International Journal on Software Tools Technology Transfer manuscript No.
(will be inserted by the editor)

Marie Duflot · Marta Kwiatkowska · Gethin Norman · David Parker

A Formal Analysis of Bluetooth Device Discovery

Abstract This paper presents a formal analysis of the device discovery phase of the Bluetooth wireless communication protocol. The performance of this process is the result of a complex interaction between several devices, some of which exhibit random behaviour. We use probabilistic model checking and, in particular, the tool PRISM to compute the best- and worst-case performance of device discovery: the expected time for the process to complete and the expected power consumption. We illustrate the utility of performing an exhaustive, low-level analysis to produce exact results in contrast to simulation techniques, where additional probabilistic assumptions must be made. We demonstrate an example of how seemingly innocuous assumptions can lead to incorrect performance estimations. We also analyse the effectiveness of improvements made between versions 1.1 and 1.2 of the Bluetooth specification.

Keywords Bluetooth · Formal verification · Probabilistic model checking

1 Introduction

The use of formal methods for the verification and analysis of real-life systems is becoming increasingly prevalent in industry. With the developments made in recent years, formal verification techniques can now be employed, not just to ascertain the correctness of a system, but also to analyse quantitative properties such as performance and reliability. In this paper, we demonstrate the applicability of an automated formal verification technique called probabilistic model checking to an analysis of the performance of the Bluetooth protocol.

Bluetooth is a wireless telecommunication technology, aimed in particular at low-power devices which communicate over short distances. It is becoming increasingly prominent in devices such as mobile phones, PDAs and laptop computers. To cope with interference, Bluetooth is based on frequency-hopping technology. This means that, before any communication can take place, an initialisation procedure must be carried out, comprising discovery of devices in the vicinity and then exchange of information to synchronise hopping sequences. From a user's point of view, this process affects both the waiting-time and the power usage. Hence, our analysis focuses on this aspect of the protocol.

As will be demonstrated shortly, the time required for completion of the Bluetooth initialisation process is the result of a non-trivial interaction between two devices, motivating the need for a formal, automated analysis. Furthermore, it includes a randomised back-off procedure to resolve contention between devices, and an effective analysis thus needs to be able to reason about the stochastic nature of the system. We use probabilistic model checking and, in particular, the tool PRISM. This process involves construction of a formal probabilistic model from a high-level description of the system, followed by calculation of one or more probabilistic properties, formally expressed in probabilistic temporal logic.

In contrast to approaches based on discrete-event simulation, for which analyses of Bluetooth have already been attempted, formal approaches such as probabilistic model checking involve an *exhaustive* analysis. We construct a complete model of the system under study and use it to compute *actual* performance values, rather than derive estimations from a large number of simulations. As we will show later, this means we can accurately identify worst-case behaviour, as opposed to generating average performance measures which rely on additional probabilistic assumptions. Furthermore, we can establish precisely the situations that lead

Supported by FORWARD and EPSRC grants GR/S46727 and GR/S11107.

Marta Kwiatkowska · Gethin Norman · David Parker
School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK.
Phone: +44 (0)121 4147264
Fax: +44 (0)121 4144281
E-mail: {mzk,gxn,dxp}@cs.bham.ac.uk

Marie Duflot
LACL, Faculté des Sciences et Technologie, Université Paris XII, 61 Avenue du Général de Gaulle, 94010 Créteil, France.
Phone: +33 (0)1 45 17 66 00
Fax: +33 (0)1 45 17 66 01
E-mail: duflot@univ-paris12.fr

to these worst-case scenarios. We will also give an example of a situation where making additional probabilistic assumptions results in inaccuracies in the performance results obtained.

2 Probabilistic Model Checking and PRISM

Probabilistic model checking is an automated technique for the formal verification of systems that exhibit stochastic behaviour. It is based on the construction and analysis of a mathematical model of the system, usually from a specification in some high-level description language. This model generally comprises a set of states, representing all the possible configurations of the system, the transitions that can occur between these states, and information about when and with what probability each transition will occur.

In this paper, the modelling formalism we use is discrete-time Markov chains (DTMCs), where time is modelled as discrete steps and the probability of making each transition is given by a discrete probability distribution. Other model types commonly used are continuous-time Markov chains (CTMCs), Markov decision processes (MDPs), and probabilistic timed automata (PTAs); see [25] for more detailed information about these. We use the probabilistic model checking tool PRISM [12,23]. This allows construction of models via specification in a high-level description language, based on the parallel composition of several modules described in a guarded command notation. We will illustrate the workings of this language in more detail later in the paper.

Models constructed in PRISM are analysed by formally specifying formulas in temporal logic. This allows a precise, unambiguous description of a wide range of properties, such as “the probability of shutdown occurring within 24 hours” or “the long-run probability that the system is stable”. In addition, by assigning real-valued costs (or, conversely, rewards) to states and transitions of the model, we can also reason about, for example, “expected time” or “expected power consumption”.

PRISM automatically ascertains values for such properties by performing probabilistic model checking, which includes both graph-based analysis and numerical computation. For the case of DTMCs, the latter usually constitutes solving a linear equation system of size equal to the number of states in the model. For this, PRISM uses iterative numerical solution techniques such as the Gauss-Seidel method, which are well suited to large problems of this type.

Furthermore, a significant amount of work has gone into the development of efficient, *symbolic* implementation techniques for numerical computation in PRISM. These use data structures such as binary decision diagrams (BDDs) [5] and their extensions, e.g. multi-terminal BDDs (MTBDDs) [6, 1], to allow compact storage and manipulation of extremely large models. We rely heavily on this efficiency for the case study presented in this paper.

The PRISM tool has already been successfully used to perform analysis of, and identify interesting behaviour, in a wide range of case studies. This includes the study of “quality of service” properties for components of real-time probabilistic communication protocols, e.g. IEEE 1394 FireWire [7,16], IEEE 802.3 CSMA/CD [17,8], Zeroconf [13] and IEEE 802.11 wireless LANs [15,24]. It has also been used to verify randomised distributed algorithms for leader election, consensus [14], Byzantine agreement [10], self-stabilisation and mutual exclusion, and probabilistic security protocols for anonymity [27], fair exchange and contract signing [20]. Finally, PRISM has been applied to analysing the performance and reliability of many different types of applications: dynamic power management schemes [19], NAND multiplexing for nanotechnology [18], computer networks, queueing systems, manufacturing processes and embedded systems [11]. The reader is invited to consult the PRISM web site [23] for detailed information and corresponding publications about all of these.

