Performance Evaluation of Sensor Networks by Statistical Modeling and Euclidean Model Checking

YOUNGMIN KWON, Microsoft Corporation
GUL AGHA, University of Illinois at Urbana Champaign

39

Modeling and evaluating the performance of large-scale wireless sensor networks (WSNs) is a challenging problem. The traditional method for representing the global state of a system as a cross product of the states of individual nodes in the system results in a state space whose size is exponential in the number of nodes. We propose an alternative way of representing the global state of a system: namely, as a probability mass function (pmf) which represents the fraction of nodes in different states. A pmf corresponds to a point in a Euclidean space of possible pmf values, and the evolution of the state of a system is represented by trajectories in this Euclidean space. We propose a novel performance evaluation method that examines all pmf trajectories in a dense Euclidean space by exploring only finite relevant portions of the space. We call our method *Euclidean model checking*. Euclidean model checking is useful both in the design phase—where it can help determine system parameters based on a specification—and in the evaluation phase—where it can help verify performance properties of a system. We illustrate the utility of Euclidean model checking by using it to design a time difference of arrival (TDoA) distance measurement protocol and to evaluate the protocol's implementation on a 90-node WSN. To facilitate such performance evaluations, we provide a Markov model estimation method based on applying a standard statistical estimation technique to samples resulting from the execution of a system.

Categories and Subject Descriptors: H.4.0 [**Information Systems Applications**]: General

General Terms: Verification, Performance, Reliability

Additional Key Words and Phrases: iLTL, DTMC estimation, statistical testing, TDoA, wireless sensor networks, performance evaluation, probabilistic model checking

ACM Reference Format:
Kwon, Y. and Agha, G. 2013. Performance evaluation of sensor networks by statistical modeling and euclidean model checking. ACM Trans. Sensor Netw. 9, 4, Article 39 (July 2013), 38 pages.
DOI: http://dx.doi.org/10.1145/2489253.2489256

1. INTRODUCTION

A wireless sensor network (WSN) is a system of embedded nodes collaborating with each other through wireless communication channels. Each sensor node has its own processor and memory, one or more sensors, wireless communication channels, and an independent power source. Because nodes in a WSN are capable of computation

This article includes some materials previously published in Kwon and Agha [2006], in *Proceedings of the 12th Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*.
This material is based upon work supported in part by Defense Advanced Research Projects Agency (DARPA) under award F33615-01-C-1907, by ONR grant N00014-02-1-0715, by NSF graht CNS 05-09321, by the AFRL and the AFOSR under agreement number FA8750-11-2-0084, and by the Army Research Office under award W911NF-09-1-0273.
Authors' addresses: Y. Kwon, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052; G. Agha, Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801; email: {ykwon4, agha}@cs.uiuc.edu.

DOI: http://dx.doi.org/10.1145/2489253.2489256

and wireless communication, nodes can filter, analyze, and synthesize data locally and communicate the results. WSNs have been proposed for use in applications which involve continuously collecting and analyzing large amounts of data. Such applications include environmental monitoring [Szewczyk et al. 2004], structural health monitoring [Xu et al. 2004; Rice et al. 2010], and target tracking [Aslam et al. 2003; Mechitov et al. 2003].

We are interested in the problem of reasoning about the performance of a WSN. One method for proving the properties of a system is *model checking* [Clarke et al. 2000]; model checking involves checking all the states of a system to see if they satisfy a property. However, because of the large number of nodes in a typical WSN, it is difficult to use the traditional representation of global states for model checking. Consider a WSN with 100 nodes, each of which can be in one of three states. If we model the global state of a system as a cross product of all the local nodes, as is the case with the standard model of concurrent systems, then the system has 3^{100} possible states. This results in the well-known *state space explosion problem*, that is, enumerating the states and checking if a property holds in each state in such a system is not possible given the large number of states. Moreover, it turns out that such global states do not give much meaningful information about what is interesting about the system.

Another challenge is that the behavior of a WSN may be random. There are two sources of the randomness: the stochastic nature of events in the operating environment (at least over longer periods of operation) and the use of randomized algorithms. Because the performance of a WSN depends largely on random factors, it is important to predict and control the response to randomness in order to meet the system requirements. A complication here is that changing one design parameter often affects several other aspects of the system at the same time. Of course, this makes it even more crucial to have a systematic way of evaluating the performance of a WSN.

