Evaluating complex MAC protocols for sensor networks with APMC

Michaël Cadilhac [1] Thomas Hérault [2] Richard Lassaigne [3]
Sylvain Peyronnet [1,3] Sébastien Tixeuil [2]

Abstract

In this paper we present an analysis of a MAC (Medium Access Control) protocol for wireless sensor networks. The purpose of this protocol is to manage wireless media access by constructing a Time Division Media Access (TDMA) schedule. APMC (Approximate Probabilistic Model Checker) is a tool that uses approximation-based verification techniques in order to analyse the behavior of complex probabilistic systems. Using APMC, we approximately computed the probabilities of several properties of the MAC protocol being studied, thus giving some insights about it performance.

Key words: Wireless sensor networks, approximate verification

1 Introduction

Wireless sensor networks are networks using a large number of machines that embed processors, sensors, actuators and radio communication capabilities. These networks are usually used for environment monitoring and asset tracking. Individual nodes are usually deployed in an *ad hoc* maneer, so they must organize themselves to form a multi-hop wireless communication network [18].

A major issue in such self-organized wireless networks is the access to communication medium, as simultaneous wireless transmissions between neighboring nodes result in collisions that garble exchanged messages. Collision management and avoidance are fundamental issues in wireless network protocols, and medium access control (MAC) protocols are distributed solutions to this problem.

In [4], Busch, Magdon-Ismail, Sivrikaya and Yener propose a MAC protocol for sensor networks. This protocol provides many interesting properties: it

[1] EPITA Research & Development Laboratory ({`cadilhac|peyronnet`}`@lrde.epita.fr`)
[2] University Paris XI, INRIA Grand-Large ({`herault|tixeuil`}`@lri.fr`)
[3] Equipe de Logique,CNRS & University Paris VII (`lassaign@logique.jussieu.fr`)

gives guarantees about the bandwidth that is allocated to each node, it is fully distributed, and does not require the existence of a global clock.

To analyze the correctness and performance of such distributed protocols, a lot of methods can be used. One of them is probabilistic model checking, a class of algorithmic methods for the verification of probabilistic systems with respect to quantitative properties. Most of these methods are based on the construction of a mathematical model of the system and on the expression of the specification in some temporal language. This model represents all the possible configurations of the system, and the probabilities of the transitions that can occur between these states.
The problem of evaluating the satisfaction probability of a temporal property to be checked, is reduced to the resolution of a system of linear equations over the state space. However, due to the state space explosion phenomenon during the modeling step, the representation of the transition matrix can be so large that the verification becomes intractable. To overcome this phenomenon, symbolic and numerical methods have been introduced in tools such as PRISM [14]. In the last years a completely different model checking technique emerged: Approximate Probabilistic Model Checking. Using this technique we can approximately compute the probability that a model verify a specification [10]. With this method, the computation time is not necessarily lowered, but the memory consumption becomes very low (or constant in some cases). Indeed the space complexity of the method is independent of the size of the model.

The results we present in this paper are twofold: we model a complex contention-free MAC protocol for wireless sensor networks [4], and we perform various experiments with this model. We thus show the interest of using approximate probabilistic model checking for the verification and analysis of protocols for sensor networks.

The structure of the paper is as follows. In Section 3 we review the framework of approximate probabilistic verification and present the tool APMC. Then we briefly describe the MAC protocol of [4] (Section 4) and its modeling (Section 5). Finally, Section 6 gives results of several experiments performed on our model using APMC.

2 Related Work

Networks now being imagined for sensors [26] and small devices [5] require energy conservation, scalability, tolerance to transient faults, and adaptivity to topology change. There essentially exists two kinds of MAC algorithms for sensor networks [24]:

(i) *contention-based protocols*: nodes compete for a shared channel, resulting in probabilistic coordination, *e.g.* ALOHA [2] and carrier sense multiple access (CSMA [13]). The CSMA protocol has been widely studied and extended, and is at the core of widely used standards such as IEEE 802.11 [1].

(ii) *scheduled protocols*: nodes collaborate to plan communications so that no collisions eventually occur. This plan can be based on frequency (FDMA, frequency division multiple acces), code (CDMA) or time (TDMA), that are widely used in modern cellular communication systems [20]. Essentially, collisions are avoided by scheduling communications in virtual sub-channels that are obtained by dividing the real channels either by time, frequency, or orthogonal codes. Since virtual sub-channels do not interfere with each other, MAC protocols in this group are essentially collision-free.