3 Device Discovery in Bluetooth

Bluetooth is a short-range, low-power, open standard for implementing wireless personal area networks. Since it uses the unlicensed 2.4GHz Industry Scientific and Medical band (a set of frequencies almost globally available), there is a potential problem of interference from other devices using this band. To resolve this, Bluetooth uses a frequency hopping scheme, where devices alternate rapidly among the 79 available frequencies in a pseudo-random fashion.

In order to communicate, Bluetooth devices organise themselves into small networks called *piconets*, comprising one *master* and up to 7 *slave* devices, in which the frequency hopping sequences are synchronised and controlled by the master. In this paper, we focus on the issue of piconet creation, the performance of which is crucial because no communication between devices can occur until it is complete. It also has considerably higher power consumption than other parts of the protocol [9], prevents existing device connections from operating and may cause interference to other nearby piconets.

Piconet formation has two steps: firstly, the *inquiry* process, where a master device discovers neighbouring slave devices; and secondly, the *page* process, where connections between them are established. During the first step, information about slave clock times is exchanged for the purposes of synchronisation. This can be used during the second step, which is hence much faster. We therefore concentrate on the inquiry process. We now describe in more detail the procedure executed by an *inquiring device* (a master trying to discover slaves) and a *scanning device* (a potential slave device who wants to be discovered).

3.1 The Inquiring Device

An inquiring device attempts to detect potential slaves in the proximity by broadcasting inquiry packets on a previously

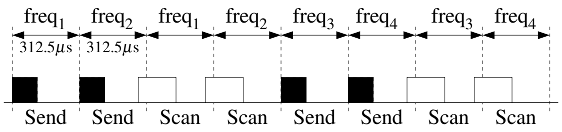


Fig. 1 Timing of the inquiring device's behaviour

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	22	23	24	25	26	27	28	29	30	31	32
17	18	19	20	21	22	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24	25	26	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	30	31	32
17	18	19	20	21	22	23	24	25	26	27	28	29	30	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	16

Fig. 2 Frequency sequences for the inquiring device

agreed sequence of 32 of the 79 available frequencies and scanning for replies. This process continues until some specified bound on the number of replies received or the total time is exceeded.

Like all Bluetooth devices, the inquiring device has a 28 bit free-running clock, which ticks every 312.5μs. On two consecutive 312.5μs time slots, it sends on two sequential frequencies. During the next two time slots, the device scans for a reply on these same two frequencies, i.e. each scan occurs 625μs after the corresponding send (in fact, a 10μs margin is added to the start and end of the scan in case replying devices are not completely synchronised). The device now proceeds to send and scan on the next pair of frequencies in the same fashion. This procedure is illustrated in Figure 1.

The 32 frequencies used for the inquiry procedure are split into two trains, A and B, of 16 frequencies each. The sequence in which the inquiring device sends and scans on these frequencies is determined by its 28 bit clock, denoted CLK, according to the following formula:

$$\text{freq} = [CLK_{16-12} + \mathit{off} + (CLK_{4-2,0} - CLK_{16-12}) \bmod 16] \bmod 32$$

where CLK_{i-j} denotes bits $i,\dots,j$ of CLK, and off is an offset to select whether train A or B is used. The inquiring device swaps between train A and B every 2.56 seconds (the time to send and scan on 16 frequencies is 10ms and each train is repeated 256 times). Furthermore, every 1.28 seconds (every time the 12th bit of CLK changes), a frequency is swapped between train A and B. The whole list of frequencies is shown in Figure 2. Each line of this table is repeated 128 times, taking 1.28 seconds. To simplify the presentation, we have assumed $\mathit{off} = 1$ for train A and $\mathit{off} = 17$ for train B, i.e. initially trains A and B comprise frequencies $1,\dots,16$ and $17,\dots,32$, respectively. Note that the formula given above also incorporates the alternation of sending and scanning on pairs of frequencies, as illustrated previously in Figure 1 (hence the omission of bit 1 in $CLK_{4-2,0}$).

3.2 The Scanning Device

Bluetooth devices that want to be discovered enter the *inquiry scan* substate and periodically scan for inquiry packets on the same 32 frequencies that the inquiring device is transmitting on. To ensure that the frequencies used eventually coincide and that messages are successfully received, the hopping rate of scanning devices is much slower than that of the inquiring device. The frequency of each scanning device, known as its *phase*, cycles through the 32 frequencies in order, according to the value of its clock and changes every 1.28s.

The scanning device listens continuously on its current frequency during an *inquiry scan window* of 11.25ms, long enough for the inquiring device to transmit on an entire train of 16 frequencies. The scanning device then sleeps, before scanning again. This process is repeated periodically. There is some flexibility in the specification [4] as to the length of this period. For our purposes, we have chosen the value 0.64s.

If the scanning device successfully hears a message, by listening on the right frequency at the right time (when the inquiring device is transmitting a packet), it will switch to the *inquiry response* substate, in which it waits 2 time slots (i.e. 625μs) and then sends a reply on the same frequency. A contention problem arises when two devices in inquiry scan try to reply to the same inquiry packet. In this case, the two replies collide and are both lost. To avoid repetition of such a problem, after sending a reply, a device draws a random number $N \in [0,\dots,127]$ and waits for $2{\cdot}N$ time slots before going back to its alternation between sleep and scan states. Note that the maximum random wait is sometimes higher than 127 but, according to the specification [4], this is an appropriate value for our scan period of 0.64s. After each successfully received message, the scanning device also adds one to its phase. Figure 3 summarises the steps of the overall process and the time spent in each.

