Probabilistic Verification of Sensor Networks

Akim Demaille
LRDE/EPITA
Akim.Demaille@lrde.epita.fr

Thomas Hérault
LRI - U. Paris XI
herault@lri.fr

Sylvain Peyronnet
LRDE/EPITA
syp@lrde.epita.fr

***Abstract*— Sensor networks are networks consisting of miniature and low-cost systems with limited computation power and energy. Thanks to the low cost of the devices, one can spread a huge number of sensors into a given area to monitor, for example, physical changes of the environment. Typical applications are in defense, environment, and design of ad-hoc networks areas.**

In this paper, we address the problem of verifying the correctness of such networks through a case study. We model a simple sensor network whose aim is to detect an event in a bounded area (such as a fire in a forest). The behavior of the network is probabilistic, so we use Approximate Probabilistic Model Checker (APMC), a tool that allows to approximately check the correctness of extremely large probabilistic systems, to verify it.

I. Introduction

The recent advances in the field of micro-electronics led to the design of small low-cost and low-energy devices. These devices have the ability to communicate on a short distance via wireless technology. Basically these *sensor nodes* can sense various physical quantities using internal detectors, process data and communicate together using broadcast mechanism.

A *sensor network* [1], [2] is a network composed of a huge number of sensor nodes. The sensor nodes are densely but randomly spread in some place with the goal of cooperatively map the entire place. To achieve this task, the sensor nodes runs ad-hoc network communication algorithms in order to communicate despite the unpredictability of the network topology. They also run energy saving processes and low-energy computation tasks (in order to pre-process the data and avoiding extra communications).

Typical applications of sensor networks arose in the fields of health and environment monitoring or military and security tracking. For example, they can be used in order to detect fires in a forest, or to detect physical intrusion of highly critical places (nuclear power plant, bank...).

For computer scientists the challenge is to design algorithms to allow sensors to communicate efficiently together, and to achieve computation even if the computational power and the energy are low. With the design of efficient but complex algorithms, design or implementation flaws may appear and ensuring the correctness and safety properties of the sensor network becomes the main issue. Many methods are dedicated to the verification of distributed systems, but for the verification of sensor networks, two characteristics can make the verification somehow difficult:

- the behavior of the sensor network is probabilistic: the nodes are randomly disseminated and they mostly achieve a probabilistic behavior (generally in the memory saving process and in the collision avoidance mechanism for the wireless communications);
- the number of nodes in the network is extremely large, making the thorough verification intractable in most cases.

Model checking [3] and testing [4] are two methods that can be used for the verification of such networks. The idea of *model checking* is to construct a model that corresponds to the behavior of the system and to write the specifications using a formula of a temporal logic. Model checking will allow to verify automatically if the formula is true into the model (i.e. the specification holds). Model checking algorithms are generally efficient: the complexity is at most polynomial in the size of the model and exponential in the size of the (small) specification. The problem here is that the model is so large that even with this complexity the computation is intractable, this is the so-called state space explosion phenomenon. *Testing* is another approach that can be used. Using testing techniques means that the system is run on a well selected set of configurations to check if it achieves a good behavior. However, this approach is not fully automatic and selecting the set of tests is a difficult task, in particular to ensure that the system exhibits all possible types of behavior.

In the last year a new model checking technique emerged: Approximate Probabilistic Model Checking. Using this technique we can approximately compute the probability that a model verify a specification [5], [6]. 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 the paper are:

- the modeling of a sensor network whose goal is to detect an event on a grid of arbitrary size and to send a signal to some specific nodes (an abstract version of a sensor network whose goal is the detection of, for example, a fire in a forest);
- various experimental results on this model, thus showing the interest of using approximate probabilistic model checking for the verification of sensor networks.

The structure of the paper is as follows. In Section II, we review the related work on sensor networks and probabilistic verification and in Section III we explain briefly the approximate model checking method and present APMC, a

tool that implements the method. Then, in Section IV, we give a detailed sketch of the kind of sensor network we consider, together with its modeling. This section pursues with experiments on the model using APMC. Finally the results are discussed in Section V.

II. Related Work

Sensor networks are used for military, environmental, or commercial applications such as intrusion detection [7], disaster monitoring [2] and habitat monitoring [8].

The study of sensor networks is a dynamic field of research. Researches are conducted in various directions. According to survey papers [1], [2], sensor networks differ from classical ad hoc networks by many characteristics such as the density of the network, the broadcast mechanism, the limited power of computation, the energy consumption etc.

The two biggest issues in the field are currently the practical and theoretical design of sensor nodes and the design of distributed algorithms for sensor networks. It means that almost everything has to be done: routing algorithms, experiments, simulation, energy saving algorithms etc.

In [7], some real field experiments deploying over 90 sensors node are conducted to evaluate the performances of disaster monitoring protocols. The proposed protocols are able to provide intrusion detection and tracking, based on Logical Grid Routing Protocols. However, due to the high complexity of the sensors (they provide heat sensing for man intrusion as well as magnetic sensing for car intrusion), the high cost of the physical equipment prevented high scale measures.

Routing protocols were proposed to build virtual coordinates without location information and route messages based on virtual coordinates [9], [10]. However, routing induces a single known destination, which is usually stationary. In this work, we address the problem of propagating some information to a mobile sink, knowing a likely path of the sink.

An other issue is the simulation/verification of the behavior of large sensor networks, in order to save money and time during the design process. While lots of work in the field use simulation with tools like Ptolemy [11], J-sim [12], Avrora [13], and others [14], [15], verification seems to be still poorly explored — except the approach of [16]. This is somehow strange since the field of ad hoc networks is widely explored by researchers from the verification community. In this paper, we investigate this problem using an approximation-based verification method.