Time Division Media Access (TDMA) is a reasonable technique for managing wireless media access, however the priorities of scalability and fault tolerance are not emphasized by most previous research. Recent analysis [9] of radio transmission characteristics typical of sensor networks shows that TDMA may not always substantially improve bandwidth when compared to randomized collision avoidance protocols, however fairness and energy conservation considerations remain important motivations. In applications with predictable communication patterns, a sensor may even power down the radio receiver during TDMA slots where no messages are expected; such timed approaches to power management are typical of the sensor regime. Among TDMA protocols especially designed for sensor networks, [11,12] were studied both theoretically and practically (to get quantitative measures), while the rather complex protocol presented in [4] was only studied in a theoretical way, in particular a quantitative analysis was not provided.

The research in the field of methods for approximating probabilistic model checking is quite young and there is only a few other approaches than our. In [25], a procedure is described for verifying properties of discrete event systems based on Monte-Carlo simulation and statistical hypothesis testing. In [22], a statistical method is proposed for model checking of black-box probabilistic systems. These approaches differ strongly from ours by using statistical hypothesis testing instead of randomized approximation schemes. More recently, in [8], a randomized algorithm for probabilistic model checking of safety properties expressed as LTL formulas was given. In another approach [19] both random testing and abstract interpretation are used for the verification of C programs. Only a few things were done for verifying communications protocols using approximation-based model checking. For instance, APMC was already used for this purpose in [7,6]. A lot of work have been done on the formal verification of communication protocols using probabilistic model checking. One can mention the work of the PRISM team (see, for instance, [15,16]).

3 Approximate Probabilistic Verification and APMC

In this section, we recall the framework of approximate probabilistic verification and present quickly the tool APMC (Approximate Probabilistic Model Checker). APMC is based on a randomised algorithm to approximate the satisfaction probability of a temporal specification, by using sampling of executions paths of the system.

3.1 Models and specifications

The approximation method of APMC can handle any probabilistic system that supports execution path generation. The input language is the same as PRISM and allows to describe in a modular way either discrete-time or continuous-time Markov chains. The specification language allows to reason about satisfaction probabilities of temporal logic formulas.
Let $\mathcal{M}$ be a discrete-time Markov chain, s be an initial state, and ψ be a linear temporal logic formula. We denote by $Path(s)$ the set of execution paths whose first state is s. The probability measure $Prob$ over the set $Path(s)$ is defined classicaly and we denote by $Prob[\psi]$ the measure of the set of paths satisfying the formula ψ. Let $Path_k(s)$ be the set of all paths of length $k > 0$ starting at s and $Prob_k[\psi]$ be the measure of paths satifying ψ in $Path_k(s)$.
Using the approximation method of APMC, we can approximate, with any degree of accuracy, the satisfaction probability of a temporal formula. In the next subsection, for the sake of clarity, we consider only bounded temporal properties and we describe a randomised algorithm to approximate $Prob_k[\psi]$.

3.2 Approximate verification

In order to estimate the probability p of a bounded property ψ with a randomised algorithm, we generate random paths of $Path_k(s)$ and compute a random variable X which estimates $p = Prob_k[\psi]$. We say that an estimation X is ε-good means if output value of the algorithm is in $[p - \varepsilon, p + \varepsilon]$.

Definition 3.1 A *randomised approximation scheme* (RAS) for p is a randomised algorithm $\mathcal{A}$ that takes as input a representation of the system, a property ψ, two real numbers $\varepsilon, \delta > 0$ and produces a value X such that: $Pr\big(X \in [p - \varepsilon, p + \varepsilon]\big) \geq 1 - \delta$. If the running time of $\mathcal{A}$ is polynomial in $|x|$, $\frac{1}{\varepsilon}$ and $\log(\frac{1}{\delta})$, $\mathcal{A}$ is said to be fully polynomial.

The probability Pr is taken over the random choices of the algorithm. The approximation is ε-good with confidence $(1 - \delta)$ after a number of samples polynomial in $\frac{1}{\varepsilon}$ and $\log(\frac{1}{\delta})$. The main advantage is that, in order to design a path generator, we only need to simulate the behaviour of the system using a succinct representation of it (called the *diagram*).The generic approximation algorithm of APMC is the following.

```
**Generic approximation algorithm** $\mathcal{GAA}$
**Input:** $diagram, k, \psi, \epsilon, \delta$
**Output:** approximation of $Prob_k[\psi]$
$N := \ln(\frac{2}{\delta})/2\varepsilon^2$ ; $A := 0$
For $i = 1$ to $N$ do $A := A + $ **Random Path**$(diagram, k, \psi)$
Return $Y = A/N$
```

where the function **Random Path** is:

```
Random Path
Input:  $diagram, k, \psi$
Output:  samples a path $\pi$ of length $k$ and check formula $\psi$ on $\pi$
  (i) Generate a random path $\pi$ of length $k$ (with the diagram)
 (ii) If $\psi$ is true on $\pi$ then return 1 else 0
```

In [17], it is proven that $\mathcal{GAA}$ is a fully polynomial RAS for computing $p = Prob_k[\psi]$ whenever ψ is a bounded temporal property and $p \in]0,1[$. The randomised algorithm of APMC allows to approximate in polynomial time, and with very high confidence, satisfaction probabilities of temporal properties. The main advantage is to eliminate space complexity by using path generation and efficiently bounding sample size. Moreover this approach is highly parallelizable and APMC uses a distributed computation model to distribute path generation and formula verification on a cluster of workstations.

