

Scalable Modeling and Performance Evaluation of Wireless Sensor Networks *

YoungMin Kwon and Gul Agha
Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, IL 61801, USA
{ykwon4, agha}@cs.uiuc.edu

Abstract

A notable feature of many proposed Wireless Sensor Networks (WSNs) deployments is their scale: hundreds to thousands of nodes linked together. In such systems, modeling the state of the entire system as a cross-product of the states of individual nodes results in the well-known state explosion problem. Instead, we represent the system state by the probability distribution on the state of each node. In other words, the system state represents the probability that a randomly picked node is in a certain state. Although such statistical abstraction of the global state loses some information, it is nevertheless useful in determining many performance metrics of systems that exhibit Markov behavior. We have previously developed a method for specifying the performance metrics of such systems in a probabilistic temporal logic called iLTL and for evaluating the behavior through a novel method for model checking iLTL properties. In this paper, we describe a method for estimating the probabilities in a Discrete Time Markov Chain (DTMC) model of a large-scale system. We also provide a statistical test so that we can reject estimated DTMCs if the actual system does not have Markov behavior. We describe results of experiments applying our method to WSNs in an experimental test-bed, as well as using simulations. The results of our experiments suggest that our model estimation and model checking method provides a systematic, precise and easy way of evaluating performance metrics of some large-scale WSNs.

1 Introduction

A wireless sensor network (WSN) is a system of sensor nodes that collaborate with each other through wireless

communication. Each node has its own processor, volatile and nonvolatile memory, one or more sensors, a wireless communication channel, and an independent power source. Because of these unique characteristics, WSNs have been successfully used in many applications such as environmental monitoring [16], structural health monitoring [17], target tracking [2, 13], and so on.

One of the distinguishing characteristics of typical WSNs is their scale. For example, a medium scale application such as shooter localization [12] may involve 60 nodes. Without an appropriate abstraction, it is hard to reason about the global behavior of a WSN, for example to compute its aggregate energy consumption level or availability. If we were to model a system state as a cross-product of each node's state, the standard in concurrency theory, we would end up with the well known state explosion problem: a WSN of 100 nodes, each with three states, has 3^{100} states. Instead, we take a statistical approach: we abstract the global state of a system as a vector of probabilities, where the size of the vector is the number of states the nodes may be in, and the i^{th} element of the vector represents the probability that a randomly picked node is in the i^{th} state. With this abstraction we can easily reason about certain expected aggregate properties of the system. Such properties include availability, energy consumption, and throughput.

We model the transition dynamics between states as a *Discrete Time Markov Chain* (DTMC). This is reasonable for many applications running on WSNs as their behavior corresponds to a probabilistic transition systems: nodes go to a sleep mode to save energy with a certain probability and, in order to minimize collisions, nodes send packets based on probabilistic choice. We choose a DTMC as our model instead of a Continuous Time Markov Chain (CTMC) because the interpretation of time is discrete in our performance evaluation logic (see Section 4). The choice of DTMC also enables our method for estimation of the model (see Section 3.2).

A DTMC is defined by a *set of states* and *transition probabilities* between states. Identifying states from a real

*This research has been supported in part by the DARPA IXO NEST program under contract F33615-01-C-1907, by NSF under grant CNS 05-09321 and by ONR under DoD MURI award N0014-02-1-0715. The authors would like to acknowledge helpful comments and feedback from Timo Latvala.

system can be done rather straightforwardly: depending on the properties we want to evaluate or monitor, we can define a necessary set of states. A developer can easily identify which behavior of an application program belongs to which state. However, assigning transition probabilities between states is not as immediate as defining and identifying states. For example, although a process may go to a sleep mode by a pre-programmed probability, we generally do not know how often a process arrives at the relevant decision point in the code. Thus, we develop a DTMC estimation algorithm based on a sequence of state estimations of an application execution.

Recall that the current state of a Markov process depends only on its immediate past state. Thus, even samples from a single execution have enough previous-next state relations to *estimate* the Markov transition matrix. We also provide a statistical testing method that checks whether the sample is from the estimated DTMC. This test is important because our estimation algorithm would always find optimal parameters that best match with the samples, even if the system were *not* a Markov process. However, if the system is not a Markov process or the parameters are poorly estimated, our test will give users a notification together for any level of desired confidence.

Once we have a DTMC model of a WSN, we can check whether this model satisfies certain performance criteria that should be met. For example, we can evaluate the expected number of message retransmissions before a sender successfully receives an ack message, the expected amount of energy consumed to reliably send a message, or how long it may take to recover from a current unacceptable availability level.

The contributions of this paper are as follows. First, we describe a novel way of modeling sensor networks as DTMCs and expressing the aggregate properties of the network, both in its transient states and its steady state. Second, we provide a way of estimating DTMCs and suggest a statistical test to ensure with high probability that the sensor network obeys the Markov property and that the estimate is sufficiently accurate. Finally, we show a novel method for model checking these properties can be applied. We use both simulations and experimental results to illustrate and validate our method. Note that we do not describe or prove our model checking algorithm, as this has been done elsewhere [9]. Rather, this paper is concerned with the application of the model checking algorithm to sensor networks. Also note that standard methods for solving Markov matrices result in efficient solutions for steady state analysis. However, we capture both the transient behaviors and the steady state. Moreover, we express our properties in a formal logic and automate the process of checking behaviors.

It is also important to point out some limitations of our approach. The methodology we are proposing is useful for establishing aggregate expected properties only of sen-

sor networks where the nodes have a sufficient degree of symmetry in the local states, for example, if the nodes implement a randomized algorithm, or if the system is event driven and the environment exhibits probabilistic behavior over the time interval of interest. In a way, our approach is similar to statistical physics which allows aggregate properties of a system, such as the entropy or the average temperature, to be estimated. Our method is not applicable to reasoning about properties of systems where the modeling requires that the different nodes have a non-uniform probabilistic representation. For example, our method would not be applicable if, for the property of interest, the behavior of the nodes is represented as a tree of join continuations [1]. However, the goodness of fit for Markov model estimation we propose should tell us when our method is inapplicable.

The paper is organized as follows. The next section motivates our choice of a probabilistic temporal logic. Section 3 briefly explains DTMCs and the notations used in the paper. In this section we propose a DTMC estimation method and a statistical testing method. Section 4 describes the syntax and informal semantics of iLTL. Section 5 experimentally demonstrates the effectiveness of our proposed approaches on a WSN of 90 Mica2 nodes. In the final section, we discuss how the applicability of our method may be extended to some WSNs where not all nodes have the same probabilities over the states of interest.

