Worst-Case Lifetime Computation of a Wireless Sensor Network by Model-Checking[*]

Laurent Mounier
Verimag
Gières, France
Laurent.Mounier@imag.fr

Ludovic Samper
France Telecom R&D, Meylan, France
and Verimag, Gières, France
Ludovic.Samper@orange-ftgroup.com

Wassim Znaidi
Verimag
Gières, France
Wassim.Znaidi@imag.fr

ABSTRACT

Wireless Sensor Network (WSN) technology is now mature enough to be used in numerous application domains. However, due to the restricted amount of energy usually allocated to each node, a crucial property of interest for the users is the minimal lifetime of the network. In practice, this value strongly depends both on the design choices performed for each network element (hardware architecture, communication protocols, etc.) and on the whole execution environment (physical environment, execution platform, network topology, etc.). We propose here an original approach to evaluate this minimal network lifetime based on model-checking techniques. It consists first in designing a timed model of the entire network behavior (taking into account its execution environment), and then to compute on the state space associated to this model the shortest execution sequences (from a temporal point of view) leading to some states considered as "terminal" (from the network lifetime point of view). This approach is illustrated on a concrete example of a WSN application to compare the influence on the network lifetime of two classical routing algorithms.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Experimentation, Verification

Keywords

Wireless Sensor Networks, Energy, Modeling, Analysis, Model-Checking

1. INTRODUCTION

Although wireless sensor networks have emerged only recently, this technology is now mature enough to be deployed in many real world applications. For most of these applications, one of the key properties expected from the users is that a minimal lifetime of the network infrastructure will be guaranteed in spite of limited energetic resources allocated to each nodes. Although a huge number of technical solutions have been proposed to maximize this lifetime (dedicated node hardware architectures, lightweight operating systems, low-consumption communication protocols, etc.), ensuring that a complete WSN application will fulfill its lifetime requirement is still a quite difficult task. In particular, the network architect has to be convinced that his design choices are the most appropriate with respect to the application of interest, for a given physical execution environment. A classical approach to tackle this problem is to use *simulation tools*, able to produce quantitative data on particular execution sequences (i.e., energy consumed by each hardware components, activity periods of each network element, etc.). Depending on the granularity of the simulation steps and on the faithfulness of the input models, these simulation results can be quite accurate, and they allow to predict some reasonable lifetime values. However, these predictions still depend on the amount of simulation runs performed, and on the criteria used to select these runs (either randomly, or according to some execution profile provided by the user). Therefore, the values obtained may not always correspond to *minimal* lifetimes, in particular because (seldom) worst case executions scenarios can be ignored during the simulation campaign.

The objective of this work is to provide a complementary approach based on *model-checking* techniques. It consists in designing first a behavioral description of the whole network, including its physical execution environment. This description is written using an appropriate specification language, with a precise operational semantics. This semantics then defines a *model* of the network behavior, consisting in a state graph representing all its possible execution sequences (the states encode the control location and data values of each nodes, whereas the transitions describe the operations performed when changing from one state to another). Interesting properties on the network behavior can then be checked during some traversal of this state graph. This technique is implemented in several tools, and it has been successfully applied to verify correctness requirements of non-trivial hardware or software systems.

[*]This work was partially supported by the French Ministry of Research under contract ARESA ANR-05-RNRT-01703.

PE-WASUN'07, October 22, 2007, Chania, Crete Island, Greece.
Copyright 2007 ACM 978-1-59593-808-4/07/0010 ...$5.00.

We propose here to adapt this technique in order to find on the state graph the worst-case execution sequences from the network lifetime point of view (i.e., execution sequences corresponding to a minimal lifetime of the network). To do so, we consider a behavioral network specification including both *time duration* and *energy consumption* indications. Thus, it is possible to characterize in the derived state graph the set of "final" states, from the network infrastructure lifetime point of view, according to a given criterion (e.g., more than half of the nodes have spent their initial amount of energy). Hence, computing the shortest paths (from the time duration point of view) leading to such final states gives both the minimal lifetime of the network, and a worst case execution scenario.