3.3 Architecture and Implementation

APMC includes two independent components: the compiler and the deployer. The APMC compiler takes the model description written with the PRISM language (a variant of Reactive Modules), and a list of temporal properties to check on this model. It produces an ad-hoc verifier for this set of properties over the given model. The output of the compiler is in fact a set of functions in ANSI C suitable for verifying the properties on the model. This file lacks a main function and an engine to produce the verification.

Providing the engine and the missing functions for the ad-hoc verifier is the goal of the deployer. It produces a stand-alone binary which takes only three parameters: the approximation parameter, the confidence and the paths length. It then runs the simulation and outputs the approximated probabilities for each of the temporal formulas. Thus, the deployer provides the working program suitable for a distributed verification inside a LAN. This distributed deployment strategy runs in parallel these components on all the participating nodes and provides the same result with a linear acceleration.

4 Sketch of the protocol

In [4], a probabilistic distributed algorithm for constructing a TDMA schedule is presented. For each sensor, the time is divided into frames (that need not be of same length at each node), that in turn are divised into slots (whose size is for simplicity considered as the same for each sensor. Essentially, the algorithm has two layers :

(i) the first layer, LooseMAC constructs a TDMA schedule where every sensor in a neighborhood is able to communicate with no conflicts; however, some bandwidth may be wasted in this process.

(ii) the second layer, TightMAC allocates the remaining slots so that the other

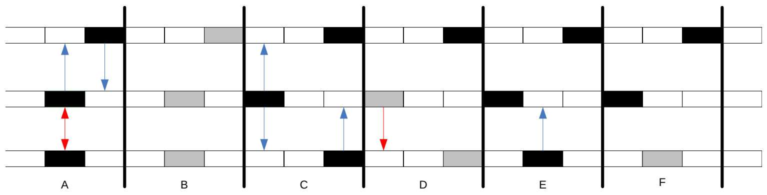


Fig. 1. Example execution of LooseMAC

wasted bandwidth is actually used.

In order to cope with sensor removal or arrival, those two layer are alternatively used.

The LooseMAC layer can informally be described as follows. Each node considers that frames are divised into slots of equal size. Then, until a node has not found a free slot, it randomly chooses a slot and emits a message in this slot. If no collision is detected (because a neighbor emited a message at the same time) or reported (because a neighbor received two messages at the same time, and is experiencing the well known hidden terminal effect), the time slot is selected by the node. The protocol is examplified in Figure 1. During Frame A, nodes randomly choose slots. The second and third nodes notice that a collision occur, while the first node found a collision free slot. In Frame C, the second and third nodes randomly choose a new slot. The second node knows that the first node chose the third slot (because it is a direct neighbor of this node), so it is able to choose a non conflicing slot. However, the third node chooses a slot that conflicts with the first one. The second node is then able to report a conflict. In Frame E, the third node is eventually able to pick a free slot, and the LooseMAC algorithm terminates for all nodes.

The TightMAC layer can informally be described as follows. Each node repeatedly uses the LooseMAC mechanism until all nodes in the neighborhood at distance 3 have successfully executed LooseMAC. Then, nodes compute how many nodes exists in their neighborhood at distance two, that is, the number ϕ of nodes they will compete with for obtaining TDMA slots. They then randomly choose slots in the remaining free slots, but no more than $1/\phi$ overall. If no collisions are detected or reported, the additional TDMA slots are gained.

Additional details about the two layers are provided in [4].

5 Modeling

In this section, we describe the modeling of the LooseMAC and TightMac protocols using the Reactive Module Language.

5.1 LooseMAC

Each sensor is modelled by an independant module. A sensor can be in one of three modes:

- $NEWSLOT$, if it has changed its randomly chosen time slot in the current frame,

- $WATCH$, after a whole frame passed since the node in the $NEWSLOT$ mode chose its time slot,
- $READY$, if no conflict has occurred in a whole frame and the node was in the $WATCH$ state.

When every node has reached the $READY$ state, the system is *stable*, i.e. the slots chosen are conflict-free. A proof of this protocol is given in [4].