2 Probabilistic Temporal Logics

A number of probabilistic temporal logics such as PCTL, PCTL*, and CSL can be used to reason about state transitions of Markov chains [7, 3, 4, 8]. With these logics one can specify the probability that a Markov process follows certain specified paths. For example, in a multi-path communication network, we can express the property “with probability larger than 0.5 a packet will arrive at its destination with path length less than 5.” These logics are described on a probability space built on a set of computational paths. That is, in the multi-path example, the sample space is a set of all routing paths and the event is a set of paths whose routing length is less than 5 starting from a certain state.

A model checking algorithm for the above logics must repeatedly compute such probabilities from each state for each sub-formula of a specification, until the probability for the whole specification from the initial state is computed. Although these logics are good for reasoning about transitions of a single process, they are not suitable for DTMCs as abstractions for large scale systems. This is largely due to the fact that they are reasoning on the probability space of computational paths instead of reasoning directly on transitions of the *probability mass function* (pmf).

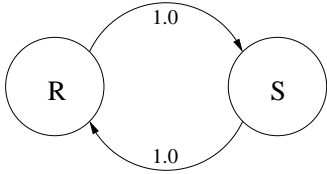


Figure 1. An oscillating DTMC

We illustrate the problem with a DTMC (Figure 1). This DTMC has two states: Run (R) and Sleep (S). The transition probabilities for each state to the other is 1. That is, a node always makes a transition to the other state. Hence, from a single process point of view, there is no consecutive R or consecutive S states in any computational paths. So, the probability that a specification “always R ” is satisfied is 0 and PCTL-like logics captures this properly. However, suppose that there are 100 processes and initially 50 of them are in state R and the others are in state S . Then, there are always 50 processes in one of the two states and a specification like “always $P[R] \geq 0.5$ ” should be true. If all 100 processes are in state R or in state S then the previous specification should be false by the same reason as “always R ”. For this system, because $P[R] \geq 0.5$ is false in S state, PCTL-like logics compute the probability of satisfying the specification to be 0, regardless of the initial condition. Thus, the semantics of PCTL-like logics are not suitable to express properties on transitions of pmf.

In order to address this problem, we have designed a logic called *iLTL* and implemented a model checker *iLTLChecker* [9]. *iLTL* is a *Linear Temporal Logic* (LTL) [10] whose atomic propositions are linear inequalities about pmfs. Those inequalities can be thought of as inequalities about expected “rewards” such as the expected energy consumption or the availability of a system. Instead of working on a probability space of computational paths, *iLTL* specifies properties based on the pmf transitions. The model checker for *iLTL* searches for an initial pmf for which trailing transitions of expected rewards violate the specification. This addresses the problem described above. Furthermore, because *iLTL* model checker looks for an initial pmf that may lead to violation of specification, it can be used as a predictive monitoring tool. For example we can specify the property $(0.1 < P[Ready] \wedge P[Ready] < 0.2) \rightarrow X X X \square (P[Ready] > 0.5)$: if the current interval estimate of the availability of the system is 10% to 20%, three steps later ($X X X$) the availability of the system always (\square) becomes larger than 50%.

3 Markov Model

Recall that a DTMC is finite state automation with transition probabilities between the states. As discussed earlier, we assume that identifying the states of interest from

application programs is straightforward. In this section, we propose a Markov transition matrix estimation method based on *Quadratic Programming* (QP) [11]. We also provide a statistical test so that we can discard the estimated DTMC model if the real system does not exhibit Markov behavior.

3.1 Discrete Time Markov Chain

A Markov process is characterized by its memoryless property: the future probability transitions depend only on their current probability distribution regardless of their past history. In other words, the current state encodes the relevant history of a Markov process for the purpose of determine the transition probability. A Markov chain is a Markov process with a countable number of states [14]. We represent a Discrete Time Markov Chain (DTMC) X as a tuple (S, \mathbf{M}) , where S is a finite set of states $S = \{s_1, s_2, \dots, s_n\}$ that X can take and \mathbf{M} is a Markov transition matrix that governs the transitions of X 's probability mass function (pmf). We also use a column vector $\mathbf{x}(t) = [x_1(t), x_2(t), \dots, x_n(t)]^T$ to represent the pmf of X at time t such that $x_i(t) = P[X(t) = s_i]$. Thus,

$$\mathbf{x}(t+1) = \mathbf{M} \cdot \mathbf{x}(t).$$

Since \mathbf{M} is a probability transition matrix, each element is non-negative and each column sums up to be one. That is, the sum of outgoing probabilities from a state, including a self loop to the state itself, is 1. These two conditions are the constraints that should be met when estimating a Markov matrix.

A Markov process is a deterministic system in the sense that once the initial pmf $\mathbf{x}(0)$ is given, the rest of the transitions $\mathbf{x}(t)$ for $t \geq 1$ are uniquely determined by the Markov matrix \mathbf{M} : $\mathbf{x}(t) = \mathbf{M}^t \cdot \mathbf{x}(0)$. That is, each initial pmf completely determines a computational path. Our specification logic *iLTL* (Section 4) specifies temporal behaviors on uncountably many computational paths; our model checking tool *iLTLChecker* searches for an initial pmf whose trailing pmf transitions will violate the specification.

3.2 DTMC Model Estimation

We now describe a Markov transition matrix estimation method from a medium to large scale WSN. We assume that the property of interest is over states satisfying the Markov property, and that every node of the WSN is independent and identical. These assumptions are valid in many WSN applications: a class of such applications is systems where each node makes a probabilistic choice to change its state and in many cases this decision is made independent of the states of the other nodes. The ability to make such independent decision saves large amounts of energy because synchronization requires information exchange among nodes. For example, in many applications,

a sensor node often goes to a sleep mode with certain probability regardless the states of other nodes. However, we can expect with high confidence that at least certain number of nodes are awake.

In medium to large scale WSNs there are hundreds of nodes which provide a sufficiently large population for robust pmf estimation. If we estimate the pmfs periodically, we can estimate the underlying Markov transition matrix by comparing two consecutive estimations. In the experimental case study described in Section 5, each time-synchronized node periodically stores its state in its memory. We subsequently collect these sequences from all nodes and estimate a sequence of pmfs. Note that this state sampling code can be used once for Markov chain estimation and then disabled so that it does not affect the performance of a system.

