Simulation Modelling Practice and Theory 77 (2017) 367–378

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Using PRISM model checker as a validation tool for an analytical model of IEEE 802.15.4 networks

Tatjana Kapus

University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška c. 46, SI-2000 Maribor, Slovenia

ARTICLE INFO

Article history:
Received 24 January 2017
Revised 23 July 2017
Accepted 1 August 2017
Available online 8 August 2017

Keywords:
Communications networks
Performance analysis
Analytical models
Validation
Formal specification
Probabilistic model checking

ABSTRACT

Commonly, simulation by using an existing network simulation tool or a simulator developed from scratch is employed for validation of analytical network performance models. An analytical model of star-shaped wireless sensor networks has been proposed in the literature in which, upon receiving a query from the coordinator, each sensor node sends one data frame to it by executing the IEEE 802.15.4 unslotted carrier-sense multiple access with collision avoidance algorithm. The model consists of expressions for calculation of the probability of successful receipt of the data at a certain time and the like. The authors of the model have written a special simulation program in order to validate the expressions. Our aim was to employ probabilistic model checker PRISM instead. PRISM only requires the user to formally specify the network as a kind of state machine and the queries about the probabilities sought in the form of logical formulas. It finds the probabilities automatically and can present them on graphs. We show how to specify the networks formally in such a way that all the expressions from the analytical model can be validated with PRISM. For those networks containing a few nodes, the validation can be carried out by normal model checking, which, in contrast to the simulation, always checks all the possible network behaviors, whereas statistical model checking can be used for the larger networks.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Performance models of networks and protocols are typically validated by using a well-known simulation framework, such as, for example, OPNET Modeler (now Riverbed Modeler), ns-2, or OMNeT++, or with a dedicated simulation tool developed from scratch. In [1], as well as in [2], an analytical model for performance analysis of the nonbeacon-enabled mode of the IEEE 802.15.4 Medium Access Control (MAC) protocol is proposed. A star-shaped wireless sensor network is considered which consists of a central node (a "sink") surrounded by sensor nodes. It is assumed that all the sensor nodes receive a query from the sink simultaneously and that each of them tries immediately to send one message to the sink by executing the version of the mentioned MAC protocol without acknowledgements. The model consists of mathematical expressions for calculation of the probability of successful receipt of the message at a certain time and similar events. An algorithm is provided for calculation with these expressions. In order to validate the expressions, the authors have written a dedicated

E-mail address: tatjana.kapus@um.si

http://dx.doi.org/10.1016/j.simpat.2017.08.002
1569-190X/© 2017 Elsevier B.V. All rights reserved.

simulation tool in C and compared the simulation results for several combinations of parameters with the analytical results calculated by using the algorithm.

Over the last decade, formal methods and tools which enable quantitative verification have been developed intensively [3]. The most important quantitative verification technique developed is probabilistic and real-time model checking [4,5]. Basically, model checking takes as input a state-machine model of a finite-state system, a specification of a property in the form of logical formula, and finds a definite answer as to whether the system has that property or not by automatically checking all its possible behaviors. In the past, model checking could only be used for the verification of logical correctness properties of systems, such as, for example, whether an event always happens after another one or whether a variable always has a particular value. With the invention of probabilistic model checking, it can also be used for verifying whether there is a certain probability of an event in the system, and, for example, to find out the probability of a certain event, the probability of an event at a given time, or the expected time until it happens. Having this capability, model checking has proved useful for the performance analysis of systems [6]. In the continuation, by formal verification we will mean model checking. Formal verification has several advantages over simulation. One of them is that all the possible runs of the system concerned are checked and the exact value of the probability or another quantity being sought is returned as a result [6]. Another advantage is that the property of interest can be checked automatically by expressing it by a logical formula and "pressing the button", whereas simulators typically require the user to carry out some calculations or observations on the outcomes of the simulation by herself/himself, or to make them a part of the simulation code in order to obtain the result [7,8]. Yet another advantage is that logical correctness properties can be stated with a logic and verified automatically before the verification of performance properties, whereas exhaustive verification of whether the model is built correctly is generally impossible by using the usual simulation frameworks.

Unfortunately, the probabilistic and real-time verification can usually only be performed for small systems because of the state-space explosion problem. It is for this reason that so-called statistical model checking is increasingly being investigated [9]. This technique is similar to simulation in that it executes the system model randomly up to a certain number of runs, but has an important advantage over simulation that properties of interest, including the correctness ones, can still be expressed by logical formulas, since it evaluates them on these runs. As, generally, not all the possible runs are examined, the answer obtained for a formula might be an approximation of the accurate result.

Probabilistic model checking has already been employed for the performance analyses of many kinds of networks and protocols (e.g., [10–12]), but to our knowledge, it has rarely been used for the validation of analytical models thereof. [13] reports that by using the probabilistic model checker PRISM [14], deficiencies have been found in an analytical web server performance model with proxy cache server. In [15], the results of using PRISM for probabilistic verification of a shuffling protocol for sharing data in a distributed network are compared to the results obtained with an analytical equational model.

PRISM is a powerful tool which supports formal modeling of networks with different kinds of state machines, enables the normal as well as statistical model checking, and can present quantitative results obtained by model checking on graphs [16]. In [17–19], it is employed for the probabilistic verification of a nonbeacon-enabled version of the IEEE 802.15.4 MAC protocol executed in similar kinds of networks as in [1], but without the aim of using the results to validate an analytical model of the protocol. The purpose of these works has been to devise accurate models of the standard protocol (see, e.g., [20]) and verify some performance properties. [17] and [18] present formal specifications of the networks with two sending nodes by using Markov Decision Processes (MDP) and Probabilistic Timed Automata (PTA) supported by PRISM, but do not provide accurate modeling of the clear-channel assessment period. Besides the improvement of the latter, [19] proposes different approaches to the formal specification of IEEE 802.15.4 star-shaped networks with an arbitrary number of sensor nodes by using PTAs. In [21], the effect of a non-standard clear-channel assessment period length in star-shaped 802.15.4 networks containing hidden nodes is analyzed with PRISM by using statistical model checking.

In contrast to [17–19,21], the aim of the research presented in this paper was to try to use PRISM for validation of an analytical model. In particular, the aim was to take the analytical model from [1] as given and to see whether it was possible to use the model checking with PRISM as an alternative to the validation of this model by classical simulation. As is often the case (cf., e.g., [22,23]), in [1] a MAC protocol slightly different from the standard one is considered, and different performance metrics are dealt with than in [17–19,21]. That is why we could not just have used the models and formulas from those papers. The main contribution of this paper are two kinds of network models for PRISM which represent exactly the networks considered in [1], allow us to specify all the performance metrics from the analytical model with probabilistic formulas in PRISM in a simple way, and enable efficient normal, i.e., exhaustive, and, respectively, statistical model checking of them. Note that there is no reason to doubt the claims of [1] about the validity of the analytical model based on the results of simulation. The aim of this paper is, therefore, not to carry out the validation of that model again, but to show how PRISM could be used for its validation advantageously.

In Section 2, we provide a description of the protocol from [1]. In Section 3, we briefly present the mathematical model of the protocol as well as the performance metrics from [1]. In Section 4, we indicate how the considered protocol differs from the standard one and explain both kinds of network models written in PRISM for the former. In Section 5, it is shown how to specify each performance metric in order to be able to use the model checking for it. In Section 6, we present some results of the normal as well as statistical model checking for these metrics. Section 7 contains a discussion and concludes the paper.