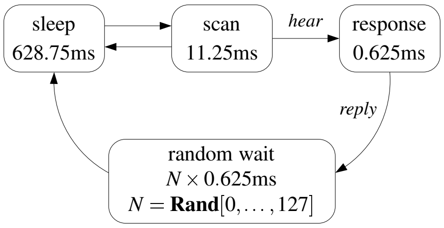


Fig. 3 Timing of the scanning device's behaviour

4 Modelling in PRISM

From the description in the previous section, it should be clear that the performance of the Bluetooth inquiry process, i.e. the time required for messages to be successfully sent and received, is the result of the interaction between two non-trivial sequences of events. This motivates the need for the construction and analysis of a formal model. We now describe how this can be achieved using the tool PRISM and its high-level model description language.

We consider a single inquiring device and a single scanning device, which in this section we refer to as the *sender* and *receiver*, respectively. This restricted scenario has, of course, some limitations. For example, it is not possible for contention to occur between multiple receivers replying to a sender simultaneously. It is worth noting though that, regardless of whether such message collisions actually occur, the randomised back-off procedure must be executed in the case of a single receiver device anyway and it is interesting to study the effect that this has on its performance.

The clocks of both devices are digital, whose time is incremented in discrete steps, corresponding to 312.5μs *slots*, and whose drift can be assumed to be negligible during the relative short inquiry process. This is consistent with the constraints imposed upon possible clock drift in the Bluetooth specification. During the device discovery phase, there is of course no guarantee that the clock ticks of each device are precisely synchronised. However, the timing with which messages are sent and scanned for ensures that the ticks of the two devices can be considered to be aligned and, with the assumption that no drift occurs, that this alignment remains consistent during device discovery.

Since the behaviour of the receiver is probabilistic, the model we construct is a discrete-time Markov chain (DTMC). Note that, as we will discuss later, the model contains no nondeterminism and we can thus avoid using a Markov decision process (MDP), for which analysis is more involved.

4.1 Modelling the Sender (Inquiring Device)

The behaviour of the sender was described in Section 3.1 and illustrated in Figures 1 and 2. The corresponding PRISM code is shown in Figure 4. In the PRISM language, a model's description comprises a number of *modules*, each corresponding to a component of the system being modelled. We specify the sender with a single module.

```
const int mrep; // maximum number of replies
const int mrec=128; // number of repetitions of each frequency sequence
formula hear_reply = (freq=f+16*o); // true when the sender hears a reply
formula c_odd=(c=1,3,5,7,9,11,13,15);
formula f_odd=(f=1,3,5,7,9,11,13,15);

module sender

   s : [1..2]; // local state (1 = sending, 2 = listening)
   f : [1..16]; // current frequency modulo 16
   o : [0..1]; // calculates current frequency (actual frequency = f + o*16)
   c : [1..16]; // used to work out frequency sequences
   rep : [1..mrec]; // number of repetitions of current sequence
   rec : [0..mrep]; // number of replies received

   // sending
   [time] s=1 ∧ f_odd  → (f'=f+1)∧(o'=(f=c) ? 1−o : o);
   [time] s=1 ∧ ¬f_odd → (s'=2)∧(f'=f−1)∧(o'=(f=c−1) ? 1−o : o);
   // receiving
   [time] s=2 ∧ f_odd → (f'=f+1)∧(o'=(f=c) ? 1−o : o);
   [time] s=2 ∧ ¬f_odd → (s'=1)∧(f'=f+1)∧(o'=(f=c) ? 1−o : o);
   [time] s=2 ∧ f=16 ∧ rep<mrep →
      (s'=1)∧(f'=1)∧(o'=(f=c) ? o : 1−o)∧(rep'=rep+1);
   [time] s=2 ∧ f=16 ∧ rep=mrep →
      (s'=1)∧(f'=1)∧(rep'=1)∧(o'=(c_odd) ? o : 1−o)∧(c'=(c=16) ? 1 : c+1);
   [reply] s=2 ∧ hear_reply ∧ rec<mrec → (rec'=rec+1);
   [reply] s=2 ∧ !hear_reply → (rec'=rec);

endmodule
```

Fig. 4 PRISM module representing the sender

The initial part of a module definition lists a set of finite-ranging variables which determine the possible states that the module can be in. The first variable of the *sender* module is s, which keeps track of which step of the protocol the sender is on: when $s{=}1$, it sends two sequential messages on a pair of odd/even frequencies; when $s{=}2$, it scans for a reply on the same two frequencies. The four variables f, c, o and rep keep track of which frequency is currently being used and the position in the frequency sequence of Figure 2. Their exact meaning can be inferred from the comments in Figure 4. The last variable rec counts the number of replies received.

The behaviour of a module is described by a set of guarded commands of the following form:

$$[action]\ guard\ \rightarrow\ update;$$

The *guard* is a predicate over the module's variables (and in fact the variables of all other modules). When the guard is satisfied, the behaviour of the module is determined by the *update*, given in terms of how the values of the module's variable should change. The first guarded command of the *sender* module, for instance, states that if $s{=}1$ and f is odd then f is incremented by 1 and, if $f{=}c$, the value of o is reversed (we use the C-style question mark operator for conditional evaluation). This corresponds to the sender finishing transmission on an odd frequency and changing to the subsequent even frequency.

In the square brackets at the start of each guarded command is an (optional) *action label*. This is used to synchronise with other modules in the PRISM model. More precisely, in a state of the entire model (which is a parallel composition of all modules), transitions of modules corre-

```
const int sleep=2012; // duration (slots) of receiver sleep (628.75ms)
const int scan=36; // duration (slots) of receiver scan (11.25ms)
formula hear = (s=1)∧(freq=f+16∗o); // receiver hears a message

module receiver

  r : [0..3]; // local state (0 = sleep, 1 = scan, 2 = reply, 3 = random delay)
  freq : [0..32]; // frequency scanning on (0 = not scanning)
  y : [0..sleep]; // local clock

  // sleep
  [time] r=0∧y>0 → (y′=y−1); // let time pass
  [] r=0∧y=0 → (r′=1)∧(freq′=phase)∧(y′=scan); // move to scan
  // scan
  [] r=1∧hear → (r′=2)∧(y′=2); // hear a message (move to reply)
  [time] r=1∧y>0∧¬hear → (y′=y−1); // let time pass
  [] r=1∧y=0∧¬hear → (r′=0)∧(freq′=0)∧(y′=sleep); // finished (sleep)
  // reply
  [time] r=2∧y>0 → (y′=y−1); // let time pass
  [reply] r=2∧y=0 → 1/128 : (r′=3)∧(freq′=0)∧(y′=0) // reply and select
                   +1/128 : (r′=3)∧(freq′=0)∧(y′=2) // random delay
                      ⋮
                   +1/128 : (r′=3)∧(freq′=0)∧(y′=254);
  // wait random time
  [time] r=3∧y>0 → (y′=y−1); // let time pass
  [] r=3∧y=0 → (r′=1)∧(freq′=phase)∧(y′=scan); // finished (scan)

endmodule
```

