regarding model checking of DTMCs and CTMCs. In this chapter, we focus principally on MDPs since, as described in the previous section, in many cases analysis of PTAs reduces to analysis of MDPs. We also refer to DTMCs, which can be seen as a special case of MDPs where nondeterminism is absent.

Properties of DTMCs and MDPs are usually expressed in the temporal logic PCTL [11, 6]. We omit here the details of this logic (see e.g. [24] for more information). PCTL can be used to reason about quantities such as:

- “the probability the protocol eventually terminates”
- “the probability that a message is successfully received within $35\mu\text{s}$ ”

In the case of MDPs, it is in fact not possible to compute exact probability values such as these due to the presence of nondeterminism in the model. In this situation, the values computed are the *minimum* or *maximum* probabilities over all possible resolutions of nondeterminism. This corresponds to a *best-case* or *worst-case* analysis (depending on the meaning of the probabilities). Furthermore, it is also often useful, both for DTMCs and MDPs, to compute the minimum or maximum probability over a range of possible configurations or parameters of a model, producing a different kind of best/worst-case analysis. Finally, we note that it is also possible to augment DTMCs and MDPs with real-valued *costs* or *rewards*, which can represent a wide range of measures, e.g. “system power consumption”, “number of lost messages” or “message queue size”. It is then possible to reason about the (possibly minimum or maximum) *expected* value of these measures, e.g.:

- “the expected power consumption during the first 20s of system operation”
- “the expected number of messages lost before protocol termination”
- “the expected number of messages queued for delivery after $500\mu\text{s}$ ”

3.1 Techniques for probabilistic model checking

Numerical solution. The conventional approach to probabilistic model checking is to construct a representation of the entire probabilistic model, and from this derive one or more numerical problems which will yield results for the properties of the model to be analysed. The first phase of this process often requires only an analysis of the underlying graph of the probabilistic model (e.g. reachability-based techniques). The remainder (which often represents the bottleneck in terms of efficiency) requires numerical computation to be performed. Commonly, solution of either a linear equation system (for DTMCs) or a linear optimisation problem (for MDPs) is required. Although such problems can be solved exactly with *direct methods* (e.g. Gaussian elimination or Simplex, respectively) for efficiency reasons, *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. [24] for more details.

Approximate probabilistic computation. An alternative approach is to use a combination of discrete event simulation and Monte Carlo methods to

estimate the probability associated with a PCTL formula [13]. This is done by generating random paths of depth k in a DTMC and computing the value of a random variable which estimates $Prob_k[\psi]$, the probability that a formula ψ is satisfied on paths of depth at most k . More specifically, the algorithm which performs this process takes as input a succinct representation of DTMC x , a formula, a positive integer k and two parameters ε and δ , producing a value $A(x, \varepsilon, \delta)$. This is a *fully polynomial randomised approximation scheme*, meaning that the result satisfies $Prob[|A(x, \varepsilon, \delta) - \mu(x)| \leq \varepsilon] \geq 1 - \delta$, where $\mu(x)$ is the exact value of $Prob_k[\psi]$. We call ε the *approximation parameter* and δ the *confidence parameter*. The algorithm must generate $O(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta})$ paths through the DTMC. The main advantage of this approach is that, because random paths can be generated from only a succinct representation of the DTMC, it avoids the (potentially very costly) construction of the model and its state space, hence using a very small amount of memory. Furthermore, the approach can be applied to a wider range of properties of DTMCs, e.g. using the linear time temporal logic LTL rather than PCTL. A related technique based on *acceptance sampling* and *hypothesis testing* [27] also uses random sampling in conjunction with discrete event simulation but is tailored to checking whether the probability of satisfying a formula meets a given bound.