2. MAC protocol

In [1], a nonbeacon-enabled star-shaped wireless personal area network is assumed consisting of a coordinator and N sensor nodes. The analytical model covers the behavior of the nodes after they receive a query from the coordinator. At that moment, "time 0", each node starts to execute the following *unslotted Carrier Sense Multiple Access with Collision Avoidance* (CSMA-CA) algorithm in order to send one data frame directly to the coordinator.

The algorithm is based on units of time called *backoff periods*, their duration denoted d_b. d_b is assumed to be equal to 20 symbol times as in [20]. The standard symbol rate 62.5 ksymbol/s is assumed, which gives the symbol time equal to 16 µs [2]. Each node maintains variables NB and BE. Initially, NB is set to 0, and BE is set to the value of BE_{min}, which is equal to 3. The node backs off (i.e., waits) for a random number of units, selected uniformly from the range $[0, 2^{BE} - 1]$, and subsequently performs *channel sensing*, called *Clear Channel Assessment* (CCA) in [20], for one unit of time. If the channel is assessed to be clear, i.e., if no node is sending data during this period, the algorithm is terminated successfully, and the node begins to transmit the data frame. If, however, the channel is assessed to be busy, the node increases NB by 1. It also increases BE by 1 if the latter is smaller than BE_{max}, which is equal to 5. If the increased NB is greater than the value of NB_{max}, which is equal to 4, the algorithm terminates with a *channel access failure* declaration. If, however, NB has not yet surpassed the maximal allowed number of channel access attempts, the node returns to the step in which the random backoff takes place and repeats the whole procedure by using the new values of BE and NB.

It is assumed that all the nodes send data frames of equal length or duration, the latter being equal to an integral number of units. Furthermore, it is assumed that all the nodes can "hear" each other and that there is no propagation delay and bit corruption. If two or more nodes perform sensing simultaneously and the channel is clear, all of them transmit a data frame. Consequently, the frames collide and are received unsuccessfully by the sink. This is the only possible way for the data frames to collide as well as to be received unsuccessfully.

3. Analytical model

In this subsection, we present the analytical model from [1] to the extent needed to understand the rest of this paper. As in the MAC protocol considered in [1], the backoff delay, the duration of sensing, as well as the duration of transmission are expressed in units of time with duration d_b, the resolution time of the model is set equal to d_b, such a unit is called a slot, and the current time is expressed as the current slot number. The jth slot, for $j = 0, 1, 2, \ldots,$ means the time from $j \cdot d_b$ to $(j+1) \cdot d_b$. Furthermore, the duration of a data frame transmission is denoted by D, meaning the duration $D \cdot d_b$. With the symbol rate of 62.5 ksymbol/s, two symbols together are one octet long [20]. As the standard prescribes the data frame length from 15 to up to 133 octets, D is taken to be between 1 and 13 (cf. [2]).

t_{max} denotes the last slot in which a transmission can start. It is equal to $\sum_{k=0}^{NB_{max}} W_k = 120$. Here, for $k = 0, 1, 2, 3, 4$, $W_k = 2^{BE}$ for the value of BE at NB equal to k. Notice that BE is equal to 3 when $NB = 0$, to 4 when $NB = 1$, and to 5 otherwise (please see Section 2). A node will start to transmit at the beginning of slot t_{max} in the worst case, i.e., if it finds the channel to be busy for NB equal to 0, 1, 2 and 3 and free for NB equal to 4, and if for $NB = k$, $k = 0, 1, 2, 3, 4$, it backs off for the greatest possible number of slots, $W_k - 1$, before the channel sensing. The last slot at the end of which a node can finish the transmission of the data frame is, therefore, $t_{max} + D - 1$. A node can be in the sensing state at the latest in slot $t_{max} - 1$.

Analytical expressions for the following performance metrics are derived in [1]:

- $P\{T^j\}$ with $j \in [0, t_{\max} + D - 1]$—the probability that a node ends the transmission of its data frame in slot j (i.e., at the end of slot j), either with or without a collision,
- $P\{Z^j\}$ with $j \in [0, t_{\max} + D - 1]$—the probability that a node ends the transmission of its data frame in slot j successfully, i.e., without a collision,
- p_s—the probability that a node ends the transmission of its data frame successfully, whatever the slot,
- $P\{R^j\}$ with $j \in [0, t_{\max} + D - 1]$—the probability that in slot j (i.e., at the end of slot j) the sink receives the end of a data frame, coming from whatever node and not collided.

4. Network specification in PRISM

In PRISM, a system specification consists of modules, which represent concurrent processes [16]. The syntax and semantics of the modules depend on the model type chosen. A difference between the MAC protocol considered in [1] and the standard one considered in [17–19,21] is in that the clear channel assessment in the latter does not last a whole backoff period, i.e., 20 symbol times, but only 8 symbol times. If the node assesses the channel to be clear for the whole CCA period, it spends the remaining 12 symbol times turning the radio from the receiving state to the transmitting one. Otherwise, generally, it starts to repeat the backoff procedure immediately after the CCA period. A consequence of the radio switching time is that, in contrast to the protocol considered in this paper, a node can start to send a data frame although another node is already sending one. Another difference is that, in the standard protocol, the length of the data frames does not need to be a multiple of backoff periods. Therefore, time units smaller than the backoff period are modeled in [17–19,21]. Besides, the PTA and MDP types of probabilistic models are used in which events from different nodes which would occur simultaneously in reality are executed one after another, the ordering being chosen nondeterministically. For this reason, the probability of certain events, such as successful receipt of a message, depends on the ordering chosen.

For the protocol assumed in this paper, there is no particular reason to model the time units smaller than the slot, representing one backoff period, and this is also not desirable. It should be noticed that all the nodes in the protocol progress fully synchronously slot by slot and that there is no nondeterminism. Our aim, of course, was to model exactly this protocol, and we wanted to model it as directly as possible for the following reasons: to obtain as small a global state space as possible during the model checking, to be able to express the performance metrics that are functions of the slot number in the analytical model also as logical formulas in PRISM referring directly to the slot number, and to obtain exactly one outcome for each performance metric as in the analytical model.

We, therefore, chose the DTMC (Discrete-Time Markov Chain) model type, which does not support nondeterminism (for definitions of DTMC, MDP, and PTA model types supported in PRISM see, e.g., [18]). In that case, a module definition consists of variable declarations (with the initial values defined optionally with the keyword `init`) and commands, which define the values of the variables in the next state based on the values of these and the variables of the other modules in the current state. The value of a variable v in the next state is denoted v'. The form of a command is [*action*] *guard* `->` p_1 : *update*$_1$ + ... + p_n : *update*$_n$, where p_i, $i = 1, \ldots, n$, are probabilities, the sum of which has to be 1. If the guard is true in the current state, *update*$_i$ of the module's variables can be executed with probability p_i, for $i = 1, \ldots, n$. The label *action* inside the brackets is optional. The commands from different modules labeled with the same action can only be executed simultaneously. Unlabeled commands execute one at a time.