Clearly, the main advantage of this approach is its exhaustiveness: it allows to detect seldom worst case execution sequences, that would be beyond the capabilities of classical simulation campaigns. Furthermore, since the complete worst case sequence is produced, it is possible to investigate why such a bad scenario could occur, and, if necessary, how to improve the network design. Similarly, by changing the network specification, for instance by replacing a MAC protocol by another one, different design choices can be compared from the lifetime point of view. However, this approach may also suffer from a potential drawback: the size of the state graph may prevent any practical application for a realistic network design, with several hundred of nodes. A possible solution is then to consider only a subset of the network, with a smaller number of nodes, and to focus the specification of each node on some particular "layers" (e.g., the MAC layer, or the routing layer), the other layers being described only at a more abstract level. Of course, the results obtained from such a model would be less accurate, but the worst case scenario produced could then be used as inputs to a classical simulation tool (operating on a detailed network description). Thus, this would provide an automatic way to produce execution scenario of interest for simulation.

We illustrate this approach on a classical WSN application for environmental monitoring, where nodes are supposed to send alarms to a base station when a threshold value is reached on their sensing device. First we explain how a timed model of this application can be obtained using the IF specification language. Then, we show how the model-checking tools available in the IF toolbox can be used to compute the minimal lifetime of the network. In particular, we compare the performances of two routing algorithms with respect to this criteria.

The rest of the paper is organized as follows: Section 2 gives some informal description of the application we consider, section 3 explains how it has been specified using IF and section 4 indicates how we proceeded to compute minimal lifetimes and presents the results we obtained. Then, a comparison with related works is performed in section 5, and section 6 gives some conclusion and perspective to future works.

2. A TYPICAL WSN APPLICATION

We present in this section what is for us a wireless sensor network. Our study focuses on the routing layer however it is impossible to analyze routing algorithms without any context. That is why we explain here our assumptions on higher and lower levels, application and medium access control respectively. We also detail the routing protocols we analyze. As we evaluate consumption, our modeling framework contains a consumption model. Our hypothesis on energy models are also presented. Lastly, we define criteria we used for network lifetime.

2.1 The Application

The range of sensor networks applications is really wide. We will concentrate on environment monitoring applications. In these applications, network reacts when an event occurs in the environment. As already mentionned, the role of the environment cannot be neglected in those applications [17].

In our example, when a sensor detects that pollution is higher than a threshold, it sends an alarm message to the sink. We assume only one single sink. The sink is a powerfull node that collects alarms (to warn the rescue) and monitors the network.

2.2 The environment

On sensor networks traffic is mostly generated by the environment. In our application of pollution detection, we need a mobility model of the target cloud. We assume that the target cannot be at several nodes at the same time. At most one node is stimulated at one moment. Moreover pollution, our target, cannot jump: when the cloud is located on one node, at the next instant it can still at the same position or move to a neighboring node.

2.3 The Routing Layer

We compare two routing protocols. Our aim is to validate the modeling method, not to evaluate new routing protocols. Hence, we study classical sensor networks routing protocols: flooding and directed diffusion.

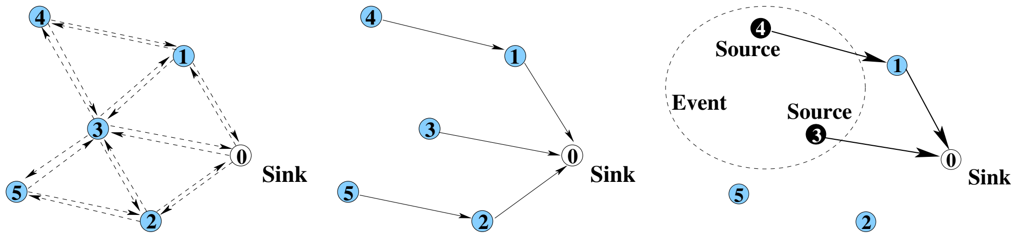
Flooding is a very greedy algorithm: each packet is sent to the whole network. When a node receives a packet, it broadcasts the packet to all its neighbors except if it has already send this packet. This prevents packets from doing loops in the network. This algorithm is really sub-optimal as each packet is sent to the whole network. Nevertheless, it doesn't need any infrastructure nor self-organization of the network.