3.2 Probabilistic model checking tools

PRISM. PRISM [14, 2] is a probabilistic model checker developed at the University of Birmingham. It provides support for analysis of DTMCs, MDPs and CTMCs. Models are described in the PRISM modelling language, a relatively simple, state-based language based on the Reactive Modules formalism [4], and properties are specified in a logic which incorporates PCTL, CSL (a variant of PCTL for CTMCs) and a number of custom extensions. The latest versions of the tool also support modelling and analysis of costs and rewards. One of the key features of PRISM is its *symbolic* implementation techniques using data structures based on binary decision diagrams (BDDs). These allow compact representation and efficient manipulation of extremely large probabilistic models, by exploiting structure and regularity derived from their high-level description. Such techniques have been successfully applied to the probabilistic verification of models with as many as 10^{10} states (see for example [18, 8]).

APMC. APMC (Approximate Probabilistic Model Checker) [13, 1] is a distributed model checker developed in a joint collaboration between LRDE/EPITA and the universities of Paris VII and Paris XI. It uses an efficient Monte-Carlo method to approximate satisfaction probabilities of PCTL properties of DTMCs, as described in the previous section. The tool comprises two parts. The first is a compiler that produces an ad-hoc verifier (including a path generator and a checker) for a DTMC, described in the PRISM language, and a property. APMC implements different strategies to generate the code of this program with respect

to the synchronisations of the Reactive Modules: the most efficient is called *sync at compile-time* which pre-computes all the combinations of rules, thus building the synchronised succinct model representation.

The second component is a deployer that takes the ad-hoc verifier and a set of available computing resources, deploys the verifier on these computers and collects the resulting approximate satisfaction probability for the formula on the model. APMC implements a massively (but natural) distributed method of model checking. The deployment is performed following a tree topology in order to make the deployment efficient and scalable. For instance, this provides a logarithmic latency to aggregate the results from all nodes to the root. A drawback of this topology is that the system may over generate and verify some samples but it ensures that there is no point of contention in the system. More information about the implementation of APMC can be found in [10].

Other tools. Other implementations of PCTL model checking for MDPs include RAPTURE [15], which is aimed at iterative abstraction and refinement techniques, and ProbMela/LiQuor [5], which checks LTL properties of MDPs building on existing LP (linear programming) libraries. Other tools which include support for probabilistic model checking of either DTMCs or MDPs include $E \vdash MC^2$, MRMC, the APNN-Toolbox and Ymer.

4 Case study: CSMA/CD

In this section, we present an illustrative case study of a randomised communication protocol, analysed using probabilistic timed automata and probabilistic model checking. We use the Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol, which is a fundamental part of the IEEE 802.3 international standard (Ethernet Network Communication protocol).

This protocol has been extensively studied with a variety of methods: simulation [26], analytical methods [9, 16], and real-time model checking [28]. The first accurate formal analysis with respect to the probabilistic aspects of the protocol was undertaken in [7] and [22], where the system was modelled by a probabilistic timed automaton and the correctness of the probabilistic behaviour of the protocol verified by analysing minimum and maximum probabilities for reachability and time-bounded reachability properties.

The two papers take differing approaches to tackling the state space explosion problem commonly encountered in probabilistic (and real-time) model checking. In [7], time is discretized (using the digital clocks approach, mentioned in Section 2) and the resulting model is analysed using PRISM [14, 2] and APMC [13, 1]. In [22] the probabilistic timed automaton model is analysed using a probabilistic extension of the model checking approach for classical timed automata [12], implemented in a prototype extension of PRISM.

4.1 The protocol

CSMA/CD is a distributed network arbitration protocol by which multiple Network Interface Cards (NICs), referred to here as *stations*, may communicate over a single channel, known as the *carrier* or *bus*. All stations can send messages onto the network (*multiple access*) and each station can detect whether the bus is idle or is transmitting a message from another station (*carrier sense*). When a station senses that the bus is busy, it will wait before trying to transmit its own message. However, because of the propagation delay of signals across the network, it is possible that stations will try to transmit messages simultaneously. When this happens, a collision occurs, both messages are lost and the stations receive a garbled signal. In this way, the stations are able to observe that messages have been lost (*collision detection*). According to CSMA/CD, after this situation has occurred, stations reschedule their own transmission by independently choosing a random delay (known as *backoff* time), before which retransmission will be attempted. This random backoff time is chosen uniformly over an interval whose length increases exponentially with the number of collisions that have already occurred. We focus here on the half duplex version of the protocol where only one message can be carried at a time.