Fig. 5 PRISM modules representing the receiver

```
// time slots until frequency changes (1.28 seconds)
const int z_max=4096;

module receiver_frequency

  phase : [0..32]; // phase (next frequency for receiver)
  z : [1..z_max]; // clock for phase

  // update frequency: one time slot elapses
  [time] z<z_max → (z′=z+1);
  [time] z=z_max → (z′=1)∧(phase′=(phase<32)?phase+1 : 1);
  // update frequency: something is sent by the receiver
  [reply] true → (phase′=(phase<32)?phase+1 : 1)

endmodule
```

Fig. 6 PRISM module that computes the phase of the receiver

sponding to guards labelled with identical actions will occur simultaneously in a single global transition. One example is the *time* action, which labels many of the commands. Since we can assume that Bluetooth devices all operate at the same clock-speed, we model time elapsing in a synchronous fashion. For each 312.5μs time slot which passes, all modules synchronise on a *time* action, with the transition of each module reflecting the changes that occur in that time slot. All other commands (some of which are synchronous, see e.g. the *reply* action used in Figure 4, and some of which are asynchronous) are assumed to correspond to an instantaneous change in state.

The code also illustrates other features of PRISM such as the use of *constants* (e.g. *mrep*), which allow definitions of fixed values to be kept separate (and possibly left undefined until run-time), and *formulas* (e.g. *hear_reply*), which allow complex expressions to be defined once and then reused.

4.2 Modelling the Receiver (Scanning Device)

The behaviour of a receiver was previously summarised in Figure 3. For convenience, the corresponding PRISM model comprises two modules: *receiver* (Figure 5), the main part of the receiver's behaviour; and *receiver_frequency* (Figure 6) which keeps track of the frequency that the receiver will use next time it starts a scan (determined by the device's clock).

The format of the guarded command notation used has already been explained in the previous section. One new feature is the method for specifying probabilistic choice. This can be seen in Figure 5 in the command labelled with the guard "$r{=}2 \wedge y{=}0$". Several possible updates are given, each with an associated probability. Here, this represents the receiver drawing the random number N (see Section 3.2).

Note that these two modules again have commands labelled with the *time* and *reply* actions, which causes all three modules to make these transitions synchronously. Note also that some commands in the *receiver* module have no action label. This means that they occur independently from the other modules. However, the fact that this is the only module with such commands, combined with the fact that the behaviour of each individual module is always deterministic (i.e. all of its guards are disjoint), means that the overall model contains no nondeterminism, and is hence suitable for representation as a DTMC.

4.3 Reducing Model Complexity

The PRISM code in Figures 4–6 is intended to provide a clear description of our model. In fact, we have made a number of subsequent optimisations, which we describe in this section. The final version of the code can be found in the case studies section of the PRISM website [23]. The changes made provide an extremely useful increase in efficiency. For example, in one of the experiments performed, our optimisations reduce the state space from 941,022 million states to 3,394 million, the model construction time from 1 hour to 1 minute, and the model checking time from over 15 hours to 2 seconds.

We observe here that a simple, and commonly used, approach to the abstraction of models with large constants, such as those found in this case study, is to scale down *all* constants by the same factor, giving an abstraction of the original model. However, for the model of Bluetooth device discovery, such a simplification is not feasible. This is due to the fact that the process includes complicated sequences of events, including both actions that take a large number of time-slots and those performed in a single slot.

Another good candidate for reducing model size would be to apply probabilistic bisimulation techniques. Unfortunately, in this case the size of the unreduced models renders the available bisimulation algorithms infeasible. We have thus had to resort to alternative, model-specific techniques to reduce model complexity. These are described in the following sections.

Aggregation of sleep transitions. The first and most effective optimisation is based on the fact that, when there is only one receiver, as in our model, the behaviour of the protocol when this receiver is in its sleep state is completely deterministic. The time that the receiver will spend sleeping is exactly 2012 time slots (corresponding to 628.75ms, the difference between the 0.64s cycle between sleep/scan states and the 11.25ms inquiry scan window). During this fixed-length period, the corresponding shift of the sender along its sequence of frequencies and the change in phase of the receiver are both relatively easy to compute.

We can thus compact the sequence of 2012 individual transitions which occur during this time into a single transition. This decreases both the number of states and transitions in the model. Furthermore, it allows the maximum value of the clock y to be reduced from 2012 to 254 (the maximum random delay), giving an additional decrease in model complexity. Since we are simply replacing a deterministic path of transitions through the model with a single step, the behaviour remains unchanged. As will be explained later, timing characteristics of the model are captured by assigning costs to transitions which correspond to time elapsing. We will hence assign a cost of 2012 to such transitions.

Prediction of scan success. In fact, we can extend the previous optimisation. At the point where the receiver begins a period of scanning, we can determine, from the global state of the model (i.e. the position of the sender in its sequence of frequencies and the phase of the receiver), whether a message will be successfully heard during the forthcoming scan. In cases where it will not, we can skip the scan and incorporate the 36 time slots which would have been spent into the subsequent 2012-slot sleep transition. By making this change, the total jump becomes 2048 slots which, being a multiple of the train length, simplifies the computation of the sender's next frequency over this period.

To incorporate this latter optimisation into our model, we add a formula *success* which is true if and only if the receiver will hear a message during its next scan. Then, just before the receiver enters its scan state, we check the value of *success* and, when the formula is true, we let the receiver scan until hearing a message (which *success* guarantees will occur within *scan* time slots). Otherwise, we let the receiver synchronise with all other components of the model to perform a transition where all variables are updated to reflect 2048 time-slots elapsing. This simplification also means that there is no need to explicitly track the amount of time that receiver spends scanning (using variable y in module *receiver*) since, when it does begin to scan, it is guaranteed to hear a message within 36 time slots. For the precise details of these changes to the model source code, see the section on this case study on the PRISM website [23].