Fig. 1 shows a specification for the network consisting of three sensor nodes. The specification starts with definitions of the constants of the protocol. If the value of a constant in PRISM is not defined in the specification, it can be set outside it before the verification. Each of the modules `Node1`, `Node2`, and `Node3` represents one sensor node. As in [17–19,21], there is no need for a module representing the central node. It suffices to specify precisely only one sensor node (see module `Node1`) as all the others (`Node2` and `Node3` in Fig. 1) can be obtained by renaming its variables.

In order to achieve synchronous execution of all the modules representing the nodes and to be able to verify properties related to slot numbers, we introduced module `timer` (cf. [18]). It contains the integer variable `t`, which represents global time, measured in slots. Its initial value is 0. It is incremented by the action named `time` until it reaches the constant value `Tmax`, equal to 120 (i.e., t_{max} from the analytical model) plus `D`, where `D` represents the length of the data frames as in [1]. The end of the jth slot, for $j = 0, \ldots, t_{\max} + D - 1$, from the analytical model is represented in PRISM by variable `t` equal to $j + 1$. Fig. 2 illustrates this relationship for the case $D = 1$.

All the commands of the modules representing the nodes are also labeled with action `time`. In this way, it is assured that each of them executes one command whenever `t` is incremented by module `timer`. Please note that the checking as to whether the channel is busy when `s1=2` and `x1=D` is not part of the protocol. This and the last two commands of the module help in property specification. Except for these, every command of `Node1` in Fig. 1 and, analogously, of the other nodes, exactly corresponds to the execution of a slot in accordance with the protocol. Consequently, the specified network behavior exactly corresponds to the behavior assumed by the analytical model.

Variable `s1` of module `Node1` (and analogous variables for the node modules with larger "indices") denotes the "state" of the module. When `s1=0`, one of the first three commands is enabled, depending on the value of variable `be1`. These commands represent the probabilistic choice of the number of slots the node will back off before sensing the channel, and something else. Notice that according to Section 2, the choice itself does not consume any time. As in our model, each command represents the execution in one slot, the commands must specify what else happens in the slot besides the choice. Two different cases have to be distinguished: the one in which the node chooses the number of backoff periods to be equal to 0 and the one in which it chooses the number to be greater than 0. In the second case, the slot in which this command is executed already has to be counted as one period of backing off. This case is represented in the first three commands by all the updates except the first one. One can see that they all set `s1` to 1 and variable `backoff1` to a value between 0 and $2^{\mathtt{be1}} - 2$, for example, to 6 if `be1=3`. As can be seen from the fifth, sixth, and seventh commands, the update to (`s1=1`, `backoff1=0`) means that the backoff has finished (its duration was one slot), whereas the updates of `backoff1` to a value greater than 0 mean that one backoff period has been executed and that the node still has to back off for the number of slots set in `backoff1`. For example, if `backoff1` is set to 6, the node still has to backoff for six slots before sensing.

The case where the node chooses the number of backoff periods to be 0 is modeled by the first probabilistic choice in the first three commands in module `Node1`. The reader can see that `backoff1` is set to 0, whereas variables `s1`, `x1`, `nb1`, and `be1` are set with the help of formulas defined before the module. This is because there must be no backoff and the slot in which the command is executed must already be the sensing slot. The reader can check that formulas `backoff0s` etc. define the updates of the variables in accordance with the protocol description in Section 2. Conditional assignment is used. These formulas use formula `busy`, which is true if at least one of the other nodes is in the state of sending (denoted by value 2 of the "state variables"). CCA is modeled by checking this formula. Please note that the incrementation of `nb1` is modeled in a slightly different way than described in [1] and Section 2 but the effect is the same. If the channel is detected to be free in the sensing state, variable `s1` is set to 2 and variable `x1` to 1. Variable `x1` is used in the guards of the commands representing the sending. Its value in the guard means how many units out of all the `D` units of the data frame will have been sent by the end of the slot in which the command is executed.

```
dtmc

const int BE_MIN = 3;
const int BE_MAX = 5;
const int NB_MAX = 4;
const int D;

const int Tmax = 120 + D;

formula busy = s2=2 | s3=2;

formula backoff0s = (busy) ? ((nb1=NB_MAX) ? 3 : 0) : 2;
formula backoff0x = (!busy) ? 1 : x1;
formula backoff0nb = (busy) ? ((nb1=NB_MAX) ? nb1 : nb1+1) : nb1;
formula backoff0be = (busy) ? ((nb1=NB_MAX) ? be1 : min(be1+1,BE_MAX)) : be1;

module Node1

  nb1 : [0..NB_MAX] init 0;
  be1 : [BE_MIN..BE_MAX] init BE_MIN;
  backoff1 : [0..pow(2,BE_MAX)-1] init 0;
  s1 : [0..4] init 0;
  x1 : [0..D] init 0;

  [time] s1=0 & be1=3 -> 1/8 : (backoff1'=0) & (s1' = backoff0s) & (x1' = backoff0x)
                             & (nb1' = backoff0nb) & (be1' = backoff0be)
                       + 1/8 : (s1'=1) & (backoff1'=0) + ... + 1/8 : (s1'=1) & (backoff1'=6);
  [time] s1=0 & be1=4 -> 1/16 : (backoff1'=0) & (s1' = backoff0s) & (x1' = backoff0x)
                              & (nb1' = backoff0nb) & (be1' = backoff0be)
                       + 1/16 : (s1'=1) & (backoff1'=0) + ... + 1/16 : (s1'=1) & (backoff1'=14);
  [time] s1=0 & be1=5 -> 1/32 : (backoff1'=0) & (s1' = backoff0s) & (x1' = backoff0x)
                              & (nb1' = backoff0nb) & (be1' = backoff0be)
                       + 1/32 : (s1'=1) & (backoff1'=0) + ... + 1/32 : (s1'=1) & (backoff1'=30);
  [time] s1=1 & backoff1>0 -> 1 : (backoff1'=backoff1-1);
  [time] s1=1 & backoff1=0 & !busy -> 1 : (s1'=2) & (x1'=1);
  [time] s1=1 & backoff1=0 & busy & nb1<NB_MAX -> 1 : (s1'=0) & (nb1'=nb1+1)
                                                   & (be1'=min(be1+1,BE_MAX));
  [time] s1=1 & backoff1=0 & busy & nb1=NB_MAX -> 1 : (s1'=3); // channel access failure
  [time] s1=2 & x1<D -> 1 : (x1'=x1+1);
  [time] s1=2 & x1=D & !busy -> 1 : (s1'=4); // successful end of transmission
  [time] s1=2 & x1=D & busy -> 1 : (s1'=3); // collision failure
  [time] s1=4 -> 1 : (s1'=3);
  [time] s1=3 -> 1 : true;

endmodule

module Node2=Node1[x1=x2,s1=s2,s2=s1,be1=be2,nb1=nb2,backoff1=backoff2] endmodule

module Node3=Node1[x1=x3,s1=s3,s3=s1,be1=be3,nb1=nb3,backoff1=backoff3] endmodule

module timer
  // global time
  t : [0..Tmax] init 0;
  [time] (t<Tmax) -> (t'=min(t+1,Tmax));
  [] (t>=Tmax) -> (t'=t);
endmodule
```

Fig. 1. DTMC model in PRISM for network with three nodes.