4.2 The PTA model

The model of the CSMA/CD protocol used here is a combination of the models from [28, 7, 22] and considers the case of two stations. The PTA comprises three components operating in parallel, representing the two stations and the bus. The model has two parameters, σ and λ , representing the time taken for data to propagate along the bus and the time required for an entire message to be sent.

The PTA for station i (where $i = 1, 2$) is shown in Figure 2. Each station has a clock x_i . It starts in the INIT state and tries to reach the DONE state where the message has been correctly sent to the other station. When the station wants to send a message (after a certain delay *delay*), it moves to state TRANSMIT. If no collision is detected within time λ , then the message will be delivered correctly and the station moves to DONE. Otherwise, a collision is detected (label cd_i) within σ time units, and the station moves to the COLLIDE state. It then draws a random waiting delay and waits for the corresponding time. When the waiting time is over, there are two options: if the bus is free, the station tries resending the message. If the bus is busy, the station increases the value of the collision counter bc (up to a maximum value K) and selects another random delay.

The PTA for the bus connecting the two stations is shown in Figure 3. The bus starts in state INIT and moves to state TRANSMIT when one of the stations begins sending a message. If nothing else happens, the station will finish its transmission (label end_i). In the state TRANSMIT, the second station can only send a message before time σ , which represents the propagation time on the bus. After this time, the station senses that the bus is busy and does not send. The