Frequency of replies. Finally we note that, by the design of the protocol itself, the frequency upon which the receiver sends a reply to the sender will always be equal to the frequency on which the sender is currently listening. Recall, from Figure 1, how the sender transmits on each frequency and then listens on it exactly two time-slots later. Observe also, from Figure 3, how the receiver waits exactly 2 time-slots (0.625ms) before sending a reply using the frequency on which it has just received a message. Thus, the reply from the receiver will always be successfully heard by the sender. At the model level, this means that the formula *hear_reply* can be removed from the second last guarded command in Figure 4, and that the the last guarded command can be removed entirely.

5 Experimental Results

Our primary concern is the performance of the Bluetooth inquiry process, i.e. the time required for a master device to successfully receive replies from listening slave devices. This is affected by the number of times the receiver sleeps before successfully scanning on the right frequency at the right time, and the random delays selected. More specifically, since the protocol is probabilistic in nature, we compute the expected time for the inquiry process to complete, with completion occurring when the number of replies received reaches a predetermined bound. We also compute the corresponding expected power consumption.

We performed an analysis of the model described in the previous section with the most recent version the PRISM model checker, which allows us to assign costs to states and transitions of the model and then compute, for example, the expected cumulated cost before reaching some set of states. In our model the costs we measure are time and power consumption. For example, in the case of time, we assign a cost of 1 to all *time* actions and 0 to all others. As described in Section 4.3, we actually sometimes accumulate multiple time steps into a single step. The costs of these transitions are modified accordingly.

The first issue we must address is the initial configuration of the model. We cannot assume that the sender and receiver both start in some fixed state because this is unrealistic. For efficiency reasons, we restrict ourselves to the case where the sender is already transmitting inquiry packets and a receiver begins scanning after some unknown delay, which is a reasonable scenario. We can hence fix the initial state of the receiver (i.e. variables r, *freq*, and y). We can also suppose that the variable *rec* of the *sender* module initially takes the value 0 since this cannot increase until after the receiver begins scanning. Note that we cannot fix the actual phase of the receiver since this is determined by its clock, whose value could be anything when it first begins scanning. This leaves us with $2 \cdot 16 \cdot 2 \cdot 16 \cdot 128 \cdot 4096 \cdot 32 = 17,179,869,184$ possible initial states.

Since formal verification aims to be exhaustive, we must consider all of these, although clearly it is not feasible to treat each one separately. Fortunately, we can deal with this in PRISM by building a single DTMC with multiple initial states and then examining the results of model checking for all these states. In fact, this required a modification to PRISM, but only a trivial one: the tool always computes results for all states in the model anyway. Initially, this single model proved too large for the workstation we were using (a Pentium 4 with 1 GB RAM and 2.80GHz processor). However, by partitioning the set of all initial states into classes (we fix variable *phase*, giving 32 sets of $536,870,912$ states each), we reduce the problem to 32 separate instances of model checking, each of which *was* feasible.

As we will see from the results in the following paragraphs, though, despite this partitioning (and despite the reductions in model complexity described in Section 4.3), this still results in models with extremely large state spaces. Intuitively, this is because the Bluetooth devices execute complicated sequences of events, comprising both actions that take a large number of time-slots and those performed in a single slot. Fortunately, there remains a certain degree of regularity in the process and we are able to exploit this using PRISM's symbolic (BDD-based) implementation. With an approach based on explicit data structures (e.g. sparse matrices) this would not be feasible.

5.1 Time for a Single Reply

We first present results for the time required for the receiver to successfully send a single reply. In fact, since the receiver does not make a random choice until after it first replies to a message, the expected time computed is in this case the exact time required. The 32 models we constructed each contained approximately 3.4×10^9 states, required less than a minute to construct, and took 1–2 seconds to analyse.

We computed the expected time to send a message for all possible initial configurations. Using PRISM, we were able to extract information about the best- and worst-case scenarios. The latter, for example, can be achieved with the following query to PRISM:

$$\mathtt{R}=?\,[\,\mathtt{F}\ rep{=}mrep\ \{\text{"}\mathtt{init}\text{"}\}\{\mathtt{max}\}\,]$$

In the above, the notation $\mathtt{R}=?\,[\,\mathtt{F}\ \phi\,]$ corresponds to the expected cumulated cost (in this case, time) of the system until condition ϕ is satisfied. The condition *rep*=*mrep* identifies states of the model where a sufficient number of replies (in this case, one) have been received. Finally, the notation $\{\text{"}\mathtt{init}\text{"}\}\{\mathtt{max}\}$ indicates that we are requested the maximum (worst-case) expected cost value over all initial states of the model.

Our results show that the minimum time for a single reply is 625μs (2 slots), which corresponds to the cases when the receiver starts listening on the frequency that the sender is currently sending messages on, and therefore the receiver sends a reply after waiting 2 slots. The maximum time is 2.5716s (8,229 slots) and is achieved in 860,160 of the possible initial states. This corresponds to the situation where the receiver does not hear the sender until it scans for the fifth time and therefore sleeps 4 times.

We can use PRISM to identify precisely the way in which this maximum time arises. The PRISM query given above causes the tool to report not only the maximum expected time, but also the actual states from which this value is observed. Using this information, the model can then be inspected in an attempt to explain the behaviour. In fact, we have made use of a prototype discrete-event simulation engine which has been developed for PRISM. This allows us to generate, either manually or automatically, a random path through the model starting from a given state. Below is an example execution which illustrates exactly how this can arise.