The LooseMAC protocol has been modelled with particular attention paid to keep the original algorithm apparent. In this perspective, its three main subparts as described in the appendixes of [4] are clearly distinguished. For a node i, we have the following:

- Send(): broadcasts the state of the node i in its current time slot. The state sent consists of the identifier i of the node, its freshness flag, and if it has detected any conflict in the previous phase. This sending is made only if the node is in the $NEWSLOT$ mode or if it has detected a conflict.
- Receive(): at every time slot, checks for conflicts or arrival of new nodes. A conflict is detected if the node received noise, i.e. more than one message, if some message is received in i's time slot or if i already received a message in this time slot before. The node stores any correspondence seen between node identifiers and time slots.
- UpdateMode(): updates the mode of the node i in its current time slot, according to the mode switches presented before.

The behavior of the main function of this TDMA protocol is as follows:

```
LooseMAC

(i)  Initialize some internal values
(ii) (a) Call sequentially Send(), Receive() and UpdateMode(),
     (b) Increment a local time reference,
     (c) Loop to (a).
```

The protocol assumes that every node increments at *regular intervals* a local time reference, interval in which we are sure every possible action are made. To model this, we define a global time reference. All nodes use this time reference to wait for the completion of a time frame and increment their local time reference.

Reactive Modules is a reactive language, so the sequentiality of the operations has to be simulated. We use the usual transformation to implement the sequentiality of events through a *state* which describes the current progress of each module. This state describes the bold execution regarding the algorithms presented before, and each algorithms has its own sub states. The equivalence between the execution of LooseMAC and the internal state of a node i is as follows:

```
Internal State    LooseMAC Main Algorithm
-------------------------------------------------------------
NotIn       {  // Node is not in the network
Init        {  $Mode \leftarrow NEWSLOT$
            {  $\sigma \leftarrow$ random slot
               while true do
Sending     {     Send() ()
Receiving   {     Receive() ()
UpdateMode  {     UpdateMode() ()
Ended       {     TimeReference ← TimeReference + 1
            {     if TimeReference = FrameSize + 1 then
            {        TimeReference ← 1
                  end if
               end while
```

Fig. 2. Node's internal state and LooseMAC algorithm

The last state, namely Ended, indicates that the node has finished the loop iteration. In the original algorithm, it is the moment when the local time reference is incremented. In our modeling, it corresponds to a *synchronization* between all the processes, so that the global time reference is correctly updated.

As an example of a sub state in an algorithm, UpdateMode() uses a DoUpdatemode state in which the newly computed mode, which has been stored in $NewMode$, is affected to the actual mode of the node. This avoids clashes when the mode is tested within UpdateMode().

Since it is assumed that we know a maximum time interval for a communication to complete, they are made quasi-synchronously. When a node i sends a message to another one, i stores directly its data in the variables of its recipient and increments a counter of received message in it. Moreover, a reading is made if and only if all sending are made: this helps knowing if a node actually received *noise*, that is to say, more than one message, or a single message.

In theory, all nodes should execute the same algorithms: the code made so far is expected to be seen only once in the source files. However, the broadcasting part of Send() needs a node-dependent information, that is known at compile-time: its neighbors.

As a consequence, each node module is unique and has to be generated according to a graph file. XRM[4], a Reactive Modules preprocessor, has been used to simplify their writing.

[4] XRM: eXtended Reactive Modules, `http://www.lrde.epita.fr`.

5.2 TightMac

TightMac and LooseMAC execute themselves in parallel. As a result, the code for TightMac is an addendum of the previous one. It simply uses the slot found by LooseMAC to compute another slot on a smaller frame.

As a simplification of the process, the mathematical calculus are made at compile time. This is made possible thanks to the global knowledge of the graph at this moment.

6 Experiments

We performed an analysis of the model using APMC 2.0 and 3.0. All experiments were run using a set of heterogenous workstations ranging from a simple ATHLON 2 Ghz 512 MB to a bi-Xeon 3.2Ghz 4 GB. We set the approximation parameter $\varepsilon = 10^{-2}$ and the confidence parameter $\delta = 10^{-5}$. The first thing to do for the verification, was to set the communication network topology. We study two topologies : peer to peer communication over a generated dense graph, and peer to peer over a sparse graph. We give results for both topologies.

6.1 Properties to be checked

Here is the properties we were interested in. The three first properties were checked on LooseMAC, and the last ones on TightMac.

Experiment 1. Contention-free from the initial state.

The first experiment we adressed was to verify the correctness of the protocol. That is we wanted to check that eventually every nodes will become ready, meaning that each node is given a unique slot for communication, thus avoid conflicts. More precisely, our goal with this experiment is to check (experimentally) the validity of the lemma 1 of the extended version of [4].

Experiment 2. A fresh node breaks the stability momentary.

The algorithm LooseMAC is supposed to handle the case of a node joining or leaving dynamically the network. When a node leave the network, the algorithm is trivially robust. The problem is what happens when a node join the network: potentially there could be a conflict. LooseMAC is designed to be self-stabilizing by forcing some nodes to become non-ready. We verified this assumption by computing the probability that, when a new node join the network, some nodes become non-ready shortly (this is the lemma 2 in the extended version of [4]).