The research in the field of methods for approximating probabilistic model checking is quite young and there is only a few other approaches than the APMC one. In [17], a procedure is described for verifying properties of discrete event systems based on Monte-Carlo simulation and statistical hypothesis testing. In [18], a statistical method is proposed for model checking of black-box probabilistic systems against specifications given in a sub-logic of Continuous Stochastic Logic (CSL). These approaches differ strongly from ours by using statistical hypothesis testing instead of randomized approximation schemes. More recently, in [19], a randomized algorithm for probabilistic model checking of safety properties expressed as Linear Time Logic (LTL) formulas was given. This approximation method uses the optimal approximation algorithm of [20]. In another approach [21] both random testing and abstract interpretation are used for the verification of C programs. The main advantage of this method is that it applies to systems that contain non-deterministic choices (we discuss about non-determinism in Section V).

III. APMC — Theory and Implementation

A. Theoretical foundations

In this paper, our aim is to compute the probabilities that some properties hold in the behavior of a sensor network. Basically, the properties are temporal in the sense that we described the behavior of the system during a fixed period of time. Moreover, the system is probabilistic since each sensor has a probability of achieving one of its possible behavior. For these reasons, the framework of approximate probabilistic model checking is convenient for the modeling and verification of sensor networks.

Indeed, the method of [6], described below, has the goal of approximately compute satisfaction probabilities of Linear Time Logic (LTL) properties over fully Probabilistic Transition System (PTS) (or discrete-time Markov chains (DTMCs)). LTL formulas are built over a set of atomic propositions using temporal operators. The syntax and semantics of LTL are not really of interest for this paper, suffice it to know that using this logic, we can describe the temporal behavior of a system such as a program or a protocol.

Let us now define the kind of system we study:

Definition 3.1: A DTMC is a tuple $\mathcal{M} = (S, \overline{s}, P)$ where S is a set of states, $\overline{s}$ is the initial state, and P is a transition probability function.

It means that our goal is to ensure properties of a system using sequences of states of the system. For us the DTMC will be the modeling corresponding to the sensor network. In our framework, a state will be a vector of the actual values of all the variables of the sensor network. A path will be an execution of the system, that is the successive values of all variables.

We denote by $Path(s)$ the set of paths whose first state is s. The length of a path π is the number of states in the path and is denoted by $|\pi|$, this length can be infinite but we will only consider here bounded length paths. The probability measure $Prob$ over the set $Path(s)$ is defined in a classical way. We denote by $Prob[\phi]$ the measure of the set of paths $\{\pi \mid \pi(0) = s \text{ and } \mathcal{M}, \pi \models \phi\}$ (see [22]). Let $Path_k(s)$ be the set of all paths of length $k > 0$ starting at s in a PTS. The probability of an LTL formula ϕ on $Path_k(s)$ is the measure of paths satisfying ϕ in $Path_k(s)$ and is denoted by $Prob_k[\phi]$.

In order to estimate the probabilities of bounded properties with a simple randomized algorithm, we generate random paths in the probabilistic space underlying the DTMC structure of depth k and compute a random variable A/N which estimates $Prob_k[\psi]$. Our approximation is 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 behavior of the system and store the values of all variable at each computation step.

Our approximation problem is defined by giving as input x a program, a formula and a positive integer k. The model is used to generate a set of execution paths of length k. A randomized approximation scheme is a randomized algorithm which computes with high confidence a good approximation of the probability measure $\mu(x)$ of the set of execution paths satisfying the formula ϕ.

Definition 3.2: A fully polynomial randomized approximation scheme for a probability problem is a randomized algorithm $\mathcal{A}$ that takes an input x, two real numbers $0 < \varepsilon, \delta < 1$ and produces a value $A(x, \varepsilon, \delta)$ such that:

$$Prob\big[|A(x,\varepsilon,\delta) - \mu(x)| \leq \varepsilon\big] \;\geq\; 1-\delta.$$

The running time of $\mathcal{A}$ is polynomial in $|x|$, $\frac{1}{\varepsilon}$ and $\log \frac{1}{\delta}$.

The probability is taken over the random choices of the algorithm. We call ε the *approximation parameter* and δ the *confidence parameter*. The approximation algorithm we use consists in generating $O(\frac{1}{\varepsilon^2}.\log \frac{1}{\delta})$ execution paths, verifying the formula ϕ on each path and computing the fraction of satisfying paths, that is an ε-approximation of $Prob_k[\phi]$.

Theorem 3.3: This approximation algorithm is a fully randomized approximation scheme for the probability $p = Prob_k[\phi]$ of an LTL formula ϕ if $p \in]0,1[$.

This result is obtained by using Chernoff-Hoeffding bounds on the tail of the distribution of a sum of independent random variables. The time complexity of the algorithm is polynomial in $\log(1/\delta)$ and $1/\varepsilon$. The space complexity is linear in the length of execution paths.

This method was recently extended to the verification of real C programs [23], using encapsulation of programs directly into a specific verifier (to generate executions and verify formulas).

B. APMC tool

The design of Approximate Probabilistic Model Checker (APMC) started in 2003 [6]. The tool implements the approximation method, with additive error, described in Section III-A. It is freely available [24] under GNU General Public License (GNU GPL) and is under permanent development. APMC is now a probabilistic distributed model checker that uses a client/server computation model in order to speed up the verification process by distributing the path generation and verification on a cluster of workstations (extensive tests with hundreds of machines were done). The tool is easy to use and features a graphical user interface to enter the model, formula and the approximation parameters.