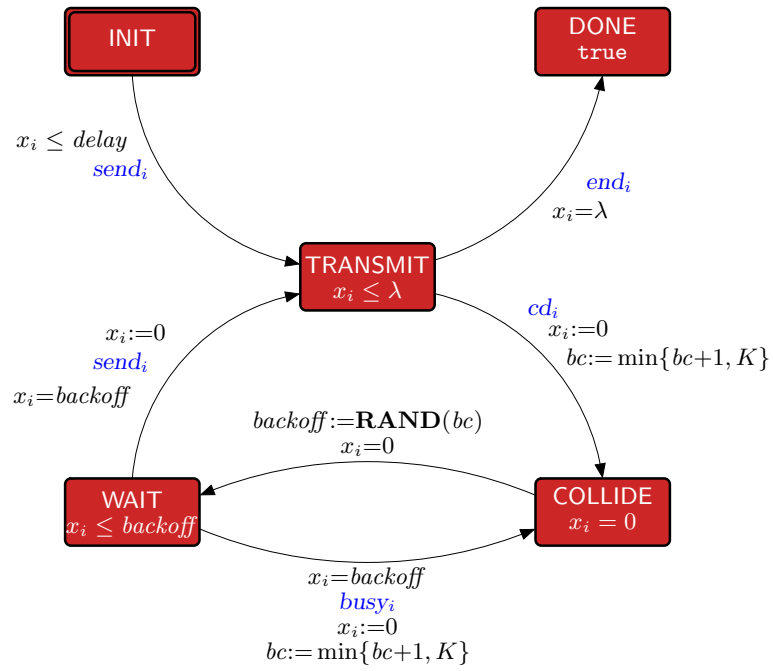


Fig. 2. Probabilistic timed automaton model of a station

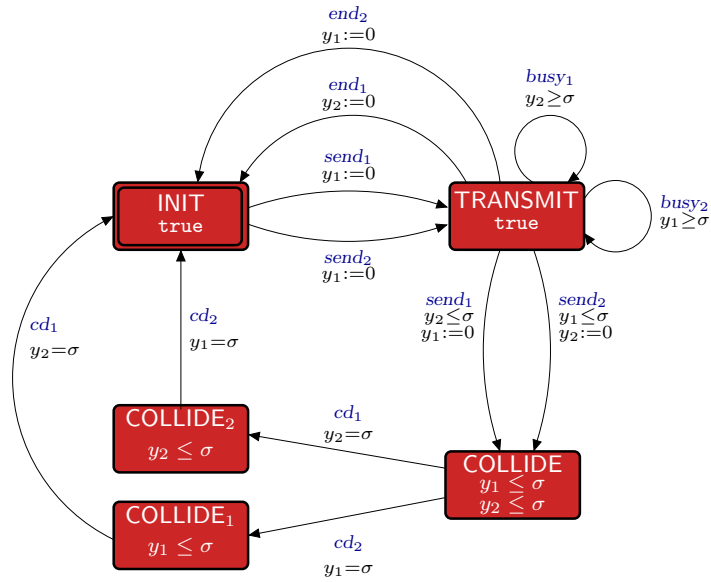


Fig. 3. Probabilistic timed automaton model of the bus

bus uses the clocks y_1 and y_2 to implement this process. If two sending actions occur within time σ , then the bus moves to the COLLIDE state. When receiving the other station’s message, both stations will detect the collision, stop sending, and the bus will return to the INIT state.

4.3 Analysis of the model

The PTA model from the previous section has been analysed using the digital clocks approach, mentioned in Section 2. Probabilistic model checking of the resulting MDP was performed with the two tools PRISM and APMC. For the CSMA/CD model parameters σ and λ , we used values of 2 and 65, respectively. The choice of the delay before which stations initially send a message (*delay*) is assumed to be nondeterministic (either 0, 1 or 2) for the PRISM analysis to ensure that all possibilities are considered. Using APMC, this is not possible so the choice is randomised.

Experiments with PRISM were run on a 2.6GHz Pentium IV laptop with 1GB RAM running Red Hat Linux. Iterative numerical computations were terminated when the maximum relative error between elements of solution vectors from successive iterations dropped below 10^{-6} .

APMC experiments were performed on a heterogeneous grid of 500 Athlon 3000+ workstations with 1GB RAM running NetBSD and 20 3GHz Pentium IV workstations with 512MB RAM mostly running Debian Linux, connected via 100MB Ethernet. APMC was executed as a “cycle stealer”, i.e. running in the background of the workstations, whilst they were used by students and staff at EPITA. Hence, timing statistics for the cost of verification are imprecise. However, all experiments were completed in two days during a period of low average load on the network. In all cases, the most efficient APMC strategy “sync at compile-time” was used and, unless specified otherwise, the approximation parameter ε and confidence parameter δ were fixed at $\varepsilon = 10^{-2}$ and $\delta = 10^{-10}$, respectively.

Probability of message transmission. The first property we consider is the probability that a message has been successfully sent by a certain time point T . In Figure 4, we show results computed for three values of the maximum backoff limit K (1, 5 and 10; the value specified in the standard is 10) and a range of values of T . We give the *maximum* and *minimum* transmission probability over all nondeterministic choices, as computed by PRISM, and the *average* values, assuming a uniform choice over initial delays, as computed by APMC.

The results show that using larger values of K increases both the minimum, maximum and average probabilities of sending a message by time T . Since we have constructed our model in such a way that the stations initially collide, the probability is 0 for any time bound T which does not allow the stations to enter backoff and send a message.