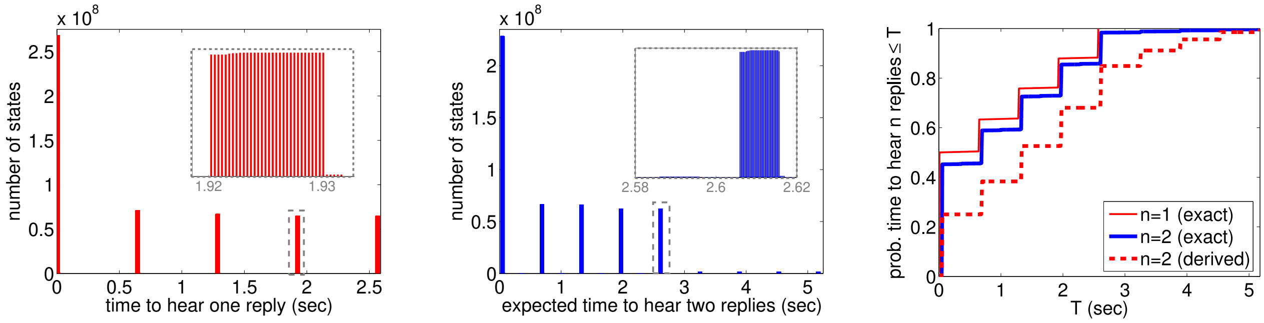
Example 1 Suppose that the receiver starts its first scan on frequency 1 and that its phase is about to change. Suppose also that the sender is performing its last repetition of the first of the following frequency sequences:

1	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
17	18	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	21	22	23	24	25	26	27	28	29	30	31	32

and has already sent on frequency 1 during this repetition. During this scan, the sender will finish the sequence and start the following one which does not contain 1. The receiver will not hear anything during this scan and will therefore enter sleep. When the receiver wakes, the sender will still be using the second frequency sequence and the receiver will now be scanning on frequency 2 (as its phase was about to change when it first scanned), and therefore the receiver will not hear anything and sleep again. In fact, since the sender's subsequent frequency sequence does not contain 2 or 3, the receiver will not hear anything on either of these two sequences as, during this time, the receiver will be scanning on 2 or 3. Only when the receiver wakes for the fourth time will its scan be successful on frequency 3 because the sender will by then be using the fourth sequence.

In Figure 7(a) we have plotted the time until the sender hears a reply against the number of initial states that result in this time. The discontinuities in the graph are to be expected and follow from the fact that, when the receiver does not hear anything during a scan (36 time slots), it sleeps for 2,012 slots before scanning again. Since, in the worst case, the receiver sleeps 4 times before hearing something from the sender, there are 5 peaks in the graph corresponding to the receiver sleeping from 0 up to 4 times. The inset in Figure 7(a) illustrates one of these peaks more clearly. The width of each peak is 11.25ms (36 slots).

If we make the assumption that, when the receiver first starts to listen, there is a uniform distribution on the set of possible initial configurations, we can calculate the cumulative probability distribution function for the time for the sender to hear a reply. This is the first plot shown in Figure 7(c). Furthermore, from this distribution, we extracted



(a) distribution over initial states (one reply)

(b) distribution over initial states (two replies)

(c) cumulative distributions ($n =$ number of replies)

Fig. 7 Expected time for the sender to hear one or two replies from the receiver

Table 1 Probability that the receiver sleeps at most K times before sending n replies

K	Bluetooth v.1.2			Bluetooth v.1.1
	$n=1$	$n=2$		$n=1$
	Exact	Exact	Derived	Exact
0	0.500305	0.461240	0.455379	0.250305
1	0.633575	0.596265	0.590829	0.383657
2	0.759062	0.731585	0.728684	0.526981
3	0.879674	0.857913	0.855329	0.681114
4	1	0.984295	0.984218	0.849408
5	1	0.988269	0.988294	0.911750
6	1	0.992398	0.992514	0.956496
7	1	0.996294	0.996519	0.985521
8	1	1	1	1

the probability that the receiver sleeps at most K times before sending its first reply to the sender, for a range of values of K. These figures are shown in the first column of Table 1.

5.2 Time for Two Replies

For the case where the sender waits until two replies have been received, the 32 constructed models each have approximately 5.6×10^{10} states and took roughly 80 minutes to build and 165 minutes to model check. The minimum expected time, over all possible initial configurations, for the sender to hear two replies is 0.0456s (146.0 slots). The maximum is 5.177 seconds (16,565 slots) and 518 of the possible initial states result in this. This is possible since the receiver can sleep up to 8 times before sending its second reply. We now extend Example 1 to illustrate how this can occur.

Example 2 Example 1 demonstrated the receiver sleeping 4 times and then finally hearing the sender when it is on frequency 3 in the sequence:

```
1 2  3  4  21  22  23  24  25  26  27  28  29  30  31  32
```

After the receiver scans, it increases its phase by 1 and waits a random delay before scanning again. During this random delay the phase will increase again and the receiver will next scan on frequency 5. Since this frequency does not appear on the above sequence and the sender has just started using it, it will not hear anything during this scan and the next one, and will therefore sleep twice. When the receiver wakes again it will scan on frequency 6 and since the sender's subsequent frequency sequences will be:

```
 1  2   3   4   5  22  23  24  25  26  27  28  29  30  31  32
17 18  19  20  21  22   7   8   9  10  11  12  13  14  15  16
```

it follows that the receiver will sleep an additional 2 times before finally hearing (for the second time) from the sender while scanning on frequency 7.

In Figure 7 we have also plotted, for the expected time for the receiver to reply to two messages, both the distribution over the states (Figure 7(b)) and the cumulative distribution function (second plot in Figure 7(c)). For the latter, as in Section 5.1, we assume that there is a uniform probabilistic choice as to the state of the system when the receiver first scans. We have also extracted the probability that the receiver sleeps at most K times before sending its second reply to the sender, for a range of values of K. This is shown in the second column of Table 1. The discontinuities in Figure 7(b) are, like those in Figure 7(a), due to the time that the receiver spends in sleep: in this case there are 9 peaks – the last 4 being considerably smaller than the first 5 but still visible – which correspond to the cases when the receiver sleeps from 0 to 8 times before sending its second reply. The inset again illustrates one of these peaks. Closer examination reveals that there are also some initial states for which the expected time falls outside of the eight peaks.

In Figure 7(c) we have also included a third cumulative distribution function, for the probability that the sender hears two replies by time T, derived from the earlier obtained distribution for one reply (shown on the same graph). This derivation is based on the assumption that the time to hear the second message is independent of the time to hear the first message and is obtained by taking the convolution of two copies of the original distribution together with a distribution representing the random delay made by the receiver between sending the first reply and beginning its next scan. The corresponding probabilities of sleeping at most K times before sending two replies are again shown in Table 1 (third column). Interestingly, these results demonstrate that the assumption that the time to reply to the second message is independent of the time to reply to the first is incorrect, i.e. leads to inaccurate results. More precisely, the results show that if the receiver sleeps before sending its first reply, it is less likely to sleep the second time.