Experiment 3. Contention-freeness after a node joining the network.

The last experiment we did on LooseMAC was to check wether all node become ready after a new arrival, thus ensuring that the algorithm is self-stabilizing.

Experiment 4. Probability of conflict.

The lemma 5 of the extended version of [4] state that for a particular node, the probability of a conflict occuring in the neighborhood of this node within a number of time slots equal to the size of the frame is at most 1/2. We verified an abstract version of this property to ensure finally the correctness of the TightMac protocol. One might say that it is not interesting to verify this property since we can verify directly the correctness of the protocol. This is indeed true, but this verification show that we can validate the intermediate steps of a complex algorithm automatically.

Experiment 5. Stability of the network

Our last experiment is to compute the probability of a node to successfully choose a conflict-free time slot. And to check wether all nodes become conflict-free. This ensure that the network state is stable (corollary 6 of the extended version of [4]).

6.2 experimental results

Since the properties we checked on the protocol have been proven in [4], the experimental results are not surprising. However, they prove the effectiveness of our approach.

Results for LooseMAC

Figures 3 and 4 present the experimental results for experiments 1 to 3 over sparse and dense networks for two frame sizes (32 and 64 respectively).

The green line of experiment 2 shows that the probability that the system detects the apparition of a node is one. There is no executions where a new sensor enters the system and none of its neighbours do not acknowledge this by selecting the state NON-READY. This result holds for all topologies, and whatever the frame size.

This is a direct consequence of the protocol, since the new process will send a beacon message when entering the network. All the neighbour processes will note this either by noting that one of the slots that where not used before is now used, or that there is a collision on the slot randomly chosen by the new process.

The red curve of experiment 3 measures the probability that the system is converged (all nodes are in the READY state) according to the time. This curve exhibits a logarithmic convergence time, quickly reaching the probabiliy one after a warmup time. Comparing figures 3(a) and 3(b), or figure 4(a) and 4(b), one can see that the sparse network provides a better percentage of converging executions at a given time, thus demonstrating that the average convergence time is quicker for sparse graphs than dense graphs. This is due to the fact that the probability of conflict is lower with a small neighborhood.

Comparing figures 3(a) and 4(a), or figures 3(b) and 4(b), one can see that with a double frame size, the convergence time is slower. This is not obvious, since a double frame size induces a larger space to place the slots, thus a higher probability that a node chooses an empty slot. However, since the protocol works in phases and each node waits for the expiration of the frame before testing the collision status, a double frame size implies an overall slower convergence time. This illustrates a treadoff on this frame size, which should be short enough to provide a good response time and frequent testing of collision, but large enough to ensure a good probability of finding an empty slot.

The last experiment illustrated in figures 3 and 4 is experiment 3, represented by the blue curve. It measures the probability that the system converges within a given time unit. The time unit presented here has its origin at the insertion of the new node, after the system has converged without it. One can see that the convergence time exhibits the same behavior as from an initial configuration where none of the nodes are in the READY state, but with a longer warmup phase. The warmup phase is longer, since the insertion of the new node can imply a modification of the slots of all the components of the system, and the failure detection has first to propagate to all the nodes. Moreover, one can see that the warmup time depends on the frame size, thus emphasizing the impact of the failure detection, which happens only once in every frame.

Results for TightMac

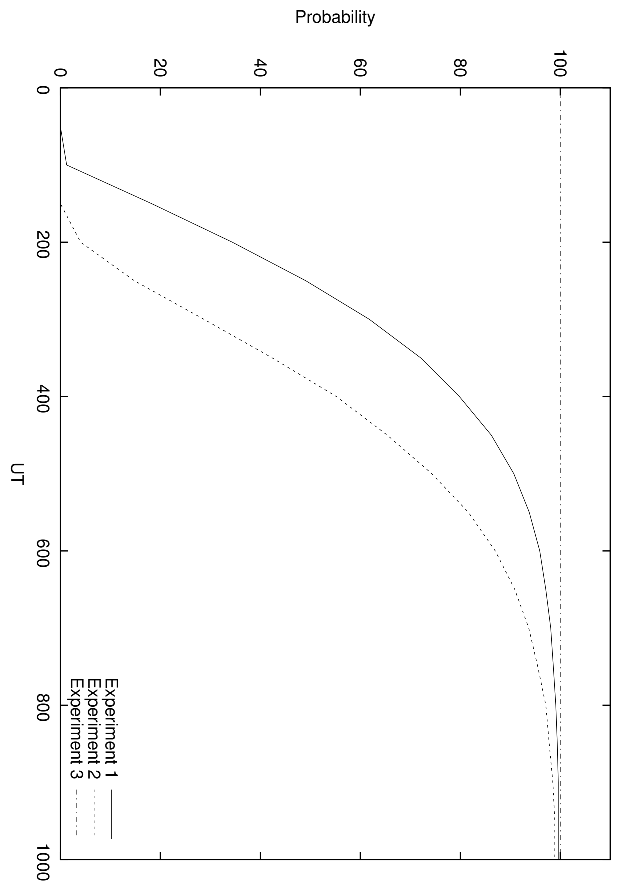
Figure 5 presents the result of experiment 5 over a sparse network with a LooseMAC framesize of 32, as in figures 3(b). One can see that the convergence time of the TightMac algorithm follows the same behavior as the LooseMAC algorithm it uses. The TightMac algorithm can converge only when the LooseMAC algorithm is also converged, and this explains why it follows the same tendency.