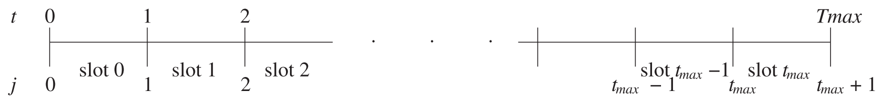

Fig. 2. Relationship between j and t shown with timeline for case $D = 1$.

The fourth command of module Node1 represents a backoff for one slot which is not the first backoff period. The next three commands represent the execution of sensing in one slot. They affect variables s1, x1, nb1, and be1 in the same way as the update for the case of no backoff in the first three commands.

The two commands with expression x1=D in the guards represent the sending of the last unit of data out of D units. That is why in these commands it is checked whether any other node is sending data. If it is, s1 is set to 4 at the end of this slot, which means successful end of sending. Otherwise, it is set to 3, which denotes a collision. (Please note that 3 is already used to denote channel access failure. We will actually set the value of s1 when x1=D depending on the metric to be validated.) It suffices to check for the collision only in the last slot of the data sending because in the simplified protocol, it is not possible for a node to start to send the data frame when another node is already sending one. If two or more frames collide, they collide in all the D slots which they occupy and thus also in the last one.

Module Node2 is obtained by renaming all the variables of Node1 to the variables with “index” 2 and by renaming variable s2 to s1, and analogously for module Node3. Please note that PRISM also renames the variables in the formulas used in module Node1.

In order to obtain the specification of the network with two sensor nodes, the definition of Node3 should, of course, be removed and condition s3=2 deleted from formula busy. In order to obtain the specification of the networks with N nodes for $N > 3$, disjunct si=2 for each i, $3 < i \leq N$, should be added in the definition of formula busy, and the module for each additional node defined with the help of Node1 by renamings analogous to those in the definition of Node3.

The direct modeling of the protocol by using the DTMC model type generally assures the minimal size of the global state space of the model (the size can vary depending on to how many different values the state variables si are set for the validation of different metrics) because the concurrent components execute in lockstep. We, however, found by experiments that statistical model checking with it takes an unexceptionally long time even for networks with less than ten nodes. Experiments confirmed that the solution is to use a network specification without the conditional assignment statements. We prepared the new specification from the one in Fig. 1 by changing the model type from DTMC to MDP, by removing the definitions of formulas that contain the conditional assignments, and by changing the first three commands of the node module in the following way. They are not labeled with action time anymore. The only other change is that in each of them, every update only sets variable s1 to 1 and variable backoff1 in turn to a value between including 0 and $2^{\texttt{be1}} - 1$, thereby achieving that these commands represent only the selection of the number of backoff periods. Because they are not labeled, the time in the model is not advanced, which is in accordance with the assumption of the protocol that the selection itself does not consume time. The absence of labels also means that they are executed one at a time. If such commands are enabled in one or more modules and some of the labeled commands in others, one of the former will be executed because the labeled ones can only be executed if they are enabled in all the modules. The order of the execution of the enabled backoff selection commands is chosen nondeterministically, which is the usual way of modeling concurrent execution. The nondeterministic choice is ensured by the MDP model type [18]. Note, however, that the different execution orders do not affect the enabling of the other commands in different ways. It follows that for any order chosen at such points, the probabilities of data sending at certain times remain the same as in the DTMC model.

5. Property specification in PRISM

In this section, we show that all the performance metrics listed in Section 3 can be expressed in PRISM’s property specification language. The latter subsumes several probabilistic temporal logics, including PCTL (Probabilistic Computation Tree Logic) [16]. For DTMC models, the queries about probabilities can be specified with formulas of the form P=? [*pathprop*], where *pathprop* is an LTL-style (LTL is short for Linear-time Temporal Logic) formula expressing a property of a path in the model. In this paper, we will use only *pathprops* containing operators F (which means “eventually” or “future”) and X (“next”). The result of model checking of the query is the probability that *pathprop* is satisfied by the paths from the initial state of the model. For normal model checking in models which include nondeterministic choice, such as MDP models, formulas of the form Pmin=? [*pathprop*] or Pmax=? [*pathprop*] have to be used. They give the minimal or, respectively, maximal probability, over all possible resolutions of nondeterminism, that *pathprop* is satisfied by the paths from the initial state of the model [16,18]. As in our MDP models the minimal and maximal probabilities are the same, we will write P=? in all the presented formulas for simplicity. Unless stated otherwise, they can be used for both kinds of models.

All the metrics except $P\{R^j\}$ in Section 3 refer to a single but arbitrary node. As the protocol and the models are symmetric, the numerical results for these performance metrics are clearly the same for all the nodes. We shall, therefore, give the formulas in PRISM’s language for all these metrics only for Node1. Besides, although we shall explicitly refer to a model of the network with three nodes, such as the one in Fig. 1, by this we will also mean the analogous models for the networks with two or more than three nodes, unless stated otherwise.

First, we verified some logical correctness properties. One of the properties that does not hold in the networks with the standard protocol, but should be true in the networks with the protocol considered, is the following. It should not be possible that a node starts to send a data frame when another one has already sent a part of a data frame. We verified it with the help of formula P=? [F (s1=2&s2=2&x1!=x2)]. The result of model checking was 0.

As to the performance metrics, please note that in the model in Fig. 1, s1 is set to 4 and in the slot following this to 3 in order for 4 to signal the slot in which the sending is ended successfully. There are different possibilities of specifying the performance metrics. One is, for example, to indicate all the important events, such as the successful end of sending, with different values of `s1`, and use the same model for expressing and verifying all the metrics. Another one is to indicate no important events. At least in the latter case, nested operator `X` would be needed in the formulas. In the former case, the global state space might be the largest, and in the latter the smallest. Yet another possibility is to use different models with only as much "signaling" as needed for each metric to be expressed without operator `X`. The advantage of this is that the state space is possibly smaller, that the normal model checking takes less time because of simpler formulas and, last but not least, that the statistical model checking can be carried out. Notice that the statistical model checker does not currently accept probabilistic logic formulas containing complex LTL-style path properties. For example, it does not allow nesting of temporal operators [16]. We shall, therefore, show only one specification with operator `X`, whereas our goal was to specify all the properties with only as much signaling in the models as necessary and with simple PCTL formulas.

First, suppose that in Fig. 1 in `Node1` there is only one command for checking the end of sending, that it does not test the value of formula `busy`, and that it sets `s1` to 3. In that case, $P\{Z^j\}$ could be obtained with the following formula: `P=? [ F (t=j&s1=2&s2!=2&s3!=2& (X (s1=3))) ]`. This formula returns $P\{Z^j\}$ both in the DTMC and MDP models. Note, however, that generally, different formulas containing `X` might be necessary to query about the same performance metric in the two models because `X` in the MDP model does not necessarily mean the next slot.

If the last four commands are as in Fig. 1, then $P\{Z^j\}$ can be expressed with formula `P=? [ F (t=j+1&s1=4) ]` and p_s with formula `P=? [ F (s1=4) ]`. $P\{T^j\}$ can be expressed with the same formula if `s1` is also set to 4 in the case of collision failure. (In that case, of course, one command, without testing `busy`, would suffice to end the sending.) All these formulas are the same for any number of sensor nodes, whereas $P\{R^j\}$ has to be expressed with different formulas for different numbers of the nodes. For the model with three nodes in Fig. 1, it can be expressed with formula `P=? [ F (t=j+1&(s1=4|s2=4|s3=4)) ]` and, generally, with as many disjuncts of the form `s`*i*`=4` as there are nodes.