Let $X = (S, \mathbf{M})$ be the true Markov process of the system. We compute an estimated Markov chain $(S, \hat{\mathbf{M}})$ such that $\hat{\mathbf{M}}$ minimizes a squared sum of differences between a one step predicted pmf $\hat{\mathbf{x}}(t+1)$ and a sampled pmf $\bar{\mathbf{x}}(t+1)$. We estimate the probability $x_i(t) = \text{P}[X(t) = s_i]$ by the fraction of nodes in state s_i over the whole population size. We describe this process in more detail in order to drive a goodness of fit test. We first introduce a one-zero *Random Variable* (RV) Y_i for each state s_i such that $Y_i(X(t)) = 1$ if $X(t) = s_i$ and 0 otherwise. Let ns be the population size at each sample, and let an RV $K_i(t)$ be

$$K_i(t) = \sum_{t=1}^{ns} Y_i(X(t)).$$

Then $K_i(t)$ has a binomial distribution: $\text{P}[K_i(t) \leq n] = \text{Bin}(n, ns, x_i(t))$, where

$$\text{Bin}(n, m, p) = \sum_{i=0}^n \binom{m}{i} \cdot p^i \cdot (1-p)^{m-i}.$$

We use a normalized frequency $\bar{x}_i(t) = K_i(t)/ns$ as our point estimator for $x_i(t)$. Since each sample at a given point in time is independent, by the Central Limit Theorem (CLT) for large ns our point estimator $\bar{x}_i(t)$ has a normal distribution $N(\mu = x_i(t), \sigma = \sqrt{x_i(t)(1-x_i(t))/ns})$ [14].

Assume that we have m pmf samples of an n state Markov process and let $\mathbf{P} \in \mathbb{R}^{m \times n}$ be a matrix of these point estimates: $P_{ij} = \bar{x}_j(i)$. We estimate a Markov matrix $\hat{\mathbf{M}} \in \mathbb{R}^{m \times n}$ such that it minimizes the differences between a sampled pmf $\bar{\mathbf{x}}(t+1)$ and a one step predicted pmf $\hat{\mathbf{x}}(t+1 | t) = \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t)$:

$$E = \min_{\hat{\mathbf{M}}} \sum_{t=1}^{m-1} \left| \bar{\mathbf{x}}(t+1) - \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t) \right|^2.$$

Let \mathbf{P}_c and \mathbf{P}_f be \mathbf{P} 's sub-matrices of the first and the last $m-1$ rows. Then the matrix $\hat{\mathbf{M}}$ that minimizes E is:

$$\hat{\mathbf{M}} = (\mathbf{P}_c^T \cdot \mathbf{P}_c)^{-1} \cdot \mathbf{P}_c^T \cdot \mathbf{P}_f.$$

However, since $\hat{\mathbf{M}}$ is an estimate of a Markov matrix, there are constraints on $\hat{\mathbf{M}}$. These are $0 \leq \hat{M}_{ij} \leq 1$ for all $1 \leq i, j \leq n$ and $\sum_{i=1}^n \hat{M}_{ij} = 1$ for all $1 \leq j \leq n$. We can minimize E subject to those constraints by Quadratic Programming [11].

Let $\mathbf{z} \in \mathbb{R}^{n^2 \times 1} = [\hat{\mathbf{M}}_{*1}^T, \hat{\mathbf{M}}_{*2}^T, \dots, \hat{\mathbf{M}}_{*n}^T]^T$ and

$$\begin{aligned} \mathbf{H} \in \mathbb{R}^{n(m-1) \times n^2} &= \begin{bmatrix} \mathbf{P}_c & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{P}_c \end{bmatrix}, \\ \mathbf{f} \in \mathbb{R}^{n(m-1) \times 1} &= [\mathbf{P}_{f*1}^T, \dots, \mathbf{P}_{f*n}^T]^T, \\ \mathbf{A} \in \mathbb{R}^{2n^2 \times n^2} &= [\mathbf{I}_{n^2}, -\mathbf{I}_{n^2}]^T, \\ \mathbf{b} \in \mathbb{R}^{2n^2 \times 1} &= [\mathbf{1}_{1,n^2}, \mathbf{0}_{1,n^2}]^T, \\ \mathbf{C} \in \mathbb{R}^{n \times n^2} &= \begin{bmatrix} \mathbf{1}_{1,n} & \cdots & \mathbf{0}_{1,n} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{1,n} & \cdots & \mathbf{1}_{1,n} \end{bmatrix}, \\ \mathbf{d} \in \mathbb{R}^{n \times 1} &= \mathbf{1}_{n,1}, \end{aligned}$$

where \mathbf{I}_n is a $n \times n$ identity matrix, $\mathbf{1}_{nm}$ is a $n \times m$ matrix of ones, $\mathbf{0}_{nm}$ is a $n \times m$ matrix of zeros, and $\hat{\mathbf{M}}_{*i}$ and \mathbf{P}_{f*i} are the i^{th} columns of $\hat{\mathbf{M}}$ and \mathbf{P}_f . Then $\hat{\mathbf{M}}$ can be obtained by

$$\begin{aligned} \min_{\mathbf{z}} \quad & 0.5\mathbf{z}^T \mathbf{H}^T \mathbf{H} \mathbf{z} - \mathbf{f}^T \mathbf{H} \mathbf{z} \\ \text{subject to} \quad & \\ & \mathbf{A} \mathbf{z} \leq \mathbf{b}, \\ & \mathbf{C} \mathbf{z} = \mathbf{d} \end{aligned}$$

3.3 Goodness of Fit Test for Estimated DTMCs

Although, we can always define a set of states and estimate transition probabilities between the states through the estimation algorithm described in Section 3.2, we still do not know whether the real system has the Markov property. In this section, we propose a goodness of fit test against the estimated model. This test method statistically rejects the estimated model with a given level of confidence if the sequence of pmf samples does not agree with the model.

For this test, we first build a null hypothesis and an alternative hypothesis:

H_0 : The actual process is a Markov chain whose transition matrix is the same as the estimated matrix.

H_a : The actual process is not a Markov chain or the Markov transition matrix is different from the estimated one.