We have also attempted to compute the probabilities for higher numbers of replies. However, in these cases, partitioning into 32 models is not feasible. We have been able to generate some results for up to 5 replies by partitioning the initial states more finely, but the considerable growth in the number of verifications required means that an exhaustive analysis has yet to be completed.

5.3 Power Consumption

We have also performed an analysis of the expected power consumption of the Bluetooth device discovery process. To do so, we need only change the costs associated with transitions of the model from the elapsed time to the power consumed. Furthermore, since during device discovery the only change in power consumption is caused by the receiver changing power mode[1], we restrict attention to the power consumed by the receiver (the power consumed by the sender can be calculated directly from our expected time results).

Using the values given in [9], which are 100mW for active mode in Inquiry-Scan and 50mW for standby, the results obtained are presented in Figure 8. These are very similar to the expected time results, with smaller gaps between two peaks (as less power is consumed during sleep).

5.4 Comparison with Bluetooth Version 1.1

Our analysis of Bluetooth device discovery is based on version 1.2 of the Bluetooth specification, the most recent at the time of modelling. We have also carried out a comparison of this and the previous version (1.1). The main difference in terms of the inquiry protocol is that, in version 1.1, the receiver only sends replies to every second message received. More precisely, the receiver's behaviour is as follows: when receiving the first message from the sender, it draws a random number, waits for the corresponding random delay and returns to its scan state. Only when receiving a message for the second time does it reply and increase its frequency. This random delay was chosen to avoid collisions. In fact, it does not take much more time (two slots) to send a reply to the first message received before the random delay and there is a chance that this reply will be successfully received by the sender. This is the reasoning behind this difference between version 1.2 and version 1.1.

We modified our model to reflect these changes and recomputed the expected time for the inquiry process. The results are included in Table 1. Unsurprisingly, we see that the time for one reply in version 1.1 and for two replies in version 1.2 are very similar. However, we successfully illustrate that, as was intended, version 1.2 indeed results in improved performance.

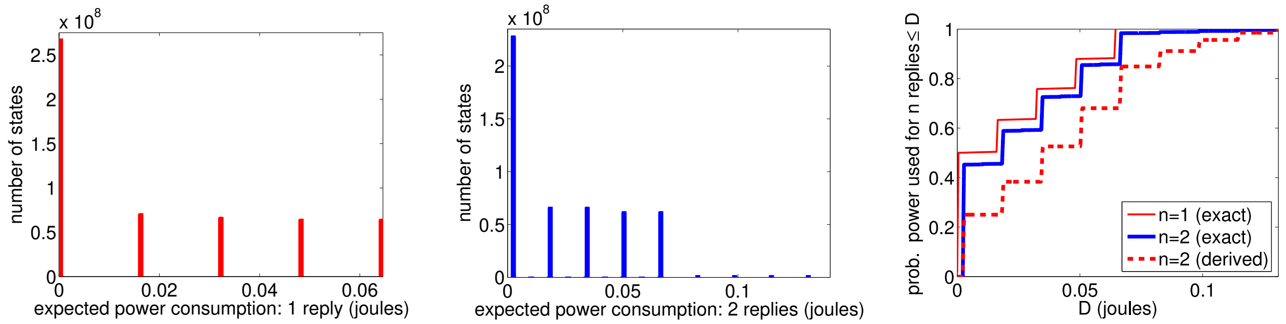
6 Related Work

Thanks to the ongoing growth in popularity of Bluetooth technology, an increasing amount of research is being carried out in order to analyse and improve its efficiency. There is, however, a limited amount of work regarding the inquiry process of the protocol. To our knowledge, this paper is the first application of formal verification to the area: the most common form of analysis being used is simulation, with tools such as ns-2 [21] and BlueHoc [3]; see for example [28,2].

Elsewhere, attempts have also been made to compute the time required for inquiry analytically. Two examples include [26] and [29]. In the former, the emphasis is primarily on the issue of scatternet formation, but they also discuss a symmetric variation of the Bluetooth inquiry process, and consider analytic expressions for the completion time, comparing them to that of the standard asymmetric version. The latter considers the standard version, but for an arbitrary number of devices, rather than just two. The authors also complement this with results from a discrete-event simulation. Both papers, however, take a far more simplistic approach to modelling the inquiry process than us. Firstly, they assume that the sender uses a single train of 32 frequencies which remains constant throughout, drastically reducing the complexity of its behaviour. They therefore also assume that the receiver will always be able to listen to all frequencies in a short period, and thus never need to sleep. Hence, in their analysis, they take the frequency synchronisation delay to be uniformly distributed.

A more comprehensive analysis was recently presented in [22]. In similar fashion, the authors produce an analytic expression for the probability distribution function of the time to complete an inquiry and then use this to validate a discrete-event simulation of the same model. Like us, they consider the correct behaviour in terms of trains (although some simplifications are still required in order to derive an analytic expression), and hence the model is much closer to ours. The most significant difference is that, whereas we aim for an exact/worst-case analysis, considering all possible overlaps between the sender and receiver, they assume an equiprobable distribution between all possibilities in order to derive a probability function. This, combined with the fact that they use different parameters to those used here (both are compliant with the official specification), means that a

[1] The receiver has two power modes: *active* (used when the receiver is sending/receiving) and *standby* (used the receiver is sleeping/waiting a random delay).



(a) distribution over initial states (one reply)

(b) distribution over initial states (two replies)

(c) cumulative distributions ($n =$ number of replies)

Fig. 8 Expected power consumed before the sender hears one or two replies from the receiver

detailed comparison of the two sets of results is impractical. Unfortunately, the publication describing the derivation of the distribution function is not yet available. In future, we plan to carry out a more comprehensive comparison of this work with our own.

It is also worth noting that all of the related work mentioned above is based on either version 1.0 or 1.1 of the specification. In this paper, we focus on the subsequent version, 1.2.