At the time we write this article, the experiment 4 is still running and we cannot include the results in this version. Additional figures, presenting the results obtained with experiment 4 will be included in the final version.

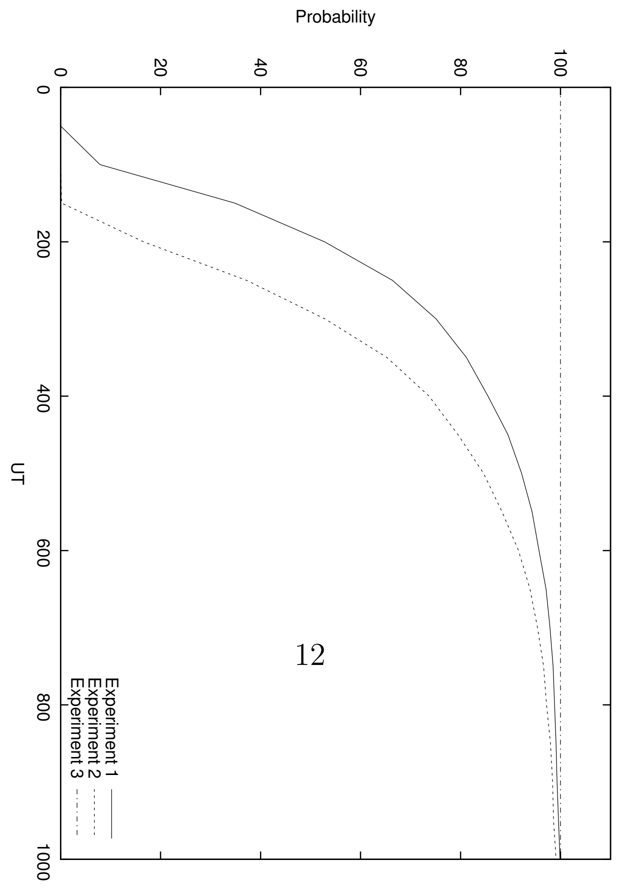
7 Conclusion

In this paper, we presented an analysis, using approximate probabilistic model checking, of the contention-free MAC protocol of [4]. We showed that this kind of method allows to efficiently verify/analyze the correctness and performance of complex distributed algorithms over sensor networks. Using these methods we can free ourselves from the state space explosion phenomenon arising with classical model checking techniques. However the numerical results we give are accurate only with respect to an approximation parameter (ε here) and in order to modelise efficiently the protocol we were required to add some constraints (such as a global timer) to make the model tractable for APMC. We also used extensively additional tools in order to generate the model, since sensor networks are very demanding objects to modelize.

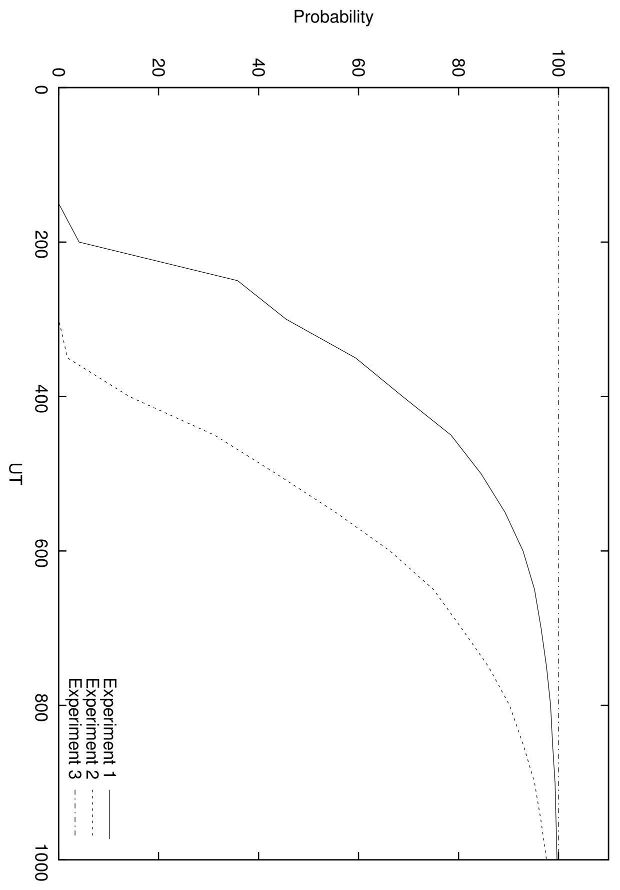
Acknowledgments Many thanks goes to Benoit Sigoure for providing us pre-processing tools for Reactives Modules.