Under the null hypothesis H_0 , we can compute one step predicted pmf using a current sampled pmf. Using a χ^2 test between the predicted pmf and a sampled next step pmf, we can compare how well they fit together. More

specifically, under the null hypothesis, the RV $q(t)$ has χ^2 distribution with $n - 1$ degrees of freedom:

$$q(t) = \sum_{i=1}^n \frac{(K_i(t) - ns \cdot \hat{x}_i(t | t-1))^2}{ns \cdot \hat{x}_i(t | t-1)},$$

where $K_i(t)$ is the RV we defined in Section 3.2 and $\hat{\mathbf{x}}(t | t-1) = \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t-1)$ is a one step predicted pmf from the previous sample $\bar{\mathbf{x}}(t-1)$. We omit the degrees of freedom in χ^2 distribution for simplicity in notation. Thus, unless otherwise specified, χ^2 means χ^2 with $n - 1$ degrees of freedom. We write χ_α^2 for the value y such that $\chi^2(y) = \alpha$.

Now, we introduce a significance level γ , and a random variable B_γ such that

$$B_\gamma = \sum_{t=2}^m \delta_{\chi_{1-\gamma}^2}(q(t)),$$

where $\delta_\theta(x) = 1$ if $x \geq \theta$ and 0 otherwise. Note that, under the null hypothesis H_0 , for the probability estimation at time t , $P[q(t) > \chi_{1-\gamma}^2] = \gamma$ and the one-zero random variable $\delta_{\chi_{1-\gamma}^2}(q(t))$ has an expected value γ which is the probability of bad sampling under H_0 . Since, there are m samples B_γ has a binomial distribution of order $m - 1$ and probability γ . Let n_γ be a least integer such that $\text{Bin}(n_\gamma, m - 1, \gamma) \geq 1 - \alpha$. We accept H_0 if for all $\gamma \in (0, 1)$, $B_\gamma \leq n_\gamma$. Otherwise we reject H_0 with the probability of type I error α .

The problem is that γ is in \mathbb{R} , which we cannot enumerate. However, because B_γ takes finite number of values, we can still perform the goodness of fit test. Let s be the index of $q(t)$ when $q(t)$'s are sorted in the increasing order and let γ_s be $1 - \chi^2(q(t))$. Note that B_{γ_s} is equal to $m - s$. We divide the interval for γ into several sub-intervals and do test on each: $(1, \gamma_1]$, $(\gamma_s, \gamma_{s+1}]$ for $1 \leq s \leq m - 2$, and $(\gamma_{m-1}, 0)$. Because n_γ is $m - 1$ and 0 for the intervals $1 < \gamma \leq \gamma_1$ and $\gamma_{m-1} < \gamma < 0$ respectively and so is B_γ , the test is trivially true in these intervals. Thus, $B_\gamma \leq n_\gamma$ for $0 < \gamma < 1$ if and only if $n_{\gamma_s} < m - s$ for $1 \leq s \leq m - 1$.

3.4 Example

We illustrate the accuracy and usefulness of our DTMC estimation algorithm and goodness of fit test by means of a simulation study. Specifically, we model a set of processes running on WSN nodes by a three state automaton with *Ready*, *Run*, *Wait* states. When a process is initiated, it is in a *Ready* state. It then transitions to a *Run* state. If the process performs an I/O operation, it will transition to a *Wait* state, and after the I/O operation is completed, it will return to the *Ready* state. A process in the *Run* state moves back to the *Ready* state so that other processes running on the same node can proceed together. We assume that the process does not terminate; this is usual in WSN applications—typically, a WSN application monitors the environment and reports the result to a base station

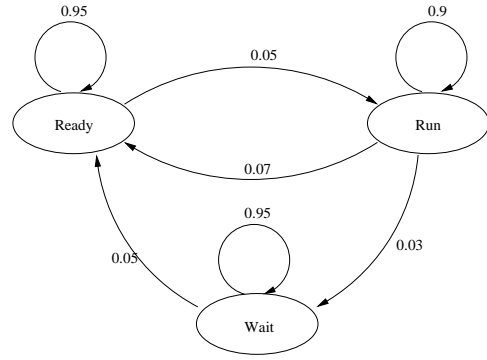


Figure 2. Process state diagram

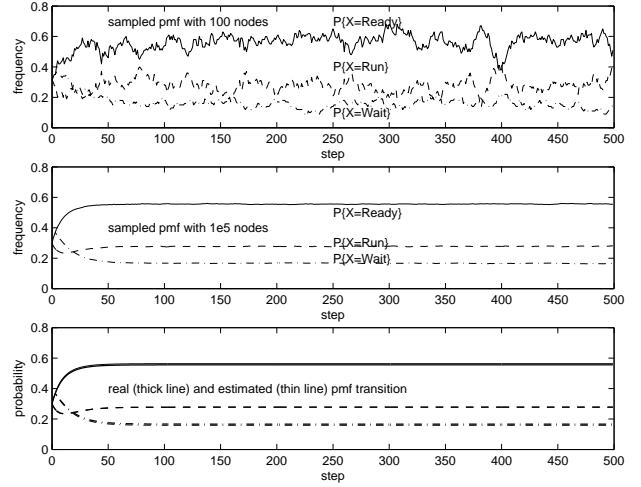


Figure 3. top: pmf samples of 100 node simulation, middle: pmf samples of 10^5 node simulation, bottom: pmf transitions of a real and an estimated Markov chains.

through the life of its platform. Assume that the system is a DTMC $PS = \{S, \mathbf{M}\}$ where $S = \{Ready, Run, Wait\}$ and

$$\mathbf{M} = \begin{bmatrix} 0.95 & 0.07 & 0.05 \\ 0.05 & 0.90 & 0.0 \\ 0.0 & 0.03 & 0.95 \end{bmatrix}.$$

Figure 2 shows a block diagram of the DTMC PS ; the ellipses represent states and the numbers represent transition probabilities.

We ran a simulation with 100 independent and identical DTMC nodes of PS . Beginning from a state S , each node changes its state based on the transition probability matrix \mathbf{M} at each step. We measured the frequencies of each state at each step and normalized these frequencies so that the sum of the fractions was 1. Recall that these normalized frequencies are the point estimates of the probabilities.

The first graph of Figure 3 shows the sampled pmf transitions of the 100 nodes. Note that, there are large jitters

in the graph. These jitters are due to the small sample population size. Based on the CLT, we hypothesize that the variance of the jitters will be inversely proportional to the number of nodes. The number of states, which is the degrees of freedom of $q(t)$ plus one, will relate inversely to the jitters in the observations. In order to confirm the cause of the jitter, we ran the same simulation again with 10^5 nodes. The second graph of Figure 3 shows the sampled pmf transitions. One can see that most of the jitters of the first graph have been removed. The thick lines of the third graph are the computed pmf transitions of the real system PS . We can see that the sampled pmf transitions of 10^5 nodes looks very close to the computed pmf transitions.