Addressing the State Space Explosion Problem. One approach for addressing the state space explosion problem—pioneered by the second author's group—is to use statistical methods for performance evaluation. Statistical model checking enables us to examine the probabilistic properties of systems. Instead of exploring all reachable states by executing a system along every possible path, *statistical model checking* gathers statistics from Monte Carlo simulations and checks formulas against these [Sen et al. 2005]. We take a different approach in this article: we build a model of the evolution of probability distributions of local states and then verify if the aggregate properties we are interested in hold in this model. In other words, our verification process itself is not statistical in nature. In this respect, our approach is similar to the work on *probabilistic theorem proving* [Hasan and Tahar 2009]. However, the model checking techniques we use are quite different from those used in probabilistic theorem proving, as will be explained in Section 7.

Prior work on probabilistic and statistical model checking has checked properties of systems using the *Probabilistic Computation Tree Logic* (PCTL) [Hansson and Jonsson 1994]. We argue that PCTL is not a suitable logic for evaluating how the aggregate performance properties are evolving over time, as it characterizes what fractions of the computation paths will satisfy a given specification. Rather, for evaluating the sort of aggregate properties of systems, we need to know what fraction of nodes are in certain states at any given time.

Our approach to addressing the state space explosion problem is to abstract the global system state as a *probability mass function* (pmf), where the pmf represents the fraction of the nodes in certain states [Kwon and Agha 2006]. While a pmf abstraction abstracts the state of individual nodes, it provides sufficient information to evaluate the overall performance of a system. Using a pmf representation, one can evaluate the expected

behavior of a system for properties such as expected energy consumption, reliability, availability, etc. Moreover, given the *law of large numbers* [Papoulis 1991], the fact that there are a large number of nodes in a WSN means that the pmf abstraction can provide an accurate representation of the system state. This representation also abstracts the model from the interactions between nodes and from the asynchronous events.

To differentiate our method for model checking from probabilistic model checking, we introduce the term *Euclidean model checking*; the term is appropriate as we represent a state as a point in an Euclidean space and check the validity of a property in the subspace defined by the trajectory of these points as the state evolves. Specifically, the state of a system is represented as a vector of probabilities. This means that the trajectory is bounded in a convex subspace of the Euclidean space. However, our model checking method is applicable to nonprobabilistic systems as well and is not restricted to the convex region. Specifically, we have applied the method to models of control systems where the trajectories are not bounded in a convex subspace of an Euclidean space [Kwon and Agha 2008].

Applications of Our Method. The pmf representation is useful not only for the processes that are spatially copresent, but also for processes that are temporally juxtaposed, that is, when some protocols run many times, possibly on different nodes. Specifically, the pmf abstraction may be used by collecting execution samples of different processes or protocols and aligning their starting times. Moreover, we can model the state (pmf) transition dynamics as a discrete-time Markov chain(DTMC). As far as a protocol execution is concerned, the participating processes are logically separated from the rest of the system and can be regarded as running asynchronously. The protocol evaluation example in Section 3.2 and Section 4.4 illustrate this way of modeling systems.

When the real systems are modeled in a Markov chain, their steady-state analysis can be done by computing the limiting probability distributions regardless of their initial or intermediate states. However, the transient state analysis depends on the initial states: because there are uncountably many initial states and so many state trajectories, manual checking is not possible. We propose a model checking approach that quantitively evaluates all state trajectories, that is, the expected system behaviors from every initial state to the limiting state.[1]

A Temporal Logic. The first step to enabling an automated evaluation is to describe system properties in a language that is human readable and machine interpretable. The language must be expressive enough to describe nontrivial properties. With these concerns in mind, we have developed a probabilistic temporal logic called *iLTL* that describes (un)desirable sets of state trajectories[2] of DTMCs [Kwon and Agha 2011].