The second routing algorithm we modeled in IF is directed diffusion [11]. See figure 1. In the first phase, which can be considered as the organization phase, the sink sends an interest message to the whole network, fig 1(a). This message is sent using the flooding routing mechanism. When a node receives a new interest, it sets up a gradient between the sender and itself (fig 1(b)) and it forwards the packet. This gradient is sets up only at the first reception of an interest. This means that the selected route is the faster one. The application is pollution detection, thus the interest is "send an alarm when the cloud is detected". Lastly when the event is detected (figure 1(c)), nodes send alarm to the sink using the gradients they set up.

2.4 Assumptions about the MAC and link layers

As we concentrate on routing, we will not detail the whole protocol layers of our network nor every components of the sensors. Nevertheless, the network we model is a sensor network and to be realistic, we have to take into account all the characteristics of such a network.

Sensor network MAC protocols are designed to be energy-



(a) Broadcast interest (b) Setup gradient (c) Send alarm

Figure 1: Directed diffusion

aware. For that the MAC protocol switches the radio off as often as possible. It implies additional delays in packet transmissions.

As nodes communicate via radio waves, errors due to collisions or noise can occur on the channel. To have guaranties about the reception of messages, some MAC protocols use acknowledgement packets. This means that the receiver replies an acknowledgment (ACK) packet to the sender just after the correct reception of the message. Thus, the sender knows that its packet was correctly received. If this ACK packet does not arrive, the sender transmits again its message. However, this mechanism cannot apply when a packet is sent to multiple nodes. If a broadcast packet was acknowledged, acknowledgments packets would arrive at the sender at the same time and then collide. We will assume that unicast messages are acknowledged but that broadcast or multicast packets are not. First, we think it is realistic and furthermore we show that our modeling language is expressive enough to model different kinds of communication that can all happen in a sensor network.

The aim of the work is to use model-checking to check properties that involve energy on sensor networks. For that, we need to evaluate the energy consumption of the sensors. In the sensor each element consumes: the radio, the microcontroller, memories and the sensor. In our application, nodes do not have to process data thus we will consider that packet emission and reception are the major consumption sources.

We will explain IF model of our network in section 3.

2.5 Lifetime Criteria

It is difficult to have a precise criterion that determines if the network is still alive or if it is dead. Let us assume that nodes do not recharge their batteries so that we can almost say when a node is dead. For the whole network, it is more difficult. The network is dead when it cannot assume its service, but this criterion depends on the application! Thus, we propose to study several criteria:

- The first node ran out of energy.
- The network is broken: almost two partitions appear in the network. As we assume a single sink, this implies that some nodes that still have energy can no more communicate with the sink.
- A fraction of the network is dead. A certain percentage (10 %, 50%, ... or whatever) of the nodes ran out of energy.

3. MODELING

3.1 The IF Modeling language

IF is a description language for asynchronous timed systems. An IF specification consists in a set of *processes*, where process instances run in parallel and interact either through shared variables or message passing. We briefly present below the main features provided by this language (see [8]) for a more complete description).

Functional features.

The general behavior of a process is described by a *timed automaton*, extended with data. Each process instance owns a set of *variables* (that can be declared as "public" or "private") and a (FIFO) *queue* to store pending messages (i.e., that have been received but not yet consumed by this instance).

A process can move from one control state to another by executing some *transition*. Two kinds of transitions are available: spontaneous transitions and transitions triggered by a message reception (the transition is then enabled only if this message is in first position in the input queue). Execution of a transition can also be restricted by boolean predicates on variables. Notice that several transitions may be enabled at the same time, allowing non-determinism. Transition bodies are sequential programs consisting of elementary *actions* (variable assignments, message sending, process creation/destruction, etc). They are structured using control-flow statements (like if-then-else, while-do, etc).

Inter-process communication media can be specified by means of *Signalroutes*, that transport messages. The behavior of a signalroute is described by its delivery policy (FIFO or multi-set), its connection policy (peer to peer, unicast or multicast), its delaying policy and its reliability (reliable or lossy). More complex communication media can be described explicitly as IF processes.

Finally, the IF notation provides several predefined basic data types (bool, integer, real, pid) and a special type called clock to measure time progress (see below). Structured data types are built using usual type constructors (enumeration, range, array, record).