From the simulation results of 100 nodes, we estimate a DTMC $\hat{PS} = (S, \hat{\mathbf{M}})$, where

$$\hat{\mathbf{M}} = \begin{bmatrix} 0.9489 & 0.0744 & 0.0512 \\ 0.0511 & 0.8963 & 0.0000 \\ 0.0000 & 0.0293 & 0.9488 \end{bmatrix}.$$

We present a computed pmf transitions of \hat{PS} as thin lines of the last graph of Figure 3. We can roughly see how close the estimated responses are to the real ones. The estimated model passes the goodness of fit test with a significance level of 0.05.

In order to demonstrate the effectiveness of our goodness of fit test, we slightly modify the first column of \mathbf{M} as follows and run the same simulation with 100 nodes with a DTMC $PS' = (S, \mathbf{M}')$, where

$$\mathbf{M}' = \begin{bmatrix} 0.95 & 0.07 & 0.05 \\ 0.01 & 0.90 & 0.0 \\ 0.04 & 0.03 & 0.95 \end{bmatrix}.$$

The first graph in Figure 4 shows how the sampled pmf transits with the modified Markov transition matrix \mathbf{M}' . Comparing with the first graph of Figure 3, the probability $P[PS = Wait]$ is increased and $P[PS = Run]$ is decreased. The second and the third graphs in Figure 4 show Euclidean distances between sampled pmfs and one step prediction by \hat{PS} for the simulations of Figure 3 and Figure 4. That is, $|\bar{\mathbf{x}}(t+1) - \hat{\mathbf{x}}(t+1 | t)|$ and $|\bar{\mathbf{x}}'(t+1) - \hat{\mathbf{x}}(t+1 | t)|$ where, $\bar{\mathbf{x}}(t)$ is a sampled pmf vector of PS at step t , $\bar{\mathbf{x}}'(t)$ are that of PS' and $\hat{\mathbf{x}}(t+1 | t) = \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t)$. As expected, the prediction error of the second graph is less than that of the third graph.

We checked the estimated DTMC \hat{PS} against the simulation results of modified DTMC PS' . When $\gamma = 0.0264$, there are 492 samples out of 499 that do not satisfy $q < \chi_{0.9736}^2$. With the significance level of 0.05 and $\gamma = 0.0264$, the confidence interval is [0,491]. Thus, we reject the null hypothesis H_0 with the probability of type I error 0.1 and accept H_a .

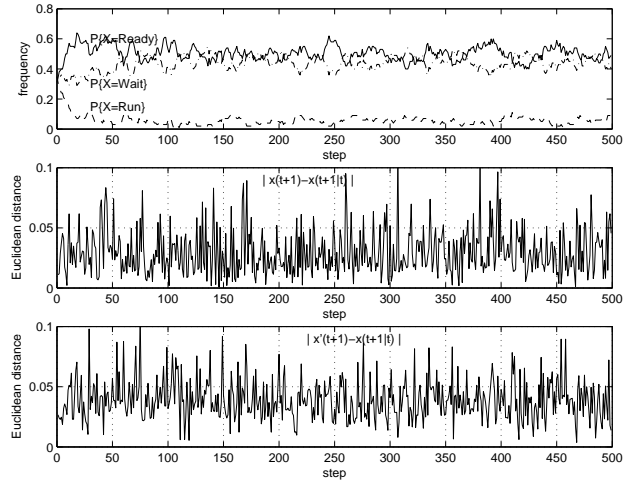


Figure 4. top: pmf samples of 100 node simulation with modified Markov chain PS' , middle: difference between expected pmf (\hat{PS}) and sampled pmf (PS), bottom: difference between expected pmf (\hat{PS}) and sampled pmf (PS').

4 Specification Logic

The syntax of probabilistic temporal logic iLTL is as follows:

$$\begin{aligned} \psi &::= T \mid F \mid ineq \mid \\ &\quad \neg\psi \mid \psi \vee \phi \mid \psi \wedge \phi \mid \\ &\quad X\psi \mid \psi \text{ U } \phi \mid \psi \text{ R } \phi \\ ineq &::= \sum_{i=1}^n a_i \cdot P[X = s_i] < b, \end{aligned}$$

where $X = (\{s_1, \dots, s_n\}, \mathbf{M})$, $a_i \in \mathbb{R}$ and $b \in \mathbb{R}$. As usual, \square and \diamond are defined as follows:

$$\begin{aligned} \square\psi &\equiv FR\psi, \\ \diamond\psi &\equiv TU\psi. \end{aligned}$$

In order to explain the semantics of an iLTL formula we first define a computational path of a DTMC $X = (\{s_1, \dots, s_n\}, \mathbf{M})$ by a function $\sigma_x : \mathbb{N}^+ \rightarrow \mathbb{R}^n$ such that $\sigma_x(t) = \mathbf{M}' \cdot \mathbf{x}$. The semantics of an iLTL formula should be interpreted over these paths. The atomic propositions of iLTL are in the form of linear inequalities (*ineq*) over a pmf. The linear inequalities can be regarded as inequalities about expected rewards: we associate a constant reward with each state of a Markov chain and when a process visits a state it earns the reward associated with that state. The expected reward earned at a given point in time is the sum of the products of the rewards at each state and the probability that a process is in that state. Formally, a *Markov Reward Process* (MRP) is a triple (S, \mathbf{M}, ρ) where (S, \mathbf{M}) is a DTMC and $\rho : S \rightarrow \mathbb{R}$ is a reward function for each

state [5]. Then the expected reward of a MRP (S, \mathbf{M}, ρ) at time t is

$$\sum_{s_i \in S} \rho(s_i) \cdot \mathbb{P}[X(t) = s_i].$$

Thus, a linear inequality over a pmf is an inequality about expected reward: replace α_i with $\rho(s_i)$ and ignore the time index t . The time index is implicitly given from the temporal operators of iLTL: the pmf of *ineq* is replaced by $\sigma_{\mathbf{x}}(0)$.