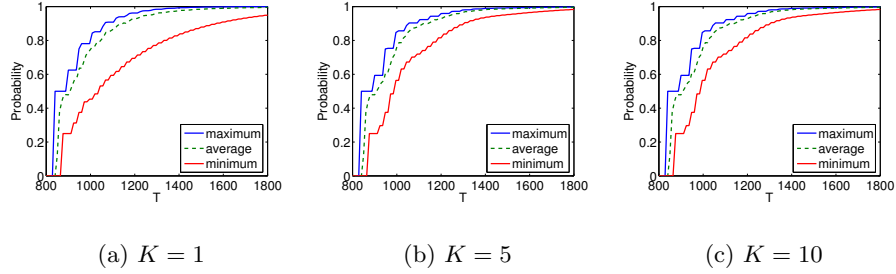


Fig. 4. Probability that a message is sent by time T

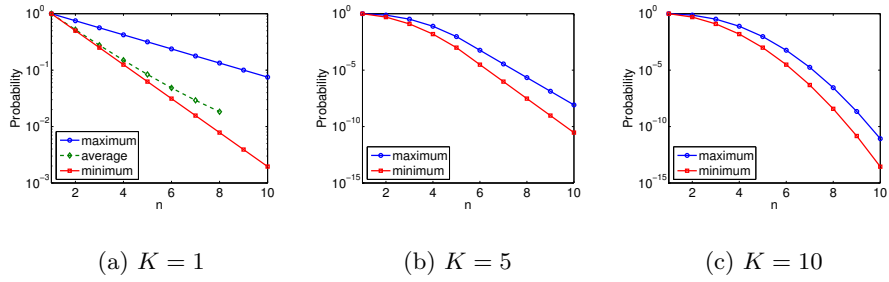


Fig. 5. Probability that the stations collide n times before a message is sent

Notice that the exact plots in the graphs (minimum and maximum) contain discrete jumps. These correspond to the fact that the probabilistic choice of backoff delay is over a discrete set of delays. The plots for the average case are smoother and, as expected, contained within the boundaries of the minimum and maximum values.

Collision probabilities. Secondly, we consider the probability that n collisions occur before a single message is successfully sent. Results are shown in Figure 5, again for three values of the maximum backoff limit K , and for n ranging between 1 and 10. As before, we show both the minimum and maximum probability, computed by PRISM. Where possible, we also show the average case, as computed by APMC. However, in the majority of cases, computation of these values is not practical due to their small size. Recall that the number of samples required is quadratic in the inverse of the desired approximation. For the set of results corresponding to $K = 1$ shown in the figure, we had to use a precision of 10^{-3} , which required approximately 2×10^9 samples. As is clear from the other graphs in Figure 5, larger values of K would require considerably greater precision and hence an infeasible number of samples.

Note that, since our model is constructed in such a way that the stations always collide initially, for K equal to 1, 5 and 10, the minimum and maximum probability of one collision occurring ($n = 1$) is 1. In all three cases, the results for $n = 2$ also agree, as do those for $n \leq 6$ where K is 5 or 10. This is to be expected since the stations' backoff counters cannot reach $n+1$ until the stations have collided $n+1$ times. Taking as an example the case where $n = 1$, we can illustrate the reasoning behind the results as follows.

- The minimum probability equals 0.5 and corresponds to the situation when the stations begin their first attempt at sending messages at the same time (with no constraint on the propagation delay). In this scenario the stations will detect the first collision at the same time, and therefore the probability that they collide a second time is the probability that the stations select the same number of slots to wait. Hence, since this is a random choice between waiting 1 or 2 slots, the stations collide a second time with probability 0.5.
- The maximum probability equals 0.75 and corresponds to the case when the propagation delay equals σ and the difference between when the stations initially try to send their messages is also σ time units. In this case, the second station to attempt transmission will detect this collision σ time units before the other, and therefore the stations will collide a second time if either the stations decide to wait the same number of slots, or the first station waits for two slots while the second station decides to wait one slot, i.e. the probability of colliding a second time is 0.75.

In general, as n increases, and since the range over which a station selects its backoff grows exponentially each time the stations collide, while $n \leq K+1$ we see that the probability that the station collide n times declines rapidly. However, once $n > K+1$ the stations' backoff counters are no longer incremented after a collision (they have reached their maximum value K), and therefore the range of values over which the stations choose to “backoff” remains the same. Hence, the chance that they collide no longer falls as rapidly.