iLTL has logical and temporal operators of *linear temporal logic* (LTL) [Lichtenstein and Pnueli 1985]: iLTL specifies properties resulting from the state transitions of all linear state trajectories without considering the branches. Because the state transitions of a Markov chain are deterministic once the initial state is given, LTL is a suitable logic for describing how the states would evolve over time. The atomic propositions of iLTL are linear equalities and linear inequalities about expected rewards. Observe that each linear equality specifies a hyperplane in an Euclidean space whose points are vectors representing pmf's, and each linear inequality specifies a half space in this

[1]Although the model checking algorithm can handle the dense set of initial probability distributions, the current implementation of Euclidean model checking of iLTL formulas is not robust against numerical errors.
[2]In the rest of this article, we will refer to a *probability mass function* (pmf) as a 'state.' In most cases, the term 'state' for a pmf is distinguishable from the use of the term for a state of a Markov chain. However, when necessary, we will use the term *system state* for the state of a Markov chain.

Euclidean space. Thus, their intersection forms a convex region in the Euclidean space, and we can specify arbitrary regions by taking the unions or the complements of the convex regions. Also, the temporal operators of iLTL enable us to specify how these regions should change over time.

When a Markov chain model does not satisfy an iLTL specification, the model checking results in a counterexample, that is, an initial pmf whose trailing state trajectory would violate the specification. This counterexample often contains important information about system design parameters, such as a probabilistic scheduling policy, to achieve a desired goal. Thus another application of model checking is to find an initial pmf which satisfies an iLTL specification, if such a pmf exists. This is accomplished by negating the given specification and finding a counterexample to the negated specification, if one exists (thus finding an example that satisfies the original goal). Failure to find such a counterexample tells us that the original specification cannot be met. Finding a design parameter from a counterexample or verifying the correctness or the impossibility of a design are two main applications of model checking.

How Model Checking of iLTL Formulas Works. A common assumption about Markov chains is their unique limiting probability distribution. Under this assumption, every state trajectory converges to a limiting probability distribution and so does the evaluation of atomic propositions of iLTL formulas, (in)equalities about expected values. An interesting question is at what point in the execution of a system do the evaluations of atomic propositions stop changing. We compute an upper bound of this time, and refer to it as the *search depth*. The time complexity of iLTL model checking is exponential in terms of the search depth. Because the search depth is computed before the main model checking process begins, users can adjust their specifications without waiting indefinitely for the results. Our algorithm has been implemented in a tool available at `http://osl.cs.uiuc.edu`.

An important concern about mathematical analysis on a real system is the accuracy of the model in terms of the real system: if a model does not capture real-system behaviors, analyzing the model would be meaningless. To ensure that the model is accurate, we develop a Markov chain estimation method based on the samples of a real system. Observe that because of the memoryless property, each pair of consecutive samples provides information that can be used to estimate the pmf. As a result, even samples from a single execution are often sufficient to accurately estimate the Markov chain. There are two main constraints in our model estimation method: (1) the states of DTMCs must be known and (2) the underlying systems must be identical. We explain these assumptions in more detail in Section 5.1. We also suggest the use of a statistical testing method that can reject an estimated model if the model is not sufficiently accurate.

We envision two primary applications for our proposed performance evaluation framework:

—Euclidean model checking may be useful in selecting the design parameters of a protocol for a particular application. These parameters can be found from the counterexamples of the negated goal.[3]
—Euclidean model checking can evaluate the performance of a real system whenever a Markov model is available or can be built. Users can systematically evaluate whether a model simultaneously satisfies multiple aspects of requirements, such as throughput, reliability, availability, and the energy consumption level.

[3]This is different than HyTech [Henzinger et al. 1997] which finds the necessary and sufficient constraints on unspecified parameters (the design parameters) in the system description.

Application to Performance Evaluation. We propose the following steps as a framework for evaluating the performance of large-scale probabilistic and memoryless systems.

(1) Define the states of a system such that the performance criteria of interest can be evaluated. Some natural choices of such states are the states of a protocol, the modules of a program, or the process states of a running application.
(2) Build a probability transition matrix that defines the conditional transition probabilities between the states. This matrix can be estimated by periodically sampling the execution of a real system or can be iteratively determined in the design process based on the performance evaluation results.
(3) If the probability transition matrix is estimated from a real system, then statistically test the model as to whether it accurately represents the real system.
(4) Write specifications in iLTL using the states defined in the first phase and model check whether the performance of the model satisfies the specifications.