Many useful properties of a system can be expressed as inequalities about expected rewards. For example, a comparison between two probabilities “ $\mathbb{P}[X = A] > \mathbb{P}[X = B]$ ” can be rewritten in a reward form as “ $1 \cdot \mathbb{P}[X = A] - 1 \cdot \mathbb{P}[X = B] > 0$ ”, where 1 and -1 are rewards in states A and B . An expected value of meaningful quantities such as expected queue length can also be expressed in reward form: “ $1 \cdot \mathbb{P}[Q = Q_1] + \dots + n \cdot \mathbb{P}[Q = Q_n] < L$ ” specifies that the expected queue length of a queuing system of capacity n is less than L . We can also compare two probabilities at different time in the reward form: suppose that $X = (\{s_1, \dots, s_n\}, \mathbf{M})$ is a DTMC then

$$\begin{aligned} \mathbb{P}[X(t) = s_i] > \mathbb{P}[X(t+k) = s_j] \text{ iff} \\ \mathbb{P}[X(t) = s_i] - \sum_{j=1}^n m_j \cdot \mathbb{P}[X(t) = s_j] > 0, \end{aligned}$$

where $[m_1, \dots, m_n]$ is the j^{th} row of \mathbf{M}^k . One of the reasons iLTL does not explicitly use time-indexes is because it can be written in a reward form as above.

The meaning of logical connectives \wedge , \vee , and \neg are usual: $\psi \wedge \phi$ is true if and only if ψ and ϕ are both true, $\psi \vee \phi$ is true if and only if ψ or ϕ are true, and $\neg\psi$ is true if and only if ψ is false.

The semantics of the temporal operators X , U , and R are explicitly related to a computational path $\sigma_{\mathbf{x}}$. $X\psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ is true in $\sigma_{\sigma_{\mathbf{x}}(1)}$. $\psi U \phi$ in $\sigma_{\mathbf{x}}$ is true if and only if ϕ eventually becomes true along the path of $\sigma_{\mathbf{x}}$ and while ϕ is false ψ is true in $\sigma_{\mathbf{x}}$. In other words, $\psi U \phi$ in $\sigma_{\mathbf{x}}$ is true if and only if there is $t \geq 0$ such that ϕ is true in $\sigma_{\sigma_{\mathbf{x}}(t)}$ and for all $s \in [0, t-1]$, ψ is true in $\sigma_{\sigma_{\mathbf{x}}(s)}$. The until operator (U) can be easily explained by an example: suppose that events are happening in a WSN and one may want to keep the WSN in an alert mode until most of the nodes detect them. Then the specification: $(\mathbb{P}[X = \text{Ready}] > 0.5) U (\mathbb{P}[X = \text{Detect}] > 0.9)$ checks whether the event will be detected by more than 90% of the nodes and until that moment more than 50% of the nodes are in ready state. The *release* operator R is a dual of U operator. $\psi R \phi$ in $\sigma_{\mathbf{x}}$ is true if and only if ϕ is true in $\sigma_{\mathbf{x}}$ while ψ is false. The *eventually* operator $\diamond\psi \equiv T U \psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ eventually become true in $\sigma_{\mathbf{x}}$ and the *always* operator $\square\psi \equiv F R \psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ is always true in $\sigma_{\mathbf{x}}$.

Finally, we can define a satisfaction relation \models between an iLTL formula and a DTMC. We say that a DTMC A is a *model* of an iLTL formula ψ and write $A \models \psi$ if and only if for all pmf transitions $\sigma_{\mathbf{x}}$ of A ψ is true in $\sigma_{\mathbf{x}}$. Note

	Processor	Radio	
		transmit	receive
Active	8 mA	25 mA	8 mA
Sleep	< 15 μ A	< 1 μ A	

Table 1. Energy consumption level of a Mica-2 mote.

that each initial pmf \mathbf{x} determines a computational path $\sigma_{\mathbf{x}}$. Thus, there are uncountably many computational paths.

Our model checker *iLTLChecker* [9] looks for a computational path in which the negation of an original specification is true. That is, given a specification ψ *iLTLChecker* looks for an initial pmf \mathbf{x} such that $\neg\psi$ is true in $\sigma_{\mathbf{x}}$.

5 Experiments

We now describe a case study on a WSN platform. Our experimental platform comprises of 90 Mica-2 motes. Each mote has an ATmega128L low power 8 bit CPU working at 8MHz clock speed, 4Kbyte of SRAM, 128Kbyte of program flash, and 512Kbyte of serial flash memory. Although, a mote has relatively large serial flash memory, this memory is slow especially for write operation. Mica-2 is also equipped with a CC1000 radio transceiver. At the lower level communication layer, the Manchester encoding is used, and we can achieve a theoretical throughput of 38.4Kbps. In order to save energy, an application can go to a sleep mode. The amount of energy spent in various modes is summarized in Table 1 [6].

TinyOS is an operating system running on Mica-2 nodes. When developing an application, TinyOS provides a programming framework and library functions to support I/O operations. TinyOS is linked with application codes and loaded on Mica-2 nodes. TinyOS has three different program blocks: I/O operations are made by calling a *command* block, the result of an I/O operation can be passed to an application through an *event* block in the form of a signal, and application programs that run a long time should be written in a *task* block. TinyOS schedules tasks by managing a queue of non-preemptable task blocks. Thus, spin-looping in a task block will block the entire operations of TinyOS.

5.1 Experimental program

We implemented the program in Figure 5. This application program samples a microphone and stores the result in a buffer (sampling code not shown). Whenever the run task is scheduled, it computes a Fourier transform coefficient of the mic sample. After the coefficient is computed, in order to reduce collisions, the application sends the result to a base station with a .05 probability. To make the

```

task run() {
    state=Run;
    DFT(mic);
    if(rand()%100<5)
        call SendMsg.send(),
        state=Wait;
    else
        post run(),
        state=Ready;
}
task dummy() {
    int i,n=rand()%4;
    for(i=0;i<n;i++)
        DFT(mic);
    post dummy();
}
}

event SendMsg.sendDone(){
    post run();
    state=Ready;
}
double DFT(int* smp) {
    s=c=0;
    for(i=0;i<T;i++)
        s+=sinV[i]*smp[i],
        c+=cosV[i]*smp[i];
    return sqrt(s*s+c*c);
}

```

Figure 5. Abbreviated experimental program

experiment more realistic, we created a *dummy* task which simulates other applications running on the same node. We track the states of a process by recording them in the variable *state*. A timer interrupt routine is called at every 1/256 sec and samples the *state* variable. The state sample data is recorded on the SRAM because the serial flash is too slow and would affect the sampling. In view of the small SRAM, we compress the sample data by a run length encoding algorithm [15].