Expected costs. Finally, we consider two properties that can be analysed by augmenting our probabilistic models with costs. First, we assign a cost of 1 to each transition in our model which corresponds to a collision and then compute the expected number of collisions that will occur for different values of the maximum backoff limit K . These results (again, minimum and maximum values, over all resolutions of nondeterminism) are shown in Figure 6(a). Second, we assign a cost of 1 to each transition which represents a discrete time-step and then compute the expected time required for message sending to complete. These results can be found in Figure 6(b). In principle, it would certainly be possible to also generate average values for these properties using discrete event simulation and Monte Carlo methods, as described above. However, these techniques are not yet supported in APMC and so we do not include such results.

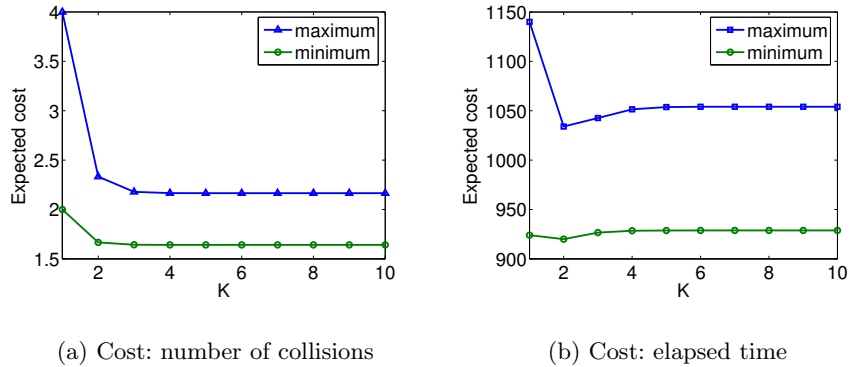


Fig. 6. Results for expected cost until a message is sent

We see that both the expected number of collisions and expected time initially decrease as K increases. This results from the fact that, as K is incremented, there is less chance of the stations colliding when they attempt to retransmit and a message will be sent sooner. For the expected number of collisions, there is very little difference in the results once K is greater than 2. This is because the probability of colliding more than three times is very small (see Figure 5). A similar situation exists for larger values of K in the graph for expected time. Note, though, that the expected time increases between $K = 2$ and $K = 4$ and is at its smallest for $K = 2$. This is because, although the probability of 3 or 4 collisions occurring is low, the larger amount of time spent in backoff in these cases increases the impact on the expected time.

5 Discussion and Conclusion

In this chapter, we have illustrated the applicability of formal verification to the analysis of communication protocols, in particular using probabilistic timed automata and probabilistic model checking, either with numerical solution methods or approximate techniques based on Monte-Carlo simulation and sampling. We conclude with a discussion comparing these two approaches.

We first discuss the types of analysis which can be performed with each. The main strength of the numerical solution approach is that it is based on the full model, constructed via an exhaustive search of the state space. This means that it is possible to derive exact answers to temporal logic property queries (in fact, as discussed earlier, iterative numerical solution methods are usually used to construct an approximation up to the desired precision, but exact methods are always available). Furthermore, this exhaustive approach makes it possible to compute best- and worst-case results, for example over all possible initial config-

urations for a model or all possible resolutions of nondeterminism (representing e.g. concurrent schedulings between processes or multiple values for an unspecified model parameter).

By comparison, sampling-based techniques are inherently approximate, and the results represent a notion of average behaviour. However, the methods are applicable to a far wider array of models and properties. Unlike numerical solution, which can only be used where tractable solution algorithms are available, sampling can be applied to any model on which a random simulation can be accurately performed. Similarly, any property which can be computed based on finite paths through the model can be analysed in this way.