Time model.

The time model of IF is that of *timed automata with urgency* [7]. As for regular timed automata [5], the execution of a transition is instantaneous, and time is allow to progress only between the execution of two transitions. Moreover, an urgency attribute is associated to transitions to indicate their priority over time progress. We distinguish between eager, lazy and delayable transitions. Eager transitions are urgent, and they have to be executed as soon as they become enabled (except if they are disabled by executing another eager transition). Lazy transitions are never urgent, and they may be disabled by time progress. Delayable transitions are a combination of both eager and lazy: they are not urgent, except for the moment when time progress would disable them.

Like in timed automata, time distances between events are measured by variables of type "clock". Clocks can be created, set to some value or reset (deleted) in any transition. They can be used in time guards to restrict the time points at which transitions can be taken. Local clocks allow the specification of timing constraints, such as *task duration* (modeled by time passing in a state associated to this task) and *deadlines*. Global time constraints, such as *end-to-end delays*, can be expressed by means of global clocks. In addition, the use of delayable transitions allows to specify *time non-determinism* (i.e., time durations chosen within some given intervals).

3.2 Modeling the application

The WSN application we consider (section 2) consists in a network of *nodes*, communicating via *radio links*, and evolving within a given *environment*. The corresponding IF specification provides a formal models for each of these three elements.

Modeling the environment.

The external environment of this application is supposed to stimulate the node sensors with some input events indicating when a given pollution level has been reached. In the real world, the time and geographical distribution of these inputs follows some specific laws: it depends on a physical source (e.g., a pollution cloud), possibly influenced by external factors (wind, etc). We chose here to model this environment by a dedicated IF process (`Environment`), running in parallel with the application. This process repeatedly mimics the behavior of a "pollution cloud" as follows:

- send an `envInput` message to a selected node n, indicating that this node is currently subject to a pollution source;

- non deterministically choose a direct neighbor of n (including n itself) to be the next selected node.

The execution period of this cyclic behavior is given by a time constant, and changing this constant allows to model various "wind" profiles.

Modeling a node.

Network nodes are specified using an IF process called `Node`, each individual node being a specific instance of this process (the sink node is considered as a particular node, distinguished by its instance number). At a coarse level, a node behavior strictly follows the informal description of the application, depending on the routing protocol under consideration:

- For the *flooding* protocol, upon reception of an `envInput` message, each node broadcasts a (uniquely identified) `Alarm` message to its neighbors. When a (non duplicated) `Alarm` message is received by a node, it is propagated using the same mechanism (broadcast to each neighbors).

- For the *directed diffusion* protocol, the sink node first initiates the construction of the virtual routes by sending an `Interest` message using the flooding protocol. Thus, each node stores the identity of its "next neighbor" along this route. When an `envInput` message is received, each node unicasts an *Alarm* message to this "next neighbor", that will be propagated up to the sink node using the same mechanism (unicast to each "next neighbor").

In addition, the model of a node should also take into account the energy consumption. We assume here that the only consuming operations are the basic communication primitives, i.e., sending/receiving a broadcast/unicast message. Therefore, each of these four operations performed by a nodes decreases its total amount of energy available by a given value. This value depends only on the nature of the operation, not on the message length. When this total amount of energy is spent, the node becomes *inactive* (it does no longer send any message).

Modeling the radio communications.

Modeling the radio communications means taking into account the network topology, the communication durations (introduced by the MAC and physical layers), and the communication hazards. To do so, we introduce an additional process, `Topology`, owning a shared matrix that contains the neighborhood relation (from the radio range point of view). This matrix is initialized with a given topology (provided by the user), and it does not change during the network lifetime. Communication operations are then modeled as follows:

- A broadcast emission consists in sending a message to all the neighbors of the transmitter, in a single atomic operation, taking a constant time duration. To do that with IF, we used a *while* loop in which the sender emits an unicast signal to each of its neighbors. From the network point of view, messages are sent simultaneously. In case of *collision*, when one of the transmitter's neighbor is already involved in another communication, then this neighbor miss (silently) the message sent (broadcasts are not acknowledged).