In [1], two additional probabilities are defined and used for validation of the analytical model:

- $F_Z(j)$ with $j \in [0, t_{\max} + D - 1]$—the probability that a node ends the transmission of its data frame successfully at the latest in slot j,
- $F_T(j)$ with $j \in [0, t_{\max} + D - 1]$—the probability that a node ends the transmission of its data frame at the latest in slot j, either with or without a collision.

Probability $F_Z(j)$ can be expressed with formula `P=? [ F (t=j+1&s1=4) ]` if the value of `s1` is left equal to 4 forever once it obtains this value in Fig. 1, i.e., if the update in the command before the last one is identical to `true` or sets `s1` to 4. Probability $F_T(j)$ can be expressed with the same formula if `s1` is also set to 4 in the case of collision failure and left equal to 4 forever by the command before the last one.

It should be noticed that in all the proposed formulas in the PRISM property specification language that contain j, the latter is a parameter which directly correponds to j in the analytical model. In PRISM, j must be defined as a constant. This allows one to obtain the results of model checking for the formulas in a similar way to calculating the results by using the formulas of the analytical model. For example, in the graphical user interface, one only has to set the value of D and the range $[0, t_{\max} + D - 1]$ for j, and invoke model checking. PRISM then executes a so-called experiment: it executes model checking for all the values of j and can represent the results on a graph. In general, several values of different constants can be set in advance and the model checking engine invoked once to execute experiments for selected combinations of the constant values [16]. The experiments are supported both for the normal and statistical model checking.

6. Some results of model checking

In this section, we present some results of model-checking the models with signaling with PRISM. In contrast to the simulation, it is possible to obtain exact numerical values for the performance metrics by using the normal model checking in PRISM to the extent allowed by the available computer. We, therefore, first provide some graphs showing the exact results. For comparison, we provide the results obtained with the analytical model on the same graphs.

We implemented the algorithm for the calculation of the performance metrics that is given in [1]. We added a code for writing the results to a selected file in the XML-based format used to plot graphical representation of numerical results in PRISM. In this way, we could combine the analytical results and the model-checking ones in one file and plot the graph for them by importing the file to PRISM. We used PRISM 4.3.beta for all experiments.

We calculated success probability p_s by running the algorithm as well as the normal model checking for the networks with the number of nodes $N = 2$ and $N = 3$ for $D \in [1, 13]$. The results are shown in Figs. 3 and 4. As expected, the values calculated with the expression derived in [1] are slightly greater than the exact ones.

In Figs. 5–9, we give graphs that show probabilities $P\{Z^j\}$, $P\{T^j\}$, and $P\{R^j\}$, as well as cumulative probabilities $F_Z(j)$ and $F_T(j)$, respectively, for $j \in [0, t_{\max} + D - 1]$ with D equal to 1, 2, and respectively 13, obtained with the normal model checking and calculated by using the analytical model for $N = 3$. (Let us remark that the analytical results for $D = 1$ in all the graphs shown in this paper were obtained for the number of competing nodes at a particular j, denoted N_c^j in [1], equal to N.) Although in Figs. 5–7, the complete intervals for j for different values of D are indicated in the legends, the probabilities for the higher values of j are not shown because their values are equal to 0 or close to 0. There is a good match between the exact values and those obtained through the analytical model for D equal to 1 and 2. For the maximal frame length, there are mismatches at j between about 15 and 55. Consequently, there are also mismatches in the graphs of the cumulative probabilities. Similarly shaped graphs were obtained for these metrics for $N = 2$ but, clearly, with greater numerical values

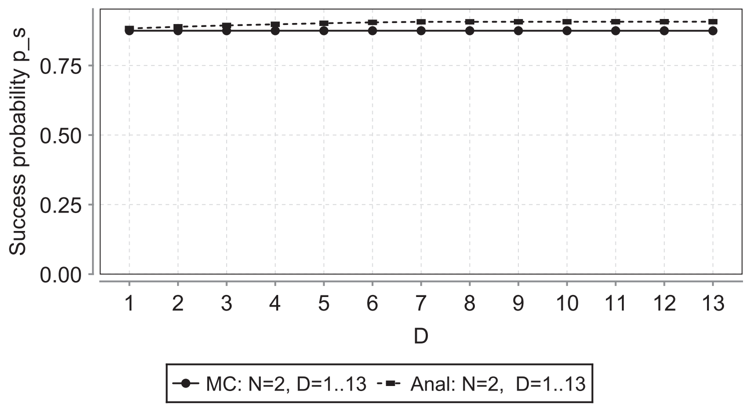

Fig. 3. Success probability p_s obtained with normal model checking and analytical model for networks with two nodes and different values of D.

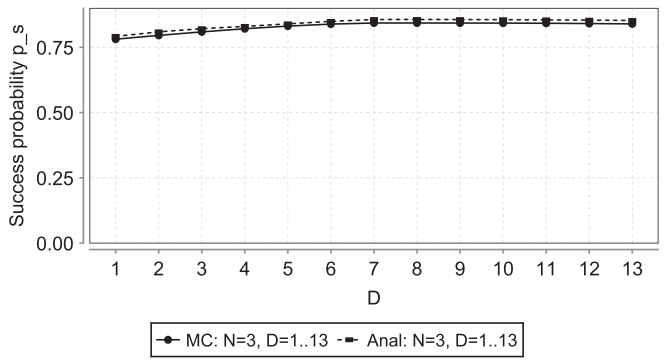

Fig. 4. Success probability p_s obtained with normal model checking and analytical model for networks with three nodes and different values of D.

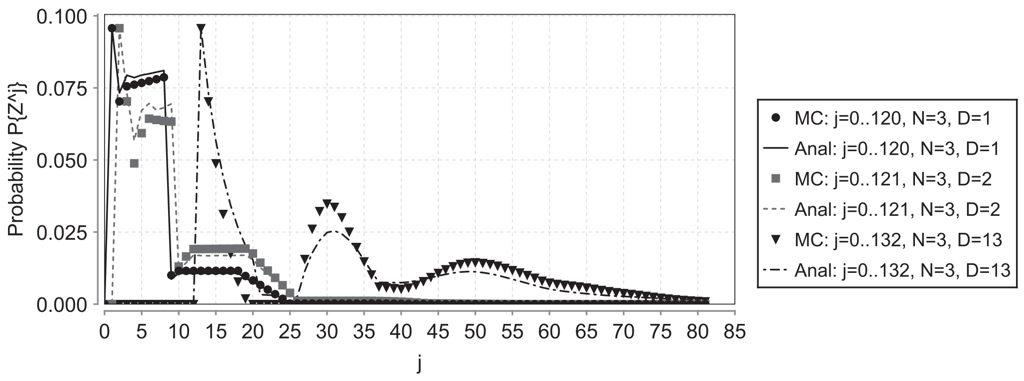

Fig. 5. Probabilities $P\{Z^j\}$ obtained with normal model checking and analytical model for networks with three nodes and different values of D.

than in the network with three nodes for the same j. According to [1] and confirmed by our experiments using statistical model checking, the mismatches tend to be smaller for larger networks.