Since 2003, numerous experiments were done, such as the verification of various probabilistic distributed algorithms (mutual exclusion, dining philosophers, leader election...) and of the IEEE 802.3 CSMA/CD protocol (part of the Ethernet protocol) [25]. We also released the core computation engine of APMC into a self-sufficient library, which is now fully integrated into the state-of-the-art probabilistic model checker PRISM [26].

Let us now describe more precisely the architecture of APMC, which is twofold. The first component, the APMC Compiler, produces an ad-hoc verifier including a sample generator and a checker for a given model (described in Reactive Modules (RM) [27]) and a given property (LTL). The second module, the APMC Deployer, takes this verifier and the set of available computing resources, deploys the verifier on this set of computers and collects the result, which is the approximated value of satisfaction probability of the formula on the model.

The technique used to approximate this value assumes the verification of the formula on a large set of independent samples of bounded length. We use the independence property of the samples to distribute the generation and verification of samples [28].

The deployment is performed following a spanning tree of bounded arity. Each node of the tree runs on a single computing resource, and spawns children up to the bound on other available resources. While its parent still accepts results from it, and until the number of collected samples is greater than the requested number if it is the root, it generates a sample and verifies the property on it. At each verification, counters of false and true samples are updated. Regularly (at fixed intervals), each node sends its counters of false and true samples to its parent, and resets them (except for the root, which awaits the end of the computation to produce these numbers). When a node receives these counters from one of its children, it aggregates these numbers as if it produced the verification.

Using the distributed version of the approximation scheme of APMC, we were able to verify systems so large that they are intractable for classical model checking methods [28]. Since a sensor network consists in a very large network with probabilistic behavior, APMC seems to be well adapted to verify it.

C. Our methodology

As already stated, our goal in this paper is to compute the probabilities of some given properties of sensor networks. In order to achieve this task, we will use APMC, thus obtaining approximate values of these probabilities. One can note that using a classical model checker, these values are not computable since the systems we consider are very large.

Our methodology is the following:

1) Write a model of the system we consider using the RM language. Here we use some additional tools to ease the modeling phase.
2) Write the properties we want to check on the model.
3) Launch APMC with the model and the specification as inputs, together with the verification parameters (approximation and confidence parameters, number of paths to generate).

As output, APMC gives the probability of each property to be true in the model, that is, in the system.

IV. Modeling & Experiments

A. Case Study: intrusion alert

In this paper we study the intrusion alert case. The goal of the sensor network is to detect an event, in a bounded area, and then to forward the information about it towards distinguished sensor nodes. In our opinion, such a system can for example be in real life a sensor network to detect the emergence of a fire in a forest: the sensors detect the fire and broadcast the information to some fixed stations. Another application can be the detection of hostile troops in a battle field.

To complete such a task, each sensor can *sense* its environment (to detect the event), can *listen* and *broadcast* information from and to other sensors and can save energy by *sleep*ing for a limited time.

We propose a simple probabilistic algorithm to achieve this goal. Sensors may be sleeping or active. When sleeping, a sensor will be waken up by an alarm, and the energy consumption is at the lowest level. When woken up a sensor flips a coin. With equiprobability, it tries to sense its environment to detect the potential event or listen for any message to propagate for a fixed amount of time.

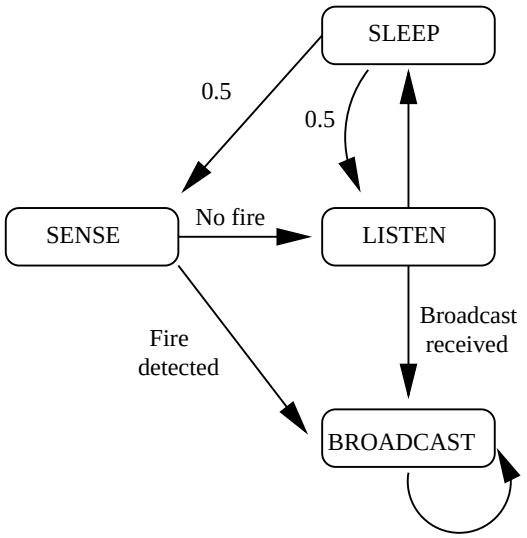
If it receives a broadcast alert message, the sensor will forever try to propagate the alert and will broadcast the alert message up to its battery exhaustion. If it did not receive an alert message, the sensor programs a new timer alarm and goes to sleep.

When sensing, if no event is detected, the sensor begins to listen for a broadcast message and executes the algorithm previously described. If an event is detected, it begins to broadcast the alert up to battery exhaustion.

This algorithm is simple, and could certainly be improved by routing techniques. However, we believe this kind of behavior is characteristic of sensor networks, which alternate between two states (normal one, when the sensors sense and communicate sparingly to lower their energy consumption, then alert state, when urgent communication at the cost of high energy consumption is needed). Moreover, although the algorithm is simple and each sensor has a small number of states, verifying and evaluating the probability of non trivial properties, like the probability that an event detection occurs, are hard to verify and to derive from the algorithm. It is thus a good case study for the approximate techniques that we propose in this article.

B. Model and modeling

We model the topology of the sensor networks as a regular square grid. Each sensor node can detect the event if it happens in its cell, and can broadcast information to his four neighbors (see Figure 1). This is a simple solution to define the neighboring of each sensor, and provide a realistic approach to wireless connectivity. Any sensor in any cell could be down, thus this grid topology is only used by our simulation mechanism to emulate a realistic geographic distribution. Many works on sensor networks makes the same assumption, and these logical grids may be built using distributed algorithm in [29], [30], or physically realized using a GPS and an election algorithm to ensure that at most one sensor is present in each logical square of the grid.