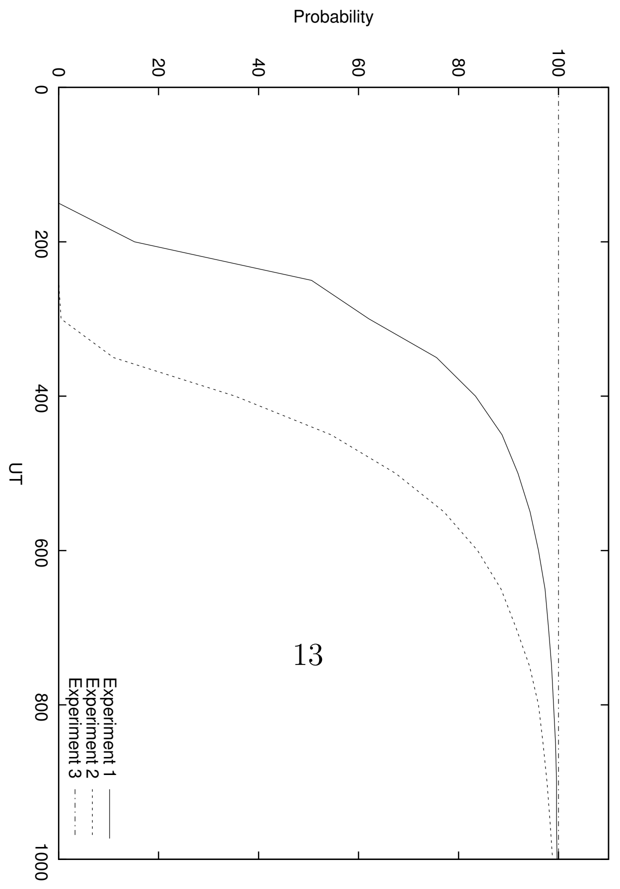
(a) prob. vs Time Units - dense graph - frame size = 32



(b) prob. vs Time Units - sparse graph - frame size = 32



(a) prob. vs Time Units - dense graph - frame size = 64



(b) prob. vs Time Units - sparse graph - frame size = 64

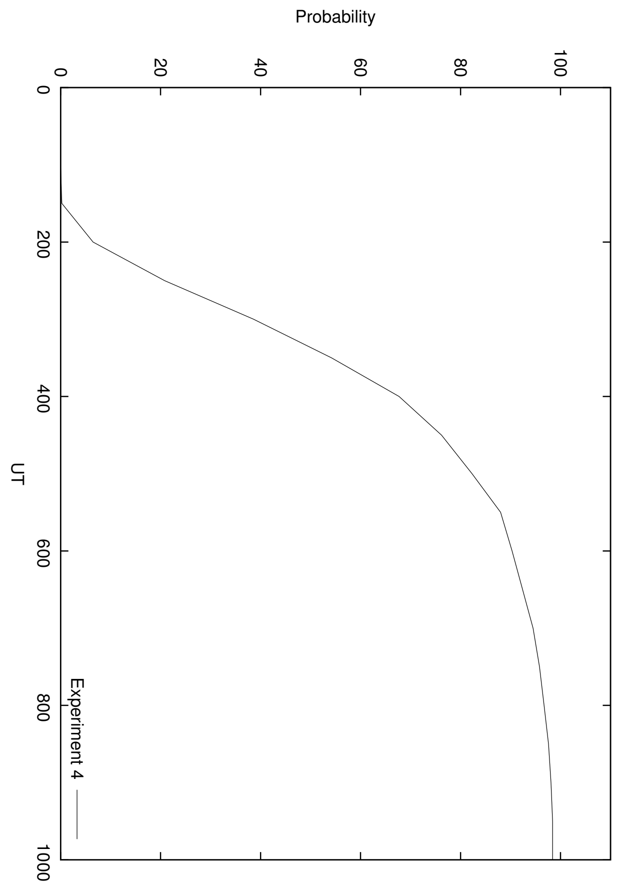


Fig. 5. Experimental results - TightMac

References

[1] LAN MAN Standards Committee of the IEEE Computer Society, Wireless LAN medium access control (MAC) and physical layer (PHY) specification, IEEE, New York, NY, USA, IEEE Std 802.11-1999 edition, 1999.