As to the numerical results, it does not matter whether the DTMC or MDP model is used, but there is some difference in the size of the state space and model-checking times. For example, we observed that, once the global state space was generated by the model checker, the computation of $P\{Z^{50}\}$ for $N = 3$ and $D = 13$ took around 38 seconds for the DTMC model and around 48 seconds for the MDP model. The size of the global state space of the former was 3,832,426 states and of the latter 4,296,126 states.

By using a personal computer with an Intel®Core$^{\text{TM}}$ i5 760 2.80 GHz processor and 3.22 GB of usable RAM, we performed experiments with the normal model checking by using the MTBDD engine for networks with up to $N = 5$. For the DTMC model with $N = 5$, which gave 97,517,142 states, it already took, for example, around 38 seconds to compute $P\{Z^j\}$

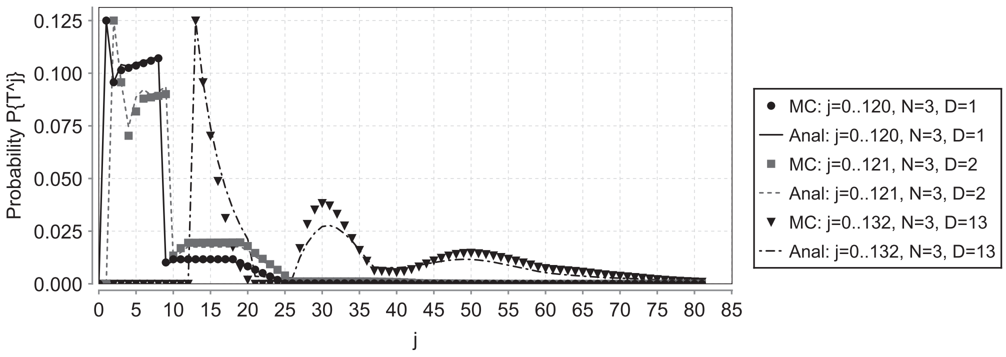

Fig. 6. Probabilities $P\{T^j\}$ obtained with normal model checking and analytical model for networks with three nodes and different values of D.

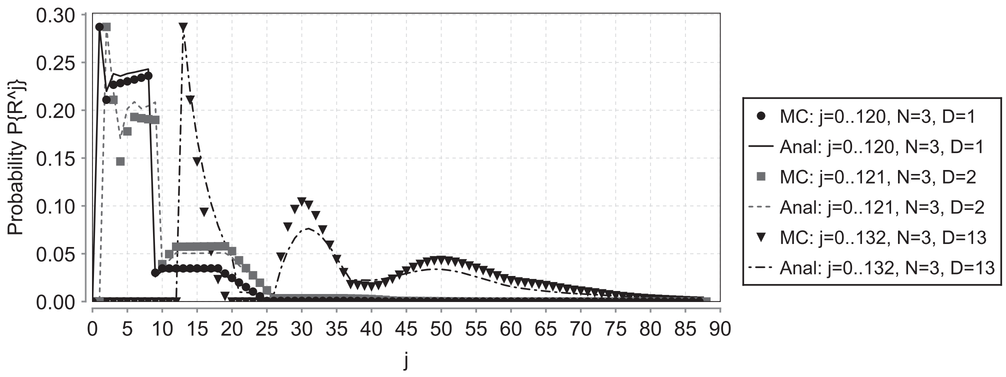

Fig. 7. Probabilities $P\{R^j\}$ obtained with normal model checking and analytical model for networks with three nodes and different values of D.

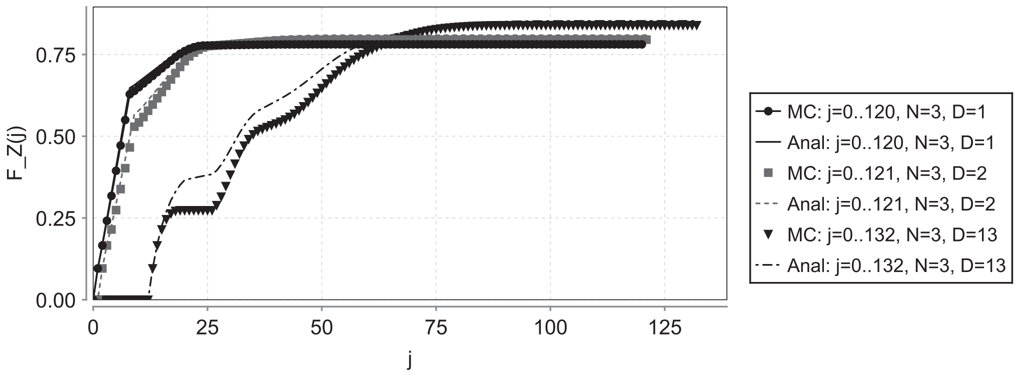

Fig. 8. Values of function $F_Z(j)$ obtained with normal model checking and analytical model for networks with three nodes and different values of D.

for $j = 14$ and $D = 1$, whereas for $j = 50$ it took around 175 seconds. The MDP model with the same number of nodes had 177,334,240 states and the model checking for $j = 50$ took around 660 seconds. For larger networks, we applied statistical model checking in order to avoid running out of memory or waiting long for the result. Whereas the normal model checking needed less time with the DTMC models, the contrary was true for the statistical model checking. While the latter did not return the results for the DTMC model of the network with seven nodes for $P\{R^j\}$ for all the possible values of j and $D = 13$ within 15 minutes, it took less than a minute with the MDP model.

It should be noticed that, although the statistical model checker in PRISM accepts formulas which ask about the minimal or maximal probabilities of path properties for MDP models, it does not differentiate between them because it is, in fact, not meant for probabilistic models which contain nondeterminism [16]. It interprets any nondeterministic choice as a probabilistic one with uniform distribution over all the alternatives and, therefore, returns the average probability [16,24].

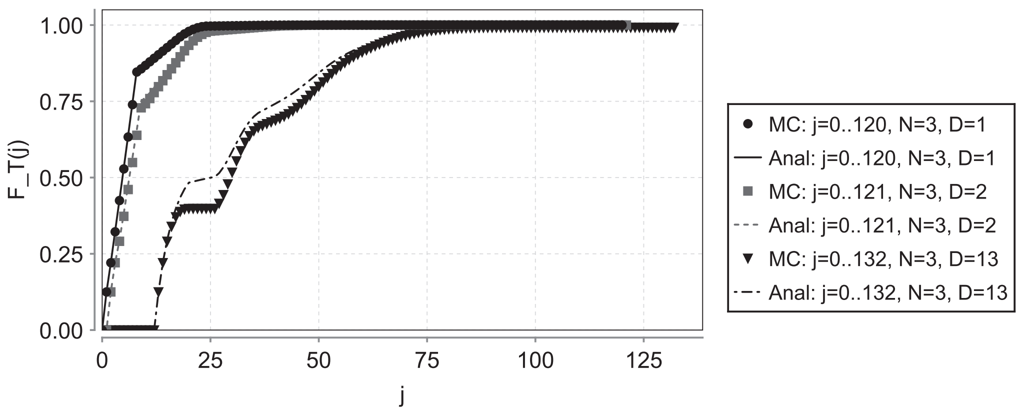

Fig. 9. Values of function $F_T(j)$ obtained with normal model checking and analytical model for networks with three nodes and different values of D.