- Unicast communications are supposed to be reliable (i.e., acknowledged). As long as collisions occur, the sender transmits again its message, after a non deterministic waiting time (taken within a given interval and modeled using IF *delayable transitions*). Therefore, the overall amount of time and energy required for a unicast communication depends on the number of attempts performed.

Finally, a (global) clock called `lifetime` is attached to the process `Topology`. This clock is started when the network begins to work, and it is used to compute its minimal lifetime duration (see section 4).

4. WORST CASE LIFETIME COMPUTATION

In this section, we explain the approach we used to compute the worst-case lifetime of the network from the IF specification. Then we give the results obtained with the two specific routing protocols we considered in our case-study.

4.1 Lifetime computation

The IF operational semantics allows to represent the behavior expressed by an IF specification as a *state graph*, where state encode the control location and data values of each IF processes, and transition corresponds to the operations performed at the IF level. This state graph can be computed on demand by the *IF simulation engine*. Several tools are then available to explore this state graph, providing various validation facilities [8] (e.g., interactive simulation, test generation, model-checking, etc.).

In particular, computing the worst-case lifetime of the network can be reduced to find a *shortest path* (regarding execution time) between the initial state of the graph (when the network starts working) and a state that corresponds to a *goal state* with respect to the lifetime criterion we consider (either at least a node is inactive, or the set of inactive nodes split the network into several strongly connected components). Since the `lifetime` global clock introduced in the IF model stamps each state of the graph with the time elapsed since the initial state, such shortest paths can be computed by means of usual graph exploration algorithms. After having tried several algorithms we chose here to use the (classical) A* algorithm, which is recalled below.

```
A* Algorithm
$p$ :  state
$Q$ :  set of pairs (state, duration)
$V$ :  set of (visited) states
begin
$Q \leftarrow$ (initial_state, 0) ; $V \leftarrow \emptyset$
while $Q \neq \emptyset$ do
  extract from $Q$ a pair $(p, d)$ s.t.$d$ is minimal
  if $p \notin V$ and not is_goal($p$) then
    foreach $q$ in succ($p$)
      $Q \leftarrow Q \cup \{(q, cost(q))\}$
  else
    if is_goal($p$) then
      return $d$
    endif
endwhile
end
```

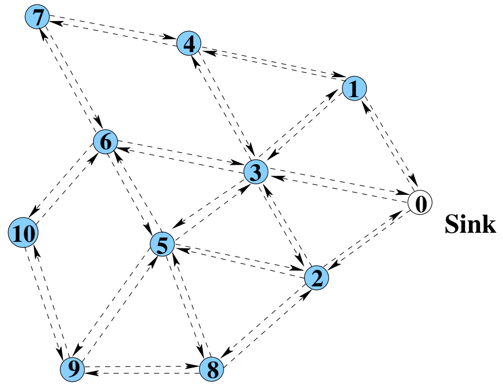


Figure 2: Different experiments, from 4 to 11 nodes

The efficiency of this algorithm is based on the following elements. First, the set Q (pending states) is implemented as a *priority queue* on the duration attribute. Second, the $cost(q)$ function is the sum of two values: the time elapsed to reach q from the initial state (obtained by consulting the value of `lifetime` in state p), and the *estimated remaining time* to reach a goal state. This second value is used to tighten the set to explore by privileging the states “that seems” bring us closer to the goal state. This is obtained by using a heuristic estimation. To ensure the optimality of the algorithm this heuristic estimation should be *admissible*: it should never overestimates the time duration required to reach a goal state to ensure that A* is optimal. The heuristic we chose is based on the following consideration: making a node inactive requires to spend at least the amount of time needed to repeatedly perform on this node the most energy consuming operation. Indeed, this amount of time is always less or equal than the time needed to reach a goal state following any possible execution sequence from a current state of the graph. Assuming that the most energy consuming operation requires an energy amount of e_0 and takes a duration d_0 to be executed, then function *cost* is defined as follows:

```
cost(p) =
let
  d = the value of lifetime in state p
  e = the energy of the weakest node in state p
  x = $e/e_0$
  d’ = $x \times d_0$
in
  d + d’
```

Finally, this algorithm can be easily extended to provide not only the worst-case lifetime value, but also a corresponding execution sequence. Such a scenario is then particularly useful to better understand this worst case behavior (see below).