First the sensor is sensing its environment (state `sense`), if it detects a fire, it goes directly to the state `broadcast` and stays in it forever (that is it sends without interruption a signal to its neighbors), otherwise it goes to state `listen`, where the sensor is listening its neighborhood. If it catches a broadcast, it goes to `broadcast` in order to forward the information, otherwise it sleeps (state `sleep`) for a while: with equiprobability it goes back to `sense` or `listen`.

Fig. 1. Sensor node behavior

Instead of modeling directly the sensor nodes, we model the cell of the regular grid. Each cell can be active (there is an alive sensor node) or inactive (either the sensor node is running out of energy or is not present in the cell). Note that it is semantically equivalent to model the sensor or the cell, but it is easier to model cells instead of sensors. Each of the sensor node has a simple behavior depicted and detailed in Figure 1 and implemented in Figure 2. To avoid the clutter, this description does not take power consumption into account, but see Figure 3 for a more faithful description of the sensors.

Real field experiments have demonstrated that wireless broadcast communication from small and low-cost units are usually unreliable. Often, messages fail to reach direct neighbors, and may communicate with more remote components of the field. This unreliable communication media is modeled here in the listen state. Since each sensor independently may not be in listen state when a neighbor (near or far) broadcasts, a message may be lost. The probability law of the listen states encompasses the algorithm's behavior and the medium losses.

```
module sensor

  state : [0..4]  init SENSE;

  [] state = SENSE     -> state = senses ? BROADCAST : LISTEN;
  [] state = LISTEN    -> state = receives ? BROADCAST : SLEEP;
  [] state = SLEEP     -> 0.5 : state = SENSE + 0.5 : state = LISTEN;
  [] state = BROADCAST -> state = BROADCAST;

endmodule
```

BROADCAST, LISTEN... are constants defining the four possible states of a sensor. *senses* and *senses* are sensor-specific formulas that check whether the event happens at the location of this sensor, and whether one of the neighbors broadcasts.

Fig. 2. Reactive Module code for a sensor as described in Figure 1

C. Implementation

Our model is written in Reactive Modules (RM), a language suitable for the description of the behavior of concurrent systems. It provides domain-specific primitives to describe the probabilistic behavior of communicating modules. The implementation of a simple sensor is therefore straightforward, see Figure 2. Unfortunately, in particular because of the lack of iteration features, describing a system composed of many modules is inconvenient. To overcome this limitation, we used a preprocessing phase, using M4. M4 is a lexical macro processor, in the spirit of C preprocessor (`cpp`), but much more expressive. The definition of the grid reads as follows, where `rm_sensor(X, Y)` is a macro defining a sensor at coordinates `X`, `Y`:

```
m4_for([X], [0], MAX_X, [1],
       [m4_for([Y], [0], MAX_Y, [1],
               [rm_sensor(X, Y)])])
```

M4 also enables to optimize the description of the modules according to the parameters of the experiment. For instance, when power consumption is taken into account, the resulting module description is more complex to model the batteries, see Figure 3. An infinite power model could be achieved by setting the initial battery level to a very large number, but not only does this incurs additional computations, it also keeps twice as many variables as needed — a severe penalty.

Because using M4 invites to factor the code as much as possible, it is easy to extend the model with very limited changes: adding a global counter incremented by each action only requires updating the battery-consumption macro.

The use of M4 is also positive to write the LTL specifications to check; it allows to:

- share a common set of formulas with the RM description of the system (to avoid by-hand rewriting);
- have the same variation points (with or without batteries, etc.);
- overcome the same limitations: lack of iteration constructs and of genericity.

The specifications check the arrival time of the signal on the peripheral of the area, thus modeling the fact that the authority (e.g. firemen) is aware of the fire.

The macro `rm_foreach_boundary` is syntactic sugar for a series of `m4_for` covering the boundaries of the grid, and `LENGTH` is parameter k in Section III-A, the maximal length of the paths explored — the highest possible value of the timer.

```
// Test the arrival of the signal at
// 0, 100, 200, ... LENGTH.
m4_for([T], [0], LENGTH, [100],
[true U (t <= T
         & (0 rm_foreach_boundary([X], [Y],
          [ | rm_state(X, Y) = BROADCAST])))
])
```

D. Experiments

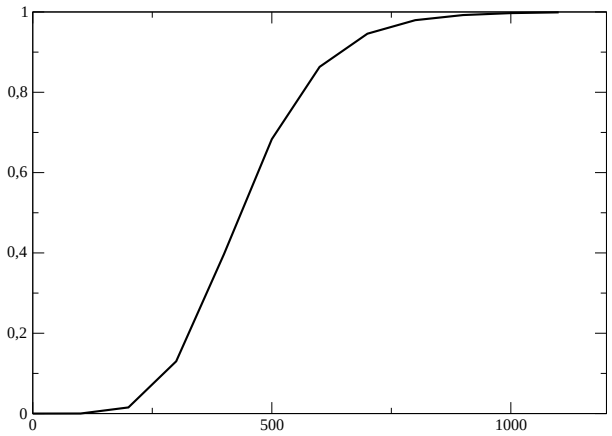
Using our parameterizable model, we did various experiments. Since APMC is a distributed model checker, we use several workstations to complete the computations. Notably, we use Athlon XP2800+ workstations with 512 Mb of RAM, Athlon XP 1000+ machines and several Pentium IV 3.2 GHz with 1 Gb of RAM. We set the approximation parameter $\varepsilon = 10^{-2}$ and the confidence parameter $\delta = 10^{-10}$, thus generating and verifying at least 940,000 paths for each formula, according to the Chernoff-Hoeffding bound. All the workstations are running under GNU/Linux.