[2] Norman Abramson. Development of the ALOHANET. IEEE Transactions on Information Theory, vol. 31, no. 2, pp. 119-123, Mar. 1985.

[3] R. Alur and T. Henzinger. Reactive modules. in Proc. of *LICS 96*, 1996.

[4] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. Contention-free MAC protocols for wireless sensor networks. In *Proceedings of 18th International Conference of Distributed Computing (DISC)*, Amsterdam, The Netherlands, October 4-7, 2004. Springer Lecture Notes in Computer Science 3274, pp. 245-259.

[5] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. In *Proceedings of Embedded Software, First International Workshop EMSOFT 2001*, Springer LNCS 2211, pp. 114-130, 2001.

[6] A. Demaille, T. Herault and S. Peyronnet. Probabilistic verification of sensor networks. In *Proc. of the RIVF 2006 conference.*

[7] M. Duflot, L. Fribourg, T. Herault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet and C. Picaronny. Probabilistic Model Checking of the CSMA/CD Protocol Using PRISM and APMC. In the proceedings of avocs 2004.

[8] R. Grosu and S. A. Smolka, "Monte carlo model checking." in *TACAS*, ser. Lecture Notes in Computer Science, N. Halbwachs and L. D. Zuck, Eds., vol. 3440. Springer, 2005, pp. 271–286.

[9] M. Haenggi and X. Liu. Fundamental throughput limits in Rayleigh fading sensor networks. *In submission*, 2003.

[10] T. Herault, R. Lassaigne, F. Magniette and S. Peyronnet. Approximate Probabilistic Model Checking. In *Proceedings of Fifth International VMCAI'04*, pp 73-84, LNCS:2937, January 2004.

[11] Ted Herman and Sbastien Tixeuil. A distributed tdma slot assignment algorithm for wireless sensor networks. In Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004), number 3121 in Lecture Notes in Computer Science, pages 45-58, Turku, Finland, July 2004. Springer-Verlag.

[12] S. S. Kulkarni and M(U). Arumugam SS-TDMA: A self-stabilizing MAC for sensor networks Sensor Network Operations, IEEE Press, 2005

[13] Leonard Kleinrock and Fouad Tobagi, Packet switching in radio channels: Part I - carrier sense multiple access modes and their throughput delay characteristics,IEEE Transactions on Communications, vol. 23, no. 12, pp. 1400-1416, Dec. 1975.

[14] M. Kwiatkowska, G. Norman and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02), pp 200-204, LNCS:2324, 2002.

[15] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. Proc. of the 2nd Int. Worshop PAPM-PROBMIV 2002, LNCS no 2399, pp. 169-187, july 2002.

[16] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol. *Formal Aspects of Computing* 14:3, pp. 295-318, 2003.

[17] R. Lassaigne and S. Peyronnet. Probabilistic verification and approximation. In Proc. of Wollic 05. Electr. Notes Theor. Comput. Sci. 143: 101-114 (2006).

[18] Nathalie Mitton, Eric Fleury, Isabelle Gurin-Lassous, and Sbastien Tixeuil. Self-stabilization in self-organized wireless multihop networks. In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (WWAN'05), pages 909-915, Columbus, Ohio, USA, June 2005. IEEE Press.

[19] D. Monniaux, "An abstract monte-carlo method for the analysis of probabilistic programs." in *POPL*, 2001, pp. 93–101.

[20] T. S. Rappaport. Wireless Communications. Principles and Practice, Prentice Hall, 1996.

[21] H. Takagi and L. Kleinrock. Throughput analysis for persistent CSMA system. IEEE Transactions on Communications, vol.33, no 7, p. 627-638, july 1985.

[22] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. in *CAV*, ser. Lecture Notes in Computer Science, R. Alur and D. Peled, Eds., vol. 3114. Springer, 2004, pp. 202–215.

[23] J. Wang and S. Keshav. Efficient and accurate Ethernet simulation. Proc. of the IEEE 24th Conference on Local Computer Networks, p. 182-191, oct. 1999.

[24] Wei Ye and John Heidemann. Medium Access Control in Wireless Sensor Networks. Wireless Sensor Networks, Taieb Znati, Krishna M. Sivalingam and Cauligi Raghavendra (eds.), Kluwer Academic Publishers, May 2004.

[25] H. L. S. Younes and R. G. Simmons. ‘Probabilistic verification of discrete event systems using acceptance sampling. in *CAV*, ser. Lecture Notes in Computer Science, E. Brinksma and K. G. Larsen, Eds., vol. 2404. Springer, 2002, pp. 223–235.

[26] F. Zhao and L. Guibas (Editors). *Proceedings of Information Processing in Sensor Networks, Second International Workshop, IPSN 2003*, Springer LNCS 2634. April, 2003.