Another important area of comparison is the relative efficiency of the two approaches. Clearly the improved accuracy and coverage of numerical solution comes at a cost. One of the principal advantages of sampling techniques is that the amount of memory required to execute them is fractional compared to exact approaches. This is because they can be performed on a succinct representation of the model (e.g. its description in some high-level modelling formalism) and the simulation process typically needs only to store a single state of the model at any time. This allows sampling to be performed on much larger models than numerical solution, where the exhaustive exploration and construction of the model could easily be infeasible. In the CSMA/CD case study presented earlier, for example, it would certainly be possible to consider larger model configurations with more than two stations.

A second advantage of the sampling is that it is significantly easier to implement in a parallel or distributed setting, which can of course dramatically improve the run-time. This is because the computations of each individual sample are independent, unlike parallelisations of numerical solution, for which large amounts of process intercommunication are necessary. In fact, this performance advantage is rather important since, as we have seen earlier, to obtain accurate results it may be necessary to generate an extremely large number of samples. In particular, this situation cannot be avoided when the actual values being computed are very small.

Overall, it is clear that both exact approaches to probabilistic model checking, based on exhaustive exploration of models followed by numerical solution, and approximate approaches based on discrete-event simulation and sampling have contrasting advantages, disadvantages and trade-offs. It is likely therefore that a successful analysis of large probabilistic system, such as a communication protocol, would make use of both types of technique.

6 Acknowledgements

The authors are supported by the French project RNTL “Averroes” (Marie Duflot and Claudine Picaronny), FORWARD and EPSRC projects GR/S11107 and GR/S46727 (Marta Kwiatkowska, Gethin Norman and David Parker) and

MIUR-FIRB Perf (Jeremy Sproston). The authors also gratefully acknowledge the contributions of Laurent Fribourg, Thomas Hérault, Richard Lassaigne, Frédéric Magniette and Stéphane Messika, and EPITA for the use of their computing facilities.

References

1. APMC web site. apmc.berbiqui.org.
2. PRISM web site. www.cs.bham.ac.uk/~dxp/prism.
3. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
5. C. Baier, F. Ciesinski, and M. Groesser. ProbMela and model checking Markov decision processes. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):22–27, 2005.
6. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. 15th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
7. M. Dufлот, L. Fribourg, T. Hérault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet, and C. Picaronny. Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. In *Proc. AVoCS'04*, volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 195–214. Elsevier Science, 2004.
8. M. Dufлот, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, 2004.
9. T. Gonsalves and F. Tobagi. On the performance effects of station location and access protocol parameters in Ethernet networks. *IEEE Transactions on Communications*, 36(4):441–449, 1988.
10. G. Guirado, T. Hérault, R. Lassaigne, and S. Peyronnet. Distribution, approximation and probabilistic model checking. In *Proc. 4th International Workshop on Parallel and Distributed Methods in Verification (PDMC'05)*, volume 135(2) of *Electronic Notes in Theoretical Computer Science*, pages 19–30. Elsevier, 2005.
11. H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
12. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
13. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proc. 5th Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 73–84, 2004.
14. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
15. B. Jeannot, P. D'Argenio, and K. Larsen. RAPTURE: A tool for verifying Markov decision processes. In *Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02)*, Technical Report FIMU-RS-2002-05, Faculty of Informatics, Masaryk University, pages 84–98, 2002.

16. J.-P. Katoen. A semi-Markov model of a home network access protocol,. In *Proc. MASCOTS'93*, pages 293–298, 1993.
17. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 2006. To appear.
18. M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.
19. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 286:101–150, 2002.
20. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Proc. PAPM/PROBMIV'02*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.
21. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.
22. M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. In *Proc. FORMATS/FTRTFT'04*, volume 3253 of *LNCS*, pages 293–308. Springer, 2004.
23. A. Roy and K. Gopinath. Improved probabilistic models for 802.11 protocol verification. In *Proc. CAV'05*, volume 3576 of *LNCS*, pages 239–252, 2005.
24. 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.
25. R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
26. J. Wang and S. Keshav. Efficient and accurate Ethernet simulation. In *Proc. 24th IEEE Conference on Local Computer Networks*, pages 182–191, 1999.
27. H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.
28. S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1:123–133, 1997.