7 Discussion and Conclusion

In this paper, we have presented a formal analysis of the performance of Bluetooth device discovery, using probabilistic model checking and the tool PRISM. We showed how this permits an exhaustive analysis of a low-level model of the specification. This allows us to examine the best- and worst-case expected times for the inquiry process and identify exactly how these situations can occur.

We are, however, limited to a certain extent by the huge size of the probabilistic models that we need for this process. Techniques based on discrete-event simulation are far less susceptible to this phenomenon, but have two disadvantages: firstly, they compute only approximations to the numerical results we obtain; and secondly, they sometimes require additional probabilistic assumptions (in this case study, on the initial configuration of the Bluetooth devices). It would be advantageous to compare or even combine the two approaches. One interesting application of the results in this paper might be to use our exact results to improve the accuracy of a simulation of Bluetooth performed at a higher level. We plan to investigate this area further.

There are also several other directions in which we we would like to extend this work. Two examples are: increasing the number of messages received and increasing the number of receivers. The latter introduces several new dimensions such as collision of messages between devices and tracking which replies correspond to which receivers. Lastly, it would be interesting to study the effect of noise and/or interference on the inquiry procedure. Since all of these areas lead to an increase in model size, we will almost certainly need to consider additional techniques, such as combination with simulation (as discussed above) or abstraction and symmetry reduction methods.

References

1. Bahar, I., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Formal Methods in System Design **10**(2/3), 171–206 (1997)
2. Basagni, S., Bruno, R., Petrioli, C.: Device discovery in Bluetooth networks: A scatternet perspective. In: Proc. Networking 2002, *LNCS*, vol. 2345, pp. 1087–1092. Springer (2002)
3. BlueHoc - An open-source Bluetooth simulator: www-124.ibm.com/developerworks/opensource/bluehoc
4. Bluetooth specification, version 1.2, Bluetooth SIG, 2003, www.bluetooth.com
5. Bryant, R.: Binary decision diagrams and beyond: Enabling technologies for formal verification. In: Proc. International Conference on Computer-Aided Design (ICCAD'95), pp. 236–243 (1995)
6. Clarke, E., Fujita, M., McGeer, P., McMillan, K., Yang, J., Zhao, X.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods in System Design **10**((2/3), 149–169 (1997)
7. Daws, C., Kwiatkowska, M., Norman, G.: Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. International Journal on Software Tools for Technology Transfer (STTT) **5**(2–3), 221–236 (2004)
8. Duflot, M., Fribourg, L., Hérault, T., Lassaigne, R., Magniette, F., Messika, S., Peyronnet, S., Picaronny, C.: Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. In: Proc. 4th Workshop on Automated Verification of Critical Systems (AVoCS'04), *Electronic Notes in Theoretical Computer Science*, vol. 128(6), pp. 195–214. Elsevier Science (2004)
9. Kasten, O., Langheinrich, M.: First experiences with Bluetooth in the Smart-Its distributed sensor network. In: Proc. PACT'01 (2001)
10. Kwiatkowska, M., Norman, G.: Verifying randomized Byzantine agreement. In: D. Peled, M. Vardi (eds.) Proc. Formal Techniques for Networked and Distributed Systems (FORTE'02), *LNCS*, vol. 2529, pp. 194–209. Springer (2002)
11. Kwiatkowska, M., Norman, G., Parker, D.: Controller dependability analysis by probabilistic model checking. In: Proc. 11th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'04) (2004)

12. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 2.0: A tool for probabilistic model checking. In: Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04), pp. 322–323. IEEE Computer Society Press (2004)
13. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. In: K. Larsen, P. Niebert (eds.) Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03), *LNCS*, vol. 2791, pp. 105–120. Springer-Verlag (2003)
14. Kwiatkowska, M., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: G. Berry, H. Comon, A. Finkel (eds.) Proc. 13th International Conference on Computer Aided Verification (CAV'01), *LNCS*, vol. 2102, pp. 194–206. Springer (2001)
15. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: H. Hermanns, R. Segala (eds.) Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02), *LNCS*, vol. 2399, pp. 169–187. Springer (2002)
16. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. Formal Aspects of Computing **14**(3), 295–318 (2003)
17. Kwiatkowska, M., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. In: Y. Lakhnech, S. Yovine (eds.) Joint Conference on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT), *LNCS*, vol. 3253, pp. 293–308. Springer (2004)
18. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S.: Evaluating the reliability of NAND multiplexing with PRISM. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **24**(10), 1629–1637 (2005)
19. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing **17**(2), 160–176 (2005)
20. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. In: A. Abdallah, P. Ryan, S. Schneider (eds.) Proc. BCS-FACS Formal Aspects of Security (FASec'02), *LNCS*, vol. 2629, pp. 81–96. Springer (2003)
21. ns-2: The Network Simulator, www.isi.edu/nsmam/ns
22. Peterson, B., Baldwin, R., Kharoufeh, J.: A specification-compatible Bluetooth inquiry simplification. In: Proc. Hawaii Int. Conference on System Sciences (HICSS'04) (2004)
23. PRISM web site. www.cs.bham.ac.uk/˜dxp/prism
24. Roy, A., Gopinath, K.: Improved probabilistic models for 802.11 protocol verification. In: Proc. 17th International Conference on Computer Aided Verification (CAV'05), *LNCS*, vol. 3576, pp. 239–252. Springer (2005)
25. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, P. Panangaden and F. van Breugel (eds.), *CRM Monograph Series*, vol. 23. American Mathematical Society (2004)
26. Salonidis, T., Bhagwat, P., Tassiulas, L., LaMaire, R.: Proximity awareness and ad hoc network establishment in Bluetooth. Tech. Rep. TR 2001-10, Institute of Systems Research, University of Maryland (2001)
27. Shmatikov, V.: Probabilistic model checking of an anonymity system. Journal of Computer Security **12**(3/4), 355–377 (2004)
28. Siegemund, F., Rohs, M.: Rendezvous layer protocols for Bluetooth-enabled smart devices. In: Proc. Conference on Architecture of Computing Systems - Trends in Network and Pervasive Computing (ARCS'02). Springer (2002)
29. Záruba, G., Gupta, V.: Simplified Bluetooth device discovery - Analysis and simulation. In: Proc. Hawaii Int. Conference on System Sciences (HICSS'04) (2004)