4.2 Experimental results

We performed this worst-case lifetime computation analysis on the IF specification considering the two routing algorithms described above.

We did experiments with several network sizes. Figure 2 shows the different topologies we experimented, the number of nodes ranges from 4 to 11. For instance, our four-node network is the network containing nodes number **0**, **1**, **2** and **3**. As explained before, our model-based technique relies on the (partial) traversal of a state graph. The size of this state graph, and hence the computation time, highly depends on the number of nodes we consider in the network: for a four nodes networks shortest path computation takes less than three minutes whereas for an eleven nodes network it lasts more than two days. Therefore, we finally chose to perform our experiments on a six nodes network, for which the computation times takes about six hours on a powerfull PC computer. All results concern the six-node network shown on figure 1.

Other important experimental parameters are the energy consumptions assigned to the different actions. Costs of emission and receptions had been set up to 3 and 2 energy units respectively. It is not our topic here to discuss about accurate costs of emission and reception regarding specific hardware architectures. Of course any other values could have been used. Battery capacities of nodes (except the sink) have a capacity of 40 energy units. It means that one node can send no more than 14 emissions during its whole life. This is very few. Indeed, we study the network behavior only during a small time interval. We could deduce the *real* network lifetime with more realistic battery capacities by extrapolating the results obtained. Nevertheless, our goal is essentially to compare protocols, not to have an absolute value of the network lifetime. Let us point out that a similar abstraction is performed when using simulators: a simulation campaign does not cover the entire network lifetime. About time, a transmission can take between 1 and 3 time units, if a retransmission is needed (in case of collision), it takes 5 more time units. The cloud sends a polluting signal each 7 time units.

To evaluate **lifetime**, we considered two lifetime criteria:

- First criterion: the network is dead when one node dies.
- Second criterion: death of the network is when the network is no more connected. Let us precise that if directed diffusion is the routing protocol, one new interest request is sent by the sink when a node dies. Thus when one node runs out of energy, the network self repairs if possible, otherwise, it is dead.

Table 1 gives the lifetimes of the network (in time units) under different routing protocols (flooding and directed diffusion) for our two criteria. As expected, the protocol directed diffusion performs better from the lifetime point of view than flooding. Indeed, to send an alarm to the sink with flooding, much more messages are generated. Moreover, the second lifetime criterion (network partition) takes more time to be reached. Indeed, if the network is partitioned, at least one node ran out of energy whereas one node can be dead without partitioning the network. That is why the first criterion is always satisfied before the second.

	1^{st} criterion	2^{nd} criterion
Flooding	14	21
Directed Diffusion	41	52

Table 1: Minimum lifetime duration

To underline the differences between a worst case and a medium case, we compared on table 2 worst-case lifetimes with medium-case lifetimes. IF simulator, from the IF toolbox, can be used to generate a possible execution sequence of the system. It corresponds to a random exploration of the state graph (without the need to generate it). This exploration leads to a simulation of the system. Here, we used this tool to generate possible execution scenarios of our network. The lifetime reported in table 2 corresponds to a mean on ten scenarios. This comparison allows to measure the importance of computing the worst-case scenario with respect to an "average" scenario, as obtained by a standard simulation tool.

We have demonstrate that our modeling approach can lead to an estimation of the worst case network lifetime. Moreover, we can exhibit the worst-case scenario corresponding to this shortest lifetime. As an example, we analyzed the worst-case scenario for directed diffusion. Figure 3 shows the gradients set up by nodes in this worst-case scenario. We can already notice that routes are not as expected. On figure 1(b), we have depicted the expected routes which seems more "natural" as they are the shortest paths between the node and the sink. Here, with our modeling hypothesis, we observe that unexpected gradient installation can occur reducing the lifetime up to 60% (table 2). In this worst case scenario, the cloud activates nodes number **3** and **4** in turns.