As one can see from the \hat{M} matrix (Section 5.2), when a process is in the *Wait* state it remains in this state with 95% probability which gives a high compression ratio to the encoding algorithm. Note that as sampling speed increases so does the probability that a process remains in its current state. That means, the run length encoding algorithm performs better at high frequency sampling. For this experiment we configure every node within a radio communication range of a base station. Because we do not need multi-hop message forwarding, we turn off the radio communication channel except when a node is sending a message. This saves energy which might be consumed by unnecessary messages.

5.2 DTMC Estimation

The 90 nodes were programmed identically except for their ids; the ids are used as seed values for a random number generator. All nodes are initially time synchronized by a start message from a base station: on receiving the start message, each node starts executing the code of Figure 5. A process state is sampled and compressed at the rate of 256 times per sec. We connected the sampling routine directly to a timer interrupt so that sampling accuracy is not compromised by the task scheduling of TinyOS. The sample data is retrieved from each node one by one. We aligned the 90 sequences of samples so that they begin at

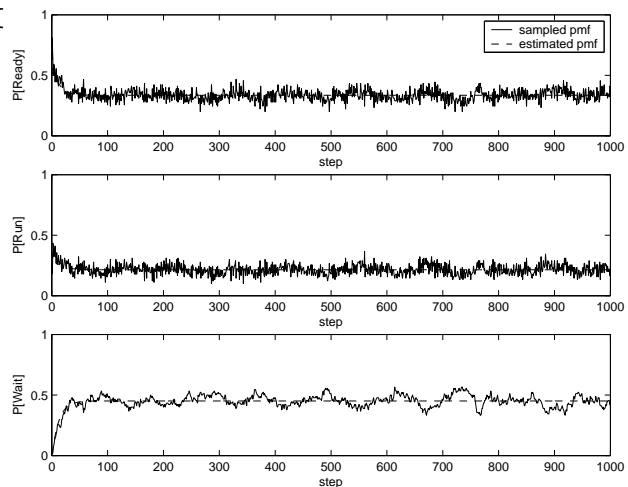


Figure 6. Sampled pmf transitions and pmf transitions of an estimated Markov chain.

the same moment and compute a sequence of estimated pmfs.

The solid lines of Figure 6 show the transitions of sampled probability distribution. As we have seen in Figure 3, the jitters are due to small sample population size. From this sequence of pmf estimations, we estimate a DTMC $A = (S, \hat{M})$, where $S = \{Ready, Run, Wait\}$ and

$$\hat{M} = \begin{bmatrix} 0.4691 & 0.7383 & 0.0435 \\ 0.4827 & 0.2455 & 0.0000 \\ 0.0482 & 0.0162 & 0.9565 \end{bmatrix}.$$

The dashed lines of Figure 6 show pmf transitions of the estimated DTMC A . We plotted the estimated pmf transition graphs starting from the initial sample pmf. We applied the goodness of fit test of Section 3.3 to the estimated DTMC A and the pmf samples. Our estimated model passed the test with a significance level of 0.05.

5.3 Performance Evaluation

Figure 7 shows an iLTLChecker description of the estimated DTMC A and an iLTL specification. The iLTLChecker has two main blocks: a model block and a specification block. The model block describes a DTMC model to be checked. This block begins with the name of the DTMC (A in this case) followed by a set of states the DTMC has and finally a Markov transition matrix. The specification block begins with an optional list of inequalities on expected rewards that will be used in iLTL specifications. Finally, an iLTL specification is written using the list of inequalities as its atomic propositions. In Figure 7, the inequalities a and b describe whether the availability of DTMC A is larger than 0.5 and 0.3 respectively. Similarly, the inequality c specifies that the probability that a process is in the *Wait* state is larger than 0.27.

```

model:
  Markov chain A
  has states:
    { Ready, Run, Wait},
  transits by :
    [ .4691, .7383, .0435;
      .4827, .2455, .0000;
      .0482, .0162, .9565 ]

specification:
  a : P[A=Ready] > 0.5,
  b : P[A=Ready] > 0.3,
  c : P[A=Wait] > 0.27,
  d : 8*P[A=Ready] + 8*P[A=Run]
      + 33*P[A=Wait] < 25,
  e : 8*P[A=Ready] + 8*P[A=Run]
      + 33*P[A=Wait] > 15

  a -> X X [] b      # 1.
  #<> [] (d /\ e)    # 2.
  #(~e) U c          # 3.

```

Figure 7. An iLTLChecker description of the estimated DTMC model and specifications.

Inequalities d and e are about expected energy consumption levels. From Table 1 we know that if a process is in the Run state or in the Ready state it consumes 8 mA of energy and if it is in the Wait state (sending messages) it consumes 33 mA (25 for radio + 8 for process) of energy. Thus, e specifies that the expected energy consumption level of a node is larger than 15 mA.

The first iLTL formula $a \rightarrow X X [] b$ specifies that if the availability of the system is larger than 50% then from two steps onward the availability never goes below 30%. The model checking result is true: iLTLChecker shows the following result.

```

Depth: 40
Result: T

```

The first line **Depth:40** is the maximum number of time step iLTL needs to search in order to check the specification. iLTLChecker shows the depth first before it begins search. This number gives a hint about the model checking time: if it is too large, one may need to change the specification. The second line **Result:T** means that the DTMC is a model of the specification. In other words, all pmf transitions of the DTMC A satisfy the specification.

If we slightly modify the specification to $a \rightarrow X [] b$ or replace the RHS of the inequality a with 0.45, the DTMC A does not satisfy the specifications: availability of 50% now does not guarantee the minimum availability of 30% from the next step, and an availability of 45% now does not guarantee a minimum availability of 30% beginning from two steps later. The following is the out-

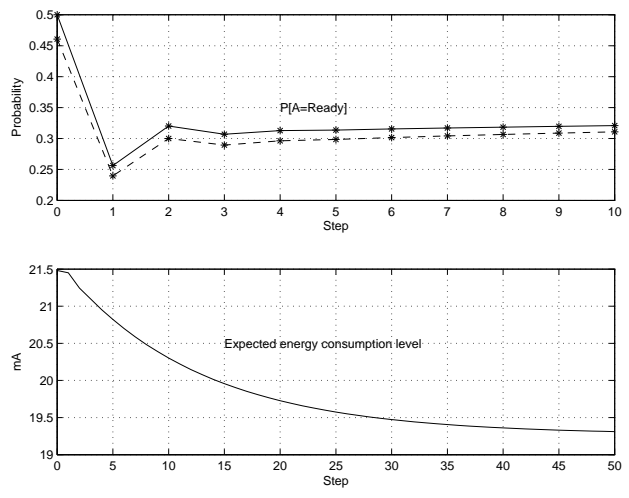


Figure 8. Availability (top) and expected energy consumption level (bottom) of A.

put of iLTLChecker for the specification “ $P[A = Ready] > 0.45 \rightarrow X X \square (P[A = ready] > 0.3)$ ”:

```

Depth: 40
Result: F
counterexample:
  pmf(A(0)): [ 0.461 0.000 0.539]

```

The second line indicates that the DTMC does not satisfy the specification and the 4th line shows a counter example: an initial pmf for the DTMC A that leads to a violation of the specification.