We did several experiments in order to sort out the behavior of the sensor network depending on the values of the parameters. At the beginning, a single event is set — at the center of the grid — and all the sensors are in the `SENSE` state; in other words, the initial time is when the first opportunity is given to network to discover the event. The various experiments are described below. Most of them use the same family of formulas: "probability of detecting the event at a given moment". These formulas measure the probability that a sensor on the peripheral of the area enters the `BROADCAST` state, meaning that the event was signaled outside the grid. Formally, this formula is given by the previous RM expression generated by the M4 macro, for each model (see Section IV-C).

```
module sensor

  s : [0..4]  init SENSE;  // The state.
  b : [0..15] init 15;     // The battery level.

  [] s = SENSE     ->       b'= b - 2 & s'= 0 < b ? senses   ? BROADCAST : LISTEN : OFF;
  [] s = LISTEN    ->       b'= b - 3 & s'= 0 < b ? receives ? BROADCAST : SLEEP  : OFF;
  [] s = SLEEP     -> 0.5 : b'= b - 1 & s'= 0 < b ? SENSE                         : OFF
                     + 0.5 : b'= b - 1 & s'= 0 < b ? LISTEN                        : OFF;
  [] s = BROADCAST ->       b'= b - 3 & s'= 0 < b ? BROADCAST                     : OFF;

endmodule
```

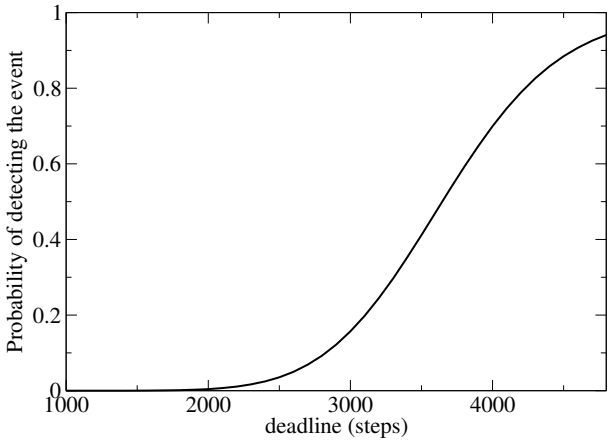
When batteries are taken into account (see Figure 2), each state incurs a cost (`SLEEP` : 1, `SENSE` : 2, `LISTEN` & `BROADCAST` : 3), and when no power remains, the sensor is put in an `OFF` dead state.

Fig. 3. Sensor with power consumption



Probability of detecting the event vs path length.

Fig. 4. 10x10 grid, infinite energy

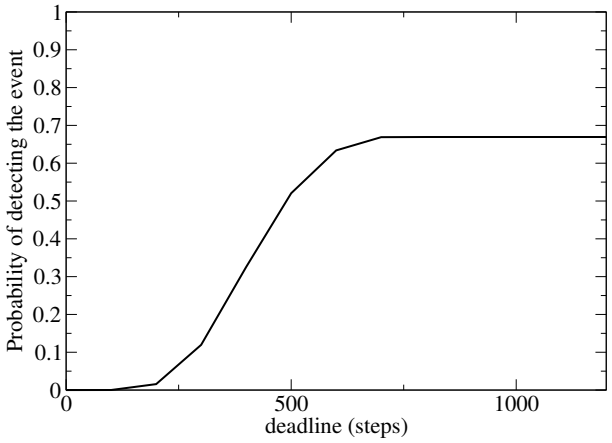


Probability of detecting the event vs path length.

Fig. 5. 20x20 grid, infinite energy

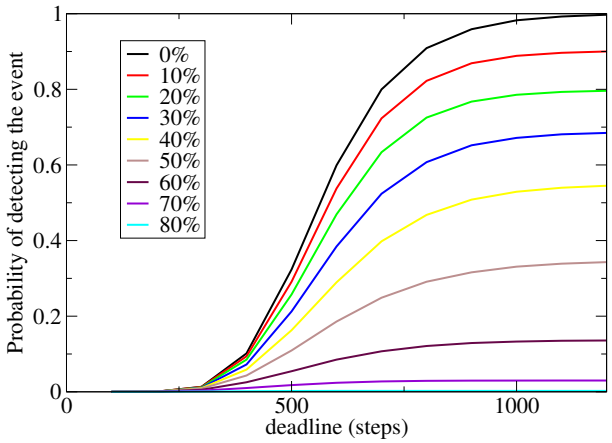
1) Estimating the minimal path length for detection: The goal of the first set of experiments is twofold: estimating the value of the k parameter (length of generated paths) that ensures the detection of the event by the stations on the peripheral of the area, and setting a reference for following experiments (Figure 4 for a 10x10 grid, and Figure 5 for 20x20). Because it is a "limit" case, the power consumption is not taken into account (Figure 2). The length parameter must be big enough to observe the delivery of the signal with a high probability if it will happen in the complete (infinite) run, and small enough to spare computer cycles. The graph obtained is also used to emphasize the role of the batteries in the following sections.

2) Impact of limited energy: Our intuition is that if the sensors have only a limited amount of energy available, it will be difficult for the sensor network to ensure the good detection of the event. The second set of experiments evaluates the impact of the limited energy of the sensors (Figure 3). The delivery of the signal can no longer be guaranteed (Figure 6) for a limited energy (15 in our experiment).