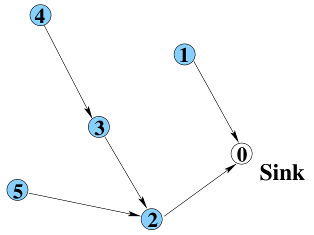


Figure 3: Gradients set up in directed diffusion for the worst case scenario

	Worst case	Mean case
Flooding	14	24
Directed Diffusion	41	101

Table 2: Computation of the lifetime for the 1^{st} criterion

5. RELATED WORKS

Sensor networks are complex distributed systems that need to be simulated, evaluated or checked before their deployment. Indeed, many optimizations are done at each layer and researchers need tools and methods to evaluate their advances.

People from network research community are used to work with network simulators. A huge number of network simulators were developed, each corresponding to a particular need. One of the most popular network simulator is NS-2 [2], The Network Simulator. As it is popular, numerous protocols for ad hoc networks are implemented in NS-2, which offers the ability to easily compare two protocols. However, NS-2 is not dedicated to sensor networks and the computation of the energy consumption is not included in the simulator. Therefore numerous simulators were developed specifically for WSN applications, able to take into account the energy consumption. A major issue is then to obtain a correct trade-off between an accurate estimation of the energy consumption and the scalability of the simulator (to handle large networks). Our point of view is that network simulators are useful tools to have ideas about the *medium* case whereas our exhaustive approach gives information about the *worst* case.

Since Kleinrock and Tobagi ([12]), a classical method to study wireless networks is to use the probability theory. In some sense this technique is more exhaustive than simulation where only a medium case is studied. However the lack of expressiveness of this approach (from the functional point of view) prevents correct estimations of the energy consumption. In particular it is really difficult to express the complex behavior of a sensor network and its energy consumption using only probabilistic models.

As far as we know, there are not formal tools specially dedicated to sensor networks validation. However, due to the success of these tools in other application domains (communication protocols, embedded systems), several attempts have been performed to experiment them for sensor networks issues. We summarize here some of these works.

Watteyne et al. propose in [18] a real-time MAC protocol for sensor networks; then they had to simulate it on large scale networks. As their application is critical, providing a formal validation is essential. However, validation here only consists in verifying that in any case, messages arrive on time at the sink node. Properties they check concern behavior only and not the energy consumption. They have used UPPAAL [4], as modeling and formal validation architecture. This example shows the interest of formal validation in the context of ad hoc networks, however it is not "sensor-network-specific". Our application focuses on sensor network key issue as we address the problem of energy and provide a way to guaranty a minimum lifetime.

A well-known framework to develop sensor network applications is TinyOS [3]. Authors of [9] modeled a node described in nesC using HyTech [10]. HyTech is a tool to model and analyze hybrid automata. Authors first modeled a single node described using hybrid automata. With this model, they perform safety checks with HyTech. Here again, what they verify does not concern energy. Then, using SHIFT [1], they simulate a network of hybrid automata that corresponds to a sensor network. Here, the simulation concerns energy consumption but it is still simulation, they are not doing exhaustive state space exploration.

Authors of [16] show on a case study that Real-Time Maude [15] provides a good framework to model, simulate and analyze a sensor network. Our work is directed at the same object. Indeed, it is not more difficult to model a sensor network problem using a formal framework as Real-Time Maude or IF than using a classical network simulator. Then, with the simulation engine of the formal framework, simulations of the model can be performed leading to the same results as with a network simulator. Moreover, thanks to the model-checker, formal verifications can also be done using the same model. Let us precise that model-checking concerns a smaller network (up to 6 nodes in their paper). However, our case-study is more sensor network-specific. First, we model the radio channel: collisions and delays can occur. In their study, perfect MAC and physical layers are assumed. Secondly, we tackle the problem of energy that is crucial in sensor networks, in their work verification only concerns functional properties.

A complementary approach consists in developing models and tools dedicated to verify power systems. In those methods, the user have to model its problem in a dedicated formalism. Formalisms are such that there exists an efficient representation to find the worst case scenario. The problem with those dedicated formalisms is that they can be dedicated to a specific problem and it can be too much restrictive to express another consumption problem in the dedicated formalism. We did not use such a formalisms in this work. However it could have been useful in order to model one node. Our node models are very abstract, those abstractions could be the result of the analysis of a more detailed model.