We can check the result from the first graph of Figure 8. This graph shows how the availability of the system is changing over time from two different initial pmfs: $[0.5, 0, 0.5]^T$ for the solid graph and $[0.461, 0, 0.539]^T$ for the dashed graph. From the red graph, we can see an example run that satisfies $a \rightarrow X X [] b$ but does not satisfy $a \rightarrow X [] b$: the availability is less than 0.3 at step 1 but it is always larger than 0.3 from step 2. The green line shows a counter example of the modified specification: the availability is less than 0.3 at step 3 and step 4.

The second commented formula $\langle \rangle [] (d \wedge e)$ specifies that in the steady state the expected energy consumption level is in between 15 mA and 25 mA. From an eigenvalue analysis, we can compute that the steady state pmf of A is $\mathbf{x}(\infty) = [0.34, 0.21, 0.45]^T$ and the expected energy consumption level in the steady state is 19.2 mA. Thus, we expect the model checking result to be true; indeed, iLTLChecker returns true. The second graph of Figure 8 shows the transition of expected energy consumption level starting from the counter example of the previous paragraph.

The third commented specification $(\neg e) U c$ describes that the probability $P[A = Wait]$ eventually be-

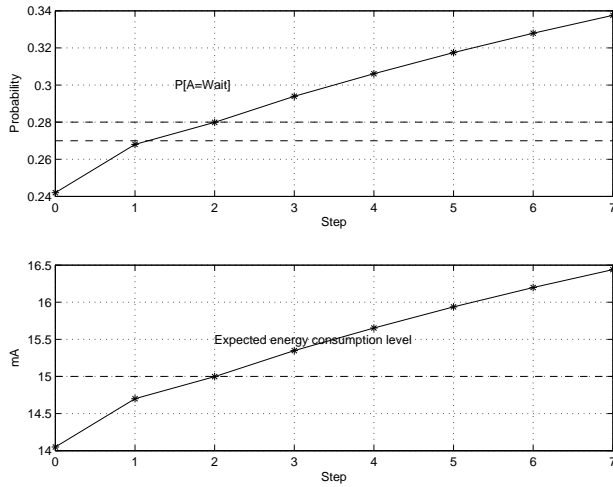


Figure 9. Expected energy consumption level and $P[A = Wait]$.

comes larger than 0.27 and while it is less than 0.27 the expected energy consumption level is less than 15 mA. iLTLChecker returns true for this statement. However, if we change the RHS of the inequality c to 0.28 it returns false with a counter example of $[0.72, 0, 0.28]^T$. We can immediately check the result at step 0: $P[A = Wait] > 0.28$ is false and the expected energy consumption level is 15 mA. In order to check the transitions overtime we draw Figure 9 from an initial pmf $[0.758, 0, 0.242]^T$. From the first graph of Figure 9, we can see that $P[A(t) = wait] > 0.27$ is true from step 2 onward and the expected energy is less than 15 mA until step 1. Thus the original specification of Figure 7 is true. However since $P[A(t) = wait] > 0.28$ is true from step 3 onward, the modified specification is false.

6 Discussion

Our case study suggests that our model estimation and model checking based performance evaluation methods may be useful for establishing some aggregate properties of certain large-scale sensor networks. In fact, we believe that, given the nature of applications on sensor networks, many sensor networks will be appropriately modeled as DTMCs. However, it is possible that there is more than one application running on a sensor network, or that there are some boundary conditions causing the behavior of certain subset of nodes to be different. In this case, we can model subsets of the sensor network as distinct DTMCs. We are currently studying methods for extending our specification logic so that it can specify properties on concurrent runs of multiple DTMCs. Such an extension will allow us to analyze the subsystems separately and then aggregate or compare their performance. It will also enable

us to evaluate the effects of different initial conditions on the performance of a system.

References

- [1] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. In *Journal of Functional Programming*, volume 7, pages 1–72. Cambridge University Press, 1997.
- [2] J. Aslam, Z. Butler, F. Constantin, V. Crepsi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Sensys*, 2003.
- [3] A. Aziz, V. Singhal, and F. Balarin. It usually works: The temporal logic of stochastic systems. In *CAV*, pages 155–165. LNCS 939, 1995.
- [4] C. Baier, J. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR*, pages 146–162. LNCS 1664, 1999.
- [5] G. Ciardo, R. A. Marie, B. Sericola, and K. S. Trivedi. Performability analysis using semi-markov reward process. In *IEEE Transactions on Computers*, volume 39, pages 1251–1264, October 1990.
- [6] Crossbow Technology, Inc. <http://www.xbow.com/>.
- [7] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. In *Formal Aspects of Computing*, volume 6, pages 512–535, 1994.
- [8] M. Kwiatkowska, G. Norman, and D. Parker. Prism 2.0: A tool for probabilistic model checking. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 322–323. IEEE Computer Society, 2004.
- [9] Y. Kwon and G. Agha. Linear inequality LTL (iLTL): A model checker for discrete time markov chains. In *Int. Conf. on Formal Engineering Methods*. LNCS 3308, 2004.
- [10] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc of 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.
- [11] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison Wesley, 2nd edition, 1989.
- [12] M. Maroti, G. Simon, A. Ledeczi, and J. Sztipanovits. Shooter localization in urban terrain. In *Computer*, pages 60–61. IEEE Computer Society Press, 2004.
- [13] K. Mechitov, S. Sundresh, Y. Kwon, and G. Agha. Cooperative tracking with binary-detection sensor networks. In *Technical Report UIUCDCS-R-2003-2379*. Department of Computer Science, University of Illinois at Urbana-Champaign, 2003.
- [14] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 3rd edition, 1991.
- [15] R. Sedgewick. *Algorithms in C*. Addison Wesley, 1990.
- [16] R. Szwedczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Sensys*, 2004.
- [17] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor-network for structural monitoring. In *Sensys*, 2004.