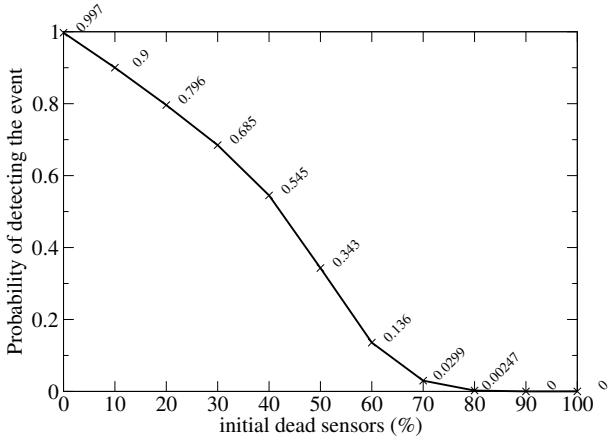
Probability of detecting the event vs path length.

Fig. 6. 10x10 grid, energy 15



Probability of detecting the event vs path length. Varying percentages of sensors are "lost" from the beginning.

Fig. 7. 10x10 grid, with initial loss and infinite energy



Probability of detecting the event vs initial losses on a 10*10 grid.

Fig. 8. Robustness to initial loss - path length 1200 - infinite energy

3) Impact of loss: The third round of experiments aims at estimating the impact of the initial loss of modules. See the Figure 7 to observe how delivery degrades; Figure 8 summarizes the impact of the initial loss onto the delivery of the signal for a fixed path length of 1200 (this length guarantees the probability to be one in the ideal case of the experiment 1). We did these experiments to know what happens if the density of sensors is too low in the field, or if the sensors are not regularly spread in the area.

4) Larger scale: Another interesting experiment is to compare the probability that the event is detected according the amount of initial energy to the same probability according the density of lost processes. However, to have a meaningful evaluation, this requires much larger deployments than the other experiments (with a larger grid of nodes). We conducted a set of experiments on square grids of 2500 sensors to make this comparison.

It appears that these experiments take too long to compute, which precluded the inclusion of the experiment in this work ($\approx$ 10 hours of computation on 200 bi-Opterons 2GHz, with 2GB of RAM, to evaluate the probability on 10,000 paths of length 100,000). This is due to the current design of the verification system: it assumes that the path is held in memory to verify the temporal formula on it. With 2500 sensors, the model has a large memory footprint, and the verification process is then limited by the memory bus speed, which is much smaller than the CPU speed.

The general method of APMC does not require that much memory. Indeed, other verification techniques, based on the automaton of the formulas to verify are now being designed and will remove this memory limitation of APMC. This limitation on the size of the system was unexpected and is due to the utilization of the meta-programming tools to build a system with a large reactive-module representation.

V. Discussion

A. On our experiments

In this section, we comment the results reported in Section IV-D.

1) Estimating the minimal path length for detection: As reported in Section IV-D.1, these experiments have two purposes. One can see that to ensure the detection of the event on a 10x10 grid one as to set the length of the path to 1200 (Figure 4) and to 4800 for a 20x20 grid (Figure 5). It suggest that the length needed to ensure the "convergence" to 1 is quadratic in the width of the square grid. Experiments on 30x30 and higher grids seem to confirm this fact. It is not strange since the propagation time is proportional to the surface of the grid. Moreover, using these experiments, we were able to fix, for the other experiments, a value of the length which is, in a sense, optimal. This is the value that ensures the good behavior of the system and that has the smallest cost possible (the shorter the paths we generate are, the better it is in term of computation time for APMC).

2) Impact of limited energy: In real sensor networks, the natural question is: "What is the best trade-off between the cost of the sensor and the energy that it must have?". Indeed, it is crucial in the design of a sensor node to provide it with enough energy to complete its task, but not too much in order to save money in the process of producing huge numbers of sensors.

The goal of our experiment (Section IV-D.2) is to show that there are values of the energy that are not sufficient to ensure the good behavior of the system. It is important to see that it is not the fact that the performance is worse with a small amount of energy, but the fact that the task is not completed anymore.

For example (see Figure 6), an energy of 15 is not sufficient in our model to make sure an event is detected. Indeed there is a plateau phenomenon: the value of the probability stays around 0.7 forever for paths with length greater than 700. This means that the probability for a fire to be detected is at most 70%.

3) Impact of loss: The third round of experiments, Section IV-D.3, measures the impact of the initial loss of sensors with infinite energy. Figure 7 clearly demonstrates it, with a quasi-linear loss of messages at the beginning, and then a severe degradation. Figure 8 amplifies the observation: with a nearly perfect spreading of sensors there is an exact correspondence between the percentage loss of sensors and that of messages. When about 40% of the sensors are lost, the probability of delivery of the message drops.

We interpret this fact by the existence of two phases in the functioning of the sensor network. The first phase, with more than 70% of working sensors, is "plastic" and robust to sensor loss: the message *will* be delivered, but with an (linearly) increasing delay. The second phase, below 60%, is more "solid" and can no longer cope with additional loss: the network is broken and not reliable anymore.

This observation naturally leads to the following question: *what can the operator do to improve and/or repair the network?* There are two obvious means: spreading new sensors, or build sensors with more energy to trade in energy for density.

4) Larger scale: In the view of the previous experiments, it is clear that another set of experiments is needed to evaluate the impact of energy versus density. If there are only few sensors but with a lot of energy for each of them, is the probability of detecting the event higher than in the case where there are a lot of sensors with only a limited amount of power?