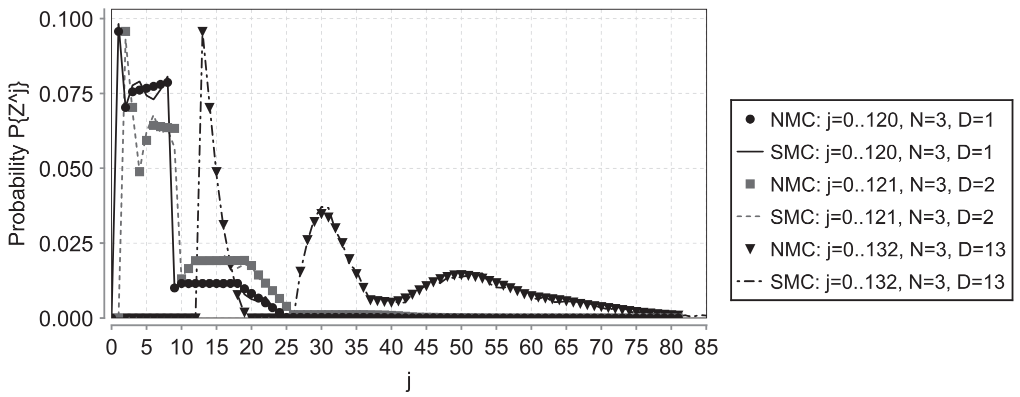

Fig. 10. Probabilities $P\{Z^j\}$ obtained with normal and statistical model checking for networks with three nodes and different values of D.

As the minimal and maximal probabilities are the same in our models, it returns exactly the results that correspond to the performance metrics from [1].

For all experiments using statistical model checking, we chose to generate 10,000 runs (called samples in PRISM), which is equal to the number of runs in [1], with the maximal generated run length equal to 1000 steps. The latter was sufficient for the model checker to be able to find a definite answer in all the executed experiments. The statistical model checker writes the maximal length of the generated runs during an experiment in a log. The length was not greater than 330 steps even for 40 nodes and long frames.

With 10,000 samples, the values obtained with the statistical model checking were very close to the exact ones. See, for example, Fig. 10, in which we give probabilities $P\{Z^j\}$ obtained by using the normal model checking as well as the statistical one. Let us mention that, given the number of samples, PRISM is capable of estimating the reliability of the results, and vice versa [16]. By using its Confidence Interval (CI) method for statistical model checking and setting the confidence level to 99 % (i.e., by setting its parameter α to 0.1), we obtained for our experiments that 99 % of the time, the exact values should, typically, be at most a few percentages smaller or greater than the obtained results.

In order to be sure about the correctness of our models and formulas for PRISM, as well as our implementation of the algorithm from [1], we performed model checking and the calculation with the analytical model for almost all the cases for which the graphs are shown in [1]. We obtained very similar results. As an evidence, we provide two of our graphs: the one for $P\{R^j\}$ in Fig. 11, which should be compared with the graph in Fig. 12 in [1], and the one for $F_Z(j)$ in Fig. 12, which the reader is invited to compare with the graph in Fig. 15 in the same article. The results of statistical model checking in Fig. 12 were obtained in around one or two minutes for $N = 10$ and, respectively, $N = 20$, and in less than 10 minutes for $N = 40$ and $D = 10$.

Finally, since statistical model checking is similar to simulation, we made the following experiment with OMNeT++ [7] for a comparison of “simulation” execution times. We mapped the DTMC model shown in Fig. 1 to a model in OMNeT++ consisting of a single simple module and having the message length D and the number of nodes N as parameters. In the C++ class defining the behavior of the simple module, we represented the variables of each node from the model in PRISM as

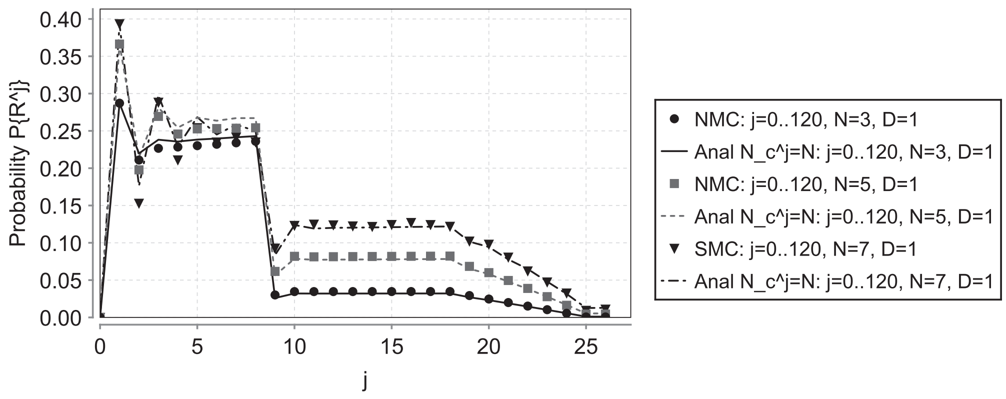

Fig. 11. Probabilities $P\{R^j\}$ obtained with normal or statistical model checking and analytical model for N equal to 3, 5 and 7, and $D = 1$.

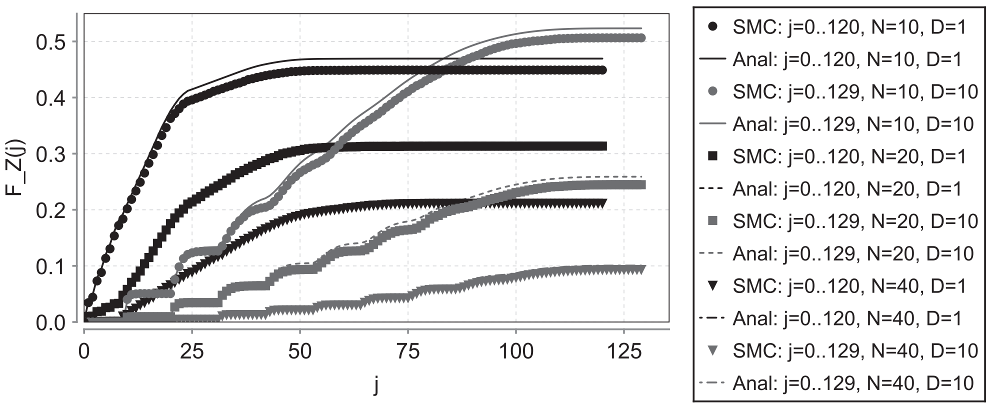

Fig. 12. Values of function $F_Z(j)$ obtained with statistical model checking and analytical model for different values of N and D.