6. CONCLUSION

We have proposed a model-based approach to automatically compute the worst-case lifetime duration of a WSN application. This approach consists in designing first a global specification of the network, using a formalism that combines a sufficient expressiveness level and a well-defined operational semantics. Thus, the exhaustive behavior expressed by this specification can be modeled as a state graph, and worst-case lifetime estimation is then reduced to a particular shortest paths computation on this state graph, using classical algorithms. This approach has been illustrated on a representative case study to compare two routing algorithms from the energy consumption point of view.

We strongly believe that this approach increases the set of techniques available to estimate the efficiency of a WSN architecture. In particular it automatically computes information about *seldom execution sequences*, that are beyond the scope of classical simulation tools (mostly tailored to analyze a system either on *average scenarios*, or according to some specific *input profiles*). Therefore, we claim that the approach we propose is particularly suitable in the early phases of the system development, for so-called *design exploration*, when choosing the most appropriate network architecture (hardware components, protocol stack, network topology, etc.), for a given application and according to a specific execution environment.

This preliminary work could be extended into several directions. First of all, it is clear that such exhaustive state exploration methods are limited by the size of the state graph that can be analyzed. To scale up, these methods should integrates sound *abstraction techniques*, allowing to reduce the size of the model to explore while preserving the properties of interest (namely the worst case lifetime duration). In this particular context these abstraction techniques should allow to focus on a particular design level (e.g., the routing algorithm), and simplify the description of the other levels. We are currently investigating how to automatically prove the soundness of such abstractions. A second direction would be to improve the lifetime computation algorithm itself. Many interesting research works have been performed in the model-checking community to alleviate the state explosion problem, and they could be re-used in this particular context. Two of them are clearly appealing: directed model-checking ([14]), and symbolic state space representations allowing to handle both dense-time and associated cost values ([13]). Finally, it would also remain to perform more detailed experiments, to better investigate the possible combination of this model-based technique with more classical simulation approaches. In practice, this could be done using the translation tool [6] from nesC to BIP (an extension of the IF formalism), that allows to derive formal specifications from a nesC application.

7. REFERENCES

[1] The hybrid simulation programming language. http://www.path.berkeley.edu/shift.

[2] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[3] Tinyos. www.tinyos.net.

[4] Uppaal home page. http://www.uppaal.com/.

[5] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] A. Basu, L. Mounier, M. Poulhiès, J. Sifakis, and J. Pulou. Using bip for modeling and verification of networked systems - a case study on tinyos-based networks. In *Proceedings of the 6th IEEE International Symposium on Network Computing and Applications*, July 2007.

[7] S. Bornot and J. Sifakis. An Algebraic Framework for Urgency. *Information and Computation*, 163:172–202, 2000.

[8] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The if toolset. In F. Corradinni and M. Bernanrdo, editors, *Proceedings of SFM'04*, volume 3185 of *LNCS*. Springer-Verlag, September 2004.

[9] S. Coleri, M. Ergen, and T. J. Koo. Lifetime analysis of a sensor network with hybrid automata modelling. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 98–104. ACM Press, 2002.

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*, pages 460–463. Springer-Verlag, 1997.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MOBICOM, The Annual International Conference on Mobile Computing and Networking*, pages 56–67, 2000.

[12] L. Kleinrock and F. A. Tobagi. Packet switching in radio channels: Part ii–the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. In *IEEE Transactions on Communications*, volume 23, pages 1417–1433, december 1975.

[13] K. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. *Lecture Notes in Computer Science*, 2102:493–505, 2001.

[14] A. Lluch-Lafuente, S. Edelkamp, and S. Leue. Partial order reduction in directed model checking. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*, pages 112–127. Springer-Verlag, 2002.

[15] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of real-time maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.

[16] P. C. Ölveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in real-time maude. In *The 14th International Workshop on Parallel and Distributed Real- Time Systems*, 2006.

[17] L. Samper, F. Maraninchi, L. Mounier, E. Jahier, and P. Raymond. On the importance of modeling the environment when analyzing sensor networks. In *IEEE International Workshop on Wireless Ad-hoc and Sensor Networks (IWWAN)*, June 2006.

[18] T. Watteyne, I. Augé-Blum, and S. Ubéda. Dual-mode real-time mac protocol for wireless sensor networks: a validation/simulation approach. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, pages 2–8. ACM Press, 2006.