The goal is to know what is the more prominent parameter in the behavior of the sensors: the energy or the density/number of nodes in the network. Basically the answer to this question is very important since it has an impact of the design of the sensors: is it better to have few costly sensors (low number/high energy) or a lot of cheap sensors (high number/low energy).

According to small scales experiments it seems that the answer is not so simple: the energy is important when the density is still not critical (like we showed in the experiments about the density) and that after a value (that depends on the size of the grid) even with infinite energy and very long paths it is not possible to ensure a correct behavior of the system (e.g. detection of the event).

And unfortunately, as reported in Section IV-D.4, we reached a limitation of the current implementation of APMC which is well suited for large models, but uneasy on large compositions of small components. These experiments are delayed to future work, once APMC adjusted to cope with a wider range of model types.

5) In general: We just want to argue about the fact that these experiments show the interest of our approach. We think that this interest is twofold: first it allows to verify *a posteriori* the correctness of the behavior of a large sensor network (routing algorithm, energy saving process...); second, and in our opinion this is the main point, it allows to modify *a priori* sensor and network algorithms by detecting flaws in the design of the system. This is crucial since the design and the production of sensor networks is a sensible and money-consuming activity.

B. On our modeling using Reactive Modules

While, Reactive Modules (RM) does provide some abstractions — constants (`BROADCAST...`) and formulas (*senses...*) — unfortunately they cannot be parametrized. As a matter fact, there is no clean parametrization features in RM, which is troublesome to implement a 20x20 grid of sensors. A weak form of parametrization is available, which is basically a clone-and-substitute: an existing module can be renamed into another one, with inner names being replaced by specified values. This mechanism proved to be inconvenient for the problem at hand, in particular because it does not cope easily with variations — such as the number of neighbors. To worsen the matter, RM does not provide loops or any other form of mechanization of the clone-and-substitute steps.

To put it in a few words, RM provides no meta-programming features. *Meta-programming*[1] denotes the ability to write programs that write programs, or rather in the current context, the ability to describe generic components and how to instantiate, parametrize, and tailor them *statically*, i.e., at compile time. Although providing higher levels of abstraction, meta-programming does not come with a run time penalty since the meta-program is run by the compiler at compile time, producing in an intermediate step a regular program. Some environments support meta-programming with varying degrees of flexibility and usability. C features a very limited meta-programming expressive power, but with a tolerable syntax thanks to `cpp`. M4 is one of the most pleasing alternative in the same tradition of lexical macro processors, but it comes with similar limitations: lack of consistence with the syntax of the host language, low-level lexical expansion that prevents catching errors of syntax or of typing, etc.

An interesting lead for future investigations is to define a higher level RM and its processing chain. A smart compiler performing traditional optimization techniques — such as constant folding, dead code elimination, copy propagation etc. — would tailor generic modules into specific ones just as we did for the handling of batteries (see Section IV-C). Two models of compilers can be considered, depending whether the target is (simple) RM, or some low level form suitable

[1] Meta-programming is given several meanings. We use it here in its sense of the possibility to run code *during the compilation*. *Macro-processors* provide a weak form of meta-programming, C++ 's templates provide meta-programming — as demonstrated by *expression templates* [31]— provide a similar builtin feature to C++, and multi-staged meta-programming pushes the concept even further (as in MetaML [32]).

for a model checker. In the first case, this compiler merely provides meta-programming to RM, while in the second case it becomes possible to imagine means to avoid the "space explosion" due to the creation of huge RM sources. Better yet, such a compiler could be tunable to trade space and time, enabling the expansion of higher level compact models into possible numerous fast and small components. Finally, any new abstraction (genericity, iteration constructs, etc.) allows the user to deliver a more faithful description of the model, enabling the compiler to discover possible symmetries or invariants which could help the model checker.

C. On APMC

This case study revealed a few facts about the adequacy of APMC for such experiments.

The built-in support for distribution made it possible to obtain results in reasonable time (from half an hour for small scale experiments, up to a couple of hours with bigger grids). The scheme implemented in APMC 2.0 [28] makes it very easy to deploy and very reliable, nevertheless it proved to be too inflexible: true load-balancing is needed to cope automatically with the varying availability of computers.

The recent addition by one of the authors of the support for checking multiple LTL formulas at the same time for each run considerably eased the gathering of all the figures that were needed. In fact, without such a feature, most of our graphs would have been impossible to make, or at least at a much higher cost — human and machine time. Some features were specifically added for these experiments, in particular to provide RM with random numbers, but this track was quickly dropped in favor of using pure RM with an additional M4 layer (see Section V-B for details on the input language). Finally, bigger grids exhibited some shortcomings in the current implementation that should be easy to fix: too optimistic timeouts that consider for instance that too long a C compilation means that the compiler failed, incorrect use of recursion in the LR parser that caused it to blow its stack, etc.

The limitation reported in Section IV-D.4 will draw the attention of the APMC designers in the short future. Various well known solutions exists that would improve the memory consumption in several ways. Another interesting path of investigation consists in making profit from the form of formulas. For instance formulas that converged to their limit no longer need to be evaluated, and henceforth, some paths no longer need to be computed. This is the case, for instance, of formulas such as "Probability of arrival before instant T" when t is already greater than T.

VI. Conclusion

In this paper we presented the modeling of a sensor network whose goal is to detect the apparition of an event on a grid of arbitrary size and to send a signal to some specific nodes. This model can be seen as an abstract version of real life sensor networks for various tasks. We also conducted several experiments, showing the interest of using APMC for the simulation/verification of sensor networks.