one component of an array of length N and "translated" the definitions of constants and formulas as well as the commands of the node given in Fig. 1 into C++. The simulation was driven by the simple module sending a timeout message to itself initially and after the execution of one command for each node upon the receipt of the previous message, i.e., after the execution of each slot. We aimed at obtaining graphs of $F_Z(j)$ for all the values of j similar to those obtained with PRISM solely with the use of OMNeT++, but this was not possible. As OMNeT++ is not able to make a calculation over the results of several simulation runs, we made the module in OMNeT++ execute the MAC protocol 10,000 times. By using the signal-based statistics recording mechanism of OMNeT++ [7], we succeeded to make the simulator generate for a selected network node a histogram showing for each slot the number of MAC protocol runs in which it has successfully transmitted a message, i.e., a graph analogous to the graph of $P\{Z^j\}$ for all the possible values of j, but not yet showing the probabilities. For $N = 40$ and $D = 10$, the simulation took around 5 seconds, but note that in order to obtain a graph of $P\{Z^j\}$ or $F_Z(j)$, the obtained histogram should still have been exported to and processed with an external tool after the simulation. The short simulation time itself might be a consequence of the implementation in OMNeT++ consisting of one module only, which was possible because of the synchronous execution of the nodes and relatively easy to prepare thanks to the formal model previously elaborated with PRISM. Generally, a model in OMNeT++ would consist of several modules executing asynchronously and communicating with message passing, but it can nevertheless be expected that the statistical model checker would need more time than simulators for the same network, since it has to generate in every reached global state possible next global transitions based on the commands of the component modules and evaluate the formula being checked [25].

7. Conclusions

We showed how PRISM could be applied successfully for the validation of the analytical performance model derived in [1] instead of writing a special simulation program for that purpose. We represented the protocol from [1] in a way that allowed us to query about all the probabilities by referring directly to the slot number, and to obtain the results quickly with normal model checking for small networks and with statistical model checking for larger ones. Instead of adapting an existing PTA or MDP model of the standard protocol, we prepared two new models by using the DTMC and, respectively, MDP model type and a timer module to achieve synchronous progress of all the nodes in every slot.

The presented case study suggests that PRISM might be worth consideration as another candidate besides the popular network simulation tools and self-written ones for performing validations of analytical performance models of protocols. In the future, we intend to study the possibilities of using probabilistic model checking in combination with classical simulation tools for validation of complex analytical models, as well as the possibilities of compositional probabilistic model checking.

Acknowledgements

This work was partially funded by the Slovenian Research Agency (ARRS) [contract number P2-0069].

References

[1] C. Buratti, R. Verdone, Performance analysis of IEEE 802.15.4 non beacon-enabled mode, IEEE Trans. Veh. Technol. 58 (2009) 3480–3493.
[2] C. Buratti, M. Martalò, R. Verdone, G. Ferrari, Performance analysis of the IEEE 802.15.4 MAC protocol, in: Sensor Networks with IEEE 802.15.4 Systems, Springer-Verlag, Berlin, 2011, pp. 161–211.
[3] G. Norman, D. Parker, Quantitative verification: Formal guarantees for timeliness, reliability and performance, a Knowledge Transfer Report from the London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering, 2014.
[4] J.-P. Katoen, Perspectives in probabilistic verification, in: Proc. 2nd IFIP/IEEE Int. Symp. Theoretical Aspects of Software Engineering (TASE), 2008, pp. 3–10.
[5] M. Kwiatkowska, G. Norman, D. Parker, J. Sproston, Verification of real-time probabilistic systems, in: S. Merz, N. Navet (Eds.), Modeling and Verification of Real-Time Systems: Formalisms and Software Tools, ISTE Ltd., John Wiley & Sons, Inc., Hoboken, London, 2008, pp. 249–288.
[6] M. Kwiatkowska, G. Norman, D. Parker, PRISM: probabilistic model checking for performance and reliability analysis, Perf. E. R. 36 (2009) 40–45.
[7] OMNeT++ Simulation Manual, https://omnetpp.org/doc/omnetpp/manual/ accessed 19.07.2017.
[8] OPNET: Manual de usuario, Universitat Politècnica de Catalunya, Departament d'Enginyeria Telemàtica, Secció de l'EPSEVG, Sept. 2004.
[9] A. Legay, B. Delahaye, S. Bensalem, Statistical model checking: An overview, in: Proc. 1st Int. Conf. Runtime Verification (RV), 2010, pp. 122–135.
[10] M. Kwiatkowska, G. Norman, J. Sproston, Probabilistic model checking of the IEEE 802.11 wireless local area network protocols, in: Proc. 2nd Joint Int. Workshop Process Algebra and Probabilistic Methods and Performance Modeling in Verification (PAPMPROBMIV), 2002, pp. 169–187.
[11] C. Daws, M. Kwiatkowska, G. Norman, Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM, Int. J. Soft. Tools Technol. Transfer 5 (2004) 221–236.
[12] M. Duflot, L. Fribourg, T. Herault, R. Lassaigne, F. Magniette, S. Messika, S. Pyronnet, C. Picaronny, Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC, Electron. Notes Theor. Comput. Sci. 128 (2005) 195–214.
[13] T. Bérczes, G. Guta, G. Kusper, W. Schreiner, J. Sztrik, Analyzing a proxy cache server performance model with the probabilistic model checker PRISM, in: Proc. 5th Int. Workshop Automated Specification and Verification of Web Systems, 2009.
[14] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: Proc. 23rd Int. Conf. Computer Aided Verification (CAV 2011), 2011, pp. 585–591.
[15] R. Bakhshi, A. Fehnker, On the impact of modelling choices for distributed information spread, in: Proc. 6th Int. Conf. Quantitative Evaluation of Systems, 2009, pp. 41–50.
[16] PRISM Manual, http://www.prismmodelchecker.org/manual/ accessed 12.01.2017.
[17] M. Fruth, Probabilistic model checking of contention resolution in the IEEE 802.15.4 low-rate wireless personal area network protocol, in: Proc. 2nd Int. Symp. Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), 2006, pp. 290–297.
[18] M. Fruth, Formal methods for the analysis of wireless network protocols, Ph.D. thesis, University of Oxford, 2011.
[19] T. Kapus, Modelling medium access control in IEEE 802.15.4 nonbeacon-enabled networks with probabilistic timed automata, Mob. Inf. Syst. 9 (2013) 157–188.
[20] Part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs), IEEE Std 802.15.4-2006.
[21] T. Kapus, Analysing the effect of CCA duration in 802.15.4 networks with hidden nodes by using PRISM, in: Proc. 23rd Telecommunications Forum (TELFOR 2015), 2015, pp. 87–90.
[22] P.D. Marco, P. Park, C. Fischione, K.H. Johansson, Analytical modeling of multi-hop IEEE 802.15.4 networks, IEEE Trans. Veh. Technol. 61 (2012) 3191–3207.
[23] Z.L. Németh, Model checking of the slotted CSMA/CA MAC protocol of the IEEE 802.15.4 standard, in: Proc. 2010 Mini-Conf. Applied Theoretical Computer Science (MATCOS-10), 2011, pp. 121–126.
[24] M. Duflot, M. Kwiatkowska, G. Norman, D. Parker, S. Peyronnet, C. Picaronny, J. Sproston, Practical applications of probabilistic model checking to communication protocols, in: S. Gnesi, T. Margaria (Eds.), Formal Methods for Industrial Critical Systems: A Survey of Applications, John Wiley & Sons, Hoboken, 2012, pp. 133–150.
[25] A. Hinton, Software project m60: Simulator for the probabilistic model checker PRISM, MEng final year project dissertation, University of Birmingham, 2005.