Basically, we learned from this case study the peculiarities of sensor networks in the framework of approximate probabilistic model checking. For example, due to the large number of nodes, even the modeling task can be a challenge, and the need of specific tools for modeling seems to be one of the issue for the verification of such systems.

In the future, we plan to develop specific tools for the purpose of verifying sensor networks. Both for improving the modeling and the writing of specifications.

References

[1] D. E. Culler, D. Estrin, and M. B. Srivastava, "Guest editors' introduction: Overview of sensor networks." *IEEE Computer*, vol. 37, no. 8, pp. 41–49, 2004.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey." *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[3] E. Clarke, O. Grumberg, and D. Long, "Model checking," in *Proceedings of the NATO Advanced Study Institute on Deductive program design*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996, pp. 305–349.

[4] G. J. Myers, *Art of Software Testing*. New York, NY, USA: John Wiley & Sons, Inc., 1979.

[5] R. Lassaigne and S. Peyronnet, "Approximate verification of probabilistic systems." in *PAPM-PROBMIV*, ser. Lecture Notes in Computer Science, H. Hermanns and R. Segala, Eds., vol. 2399. Springer, 2002, pp. 213–214.

[6] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking." in *VMCAI*, ser. Lecture Notes in Computer Science, B. Steffen and G. Levi, Eds., vol. 2937. Springer, 2004, pp. 73–84.

[7] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashitet, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks*, vol. 46, no. 5, pp. 605–634, December 2004.

[8] A. Mainwaring, J. Polastre, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[9] Q. Cao and T. Abdelzaher, "A scalable logical coordinates framework for routing in wireless sensor networks," in *Proceedings of the IEEE Real-time Systems Symposium (IEEE RTSS'04)*, Lisbon, Portugal, December 2004.

[10] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proceedings of the ACM MobiCom Conference*, September 2003, pp. 96 – 108.

[11] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao, "Modeling of sensor nets in ptolemy ii." in *IPSN*, K. Ramchandran, J. Sztipanovits, J. C. Hou, and T. N. Pappas, Eds. ACM, 2004, pp. 359–368.

[12] A. Sobeih, W.-P. Chen, J. C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang, "J-sim: A simulation environment for wireless sensor networks." in *Annual Simulation Symposium*. IEEE Computer Society, 2005, pp. 175–187.

[13] B. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing." in *IPSN*. IEEE, 2005, pp. 477–482.

[14] A. Savvides, S. Park, and M. B. Srivastava, "On modeling networks of wireless microsensors." in *SIGMETRICS/Performance*. ACM, 2001, pp. 318–319.

[15] S. Park, A. Savvides, and M. B. Srivastava, "Simulating networks of wireless sensors." in *Winter Simulation Conference*, 2001, pp. 1330–1338.

[16] P. Volgyesi, M. Maroti, S. Dora, E. Osses, and A. Ledeczi, "Software composition and verification for sensor networks," *Sci. Comput. Program.*, vol. 56, no. 1-2, pp. 191–210, 2005.

[17] 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.
[18] 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.
[19] 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.
[20] P. Dagum, R. M. Karp, M. Luby, and S. Ross, "An optimal algorithm for monte carlo estimation." *SIAM J. Comput.*, vol. 29, no. 5, pp. 1484–1496, 2000.
[21] D. Monniaux, "An abstract monte-carlo method for the analysis of probabilistic programs." in *POPL*, 2001, pp. 93–101.
[22] M. Vardi, "Automatic verification of probabilistic concurrent finite-state programs," in *FOCS*, 1985, pp. 327–338.
[23] J. Darbon and S. Peyronnet, "Approximate probabilistic model checking for programs," in *Submission*.
[24] S. Peyronnet, "Apmc webpage," in *http://apmc.berbiqui.org*.
[25] M. Duflot, L. Fribourg, T. Hérault, R. Lassaigne, F. Magniette, S. Messika, S. Peyronnet, and C. Picaronny, "Probabilistic model checking of the csma/cd protocol using prism and apmc." *Electr. Notes Theor. Comput. Sci.*, vol. 128, no. 6, pp. 195–214, 2005.
[26] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala, "Symbolic model checking of probabilistic processes using mtbdds and the kronecker representation." in *TACAS*, ser. Lecture Notes in Computer Science, S. Graf and M. I. Schwartzbach, Eds., vol. 1785. Springer, 2000, pp. 395–410.
[27] R. Alur and T. A. Henzinger, "Reactive modules." in *LICS*, 1996, pp. 207–218.
[28] G. Guirado, T. Hérault, R. Lassaigne, and S. Peyronnet, "Distribution, approximation and probabilistic model checking," in *PDMC*, 2005.
[29] R. Friedman and G. Korland, "Timed grid routing (TIGR) bites off energy," in *Proceedings of MobiHoc 2005*, 2005.
[30] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–66, August 2001.
[31] T. L. Veldhuizen, "Expression templates," *C++ Report*, vol. 7, no. 5, pp. 26–31, June 1995, reprinted in C++ Gems, ed. Stanley Lippman.
[32] W. Taha and T. Sheard, "MetaML and multi-stage programming with explicit annotations," *Theoretical Computer Science*, vol. 248, no. 1–2, pp. 211–242, Oct. 2000. [Online]. Available: http://www.elsevier.nl/gej-ng/10/41/16/183/21/27/abstract.html;http://www.elsevier.nl/gej-ng/10/41/16/183/21/27/article.pdf