In Steps (2) and (3), estimating and testing the probability transition matrix of Markov chains are orthogonal to the performance evaluation method of Step (4), that is, other Markov chain estimation techniques can be freely substituted.

We demonstrate the usefulness of our model checking method as a performance evaluation framework by designing a TDoA (time difference of arrival) distance measurement protocol and then evaluating its performance metrics on a WSN with 90 nodes.

Organization of Paper. In Section 2, we discuss the assumptions and the limitations of our iLTL-based performance evaluation method. Section 3 formally defines the DTMC models of iLTL and explains their properties that are used in this work. We conclude the section by building a DTMC model for a TDoA distance measurement protocol. Section 4 provides the syntax and the semantics of iLTL. We evaluate the performances of the TDoA model built in Section 3. Section 5 describes a DTMC model estimation method and a statistical test. In Section 6, the usefulness of our proposed performance evaluation framework is demonstrated by evaluating the performance of a real WSN with 90 nodes. Finally, in Section 7 we explain model checking approaches in general, performance evaluation techniques for WSNs, and the related work of our own. In particular, the differences between iLTL and conventional probabilistic temporal logics are discussed.

2. MODELING ASSUMPTIONS AND LIMITATIONS

As noted in the introduction, we assume that the behavior of the system we are analyzing is Markovian. The performance evaluation framework explained in this article may not properly handle systems with nondeterministic and probabilistic behaviors. Although we are extending our work to Markov decision processes which provide a model for systems with nondeterministic and probabilistic behaviors [Korthikanti et al. 2010], we focus here on fully probabilistic Markovian systems.

Markov models have two useful properties which simplify the analysis of a system: namely, the *memoryless* property and *the unique limiting probability distribution* property.[4] Because sensor nodes often make decisions based on their current state (or on a limited history which can be encoded in a small number of states), sensor networks

[4]The unique limiting probability distribution property is a property of Markov chains such that their probability distributions at the infinite future are determined solely by the structure of the chains regardless of their initial probability distributions. Not every Markov chain has this property, and we discuss this issue later with more details.

satisfy the memoryless property, and we can use DTMCs to represent the system behaviors quite accurately [Kwon and Agha 2006]. The memoryless property simplifies the computation of future probabilities, and the unique limiting probability distribution property makes the steady-state.[5] analysis statically feasible. In other words, one can easily compute the steady state of a finite state Markov chain from the probability transition matrix or from the infinitesimal generator matrix without considering the initial states.

The system models of this article are built up from a system composed of many subsystems. Our model estimation and model checking methods assume that these underlying systems are identical. As we will further discuss in Section 5.1, this identicalness does not mean that individual sensor nodes are identical. It simply means that the processes we are interested in are identical. For example, a distance measurement operation may involve several sensor nodes that may have different hardware or different software running on it, but the distance measurement protocol should be identical regardless of the set of sensor nodes involved.

If a system is composed of several nonidentical systems, one can still adapt our performance evaluation method. Using the Kronecker operators or Stochastic Automata Networks (SAN),[6] a single system model can be built out of nonidentical subsystems even when there are interactions between the subsystems [Langville and Stewart 2004; Plateau and Atif 1991; Lam et al. 2004]. In this case, the size of the model can grow exponentially in terms of the number of nonidentical subsystems. Observe that the size of the model does not grow as a function of the number of nodes, but as a function of the number of different kinds of the subsystems.

Our proposed performance evaluation framework is well suited for systems with many processes running concurrently. However, the framework is also effective when a single process is run repeatedly (i.e., sampled in time). As long as the repeated processes are identical and independent, by juxtaposing their sampled executions one can consider them as one would consider samples from concurrent processes. Even in the case of a single execution of a process, our performance evaluation methods are still applicable. In this case, the model checking result may have a purely mathematical meaning. Just as the expected value of 3.5 cannot be observed in a single dice-rolling experiment, the model checking result may be different from real-system behavior. However, from the viewpoint of the expectations, the results are still valid.

The atomic propositions of iLTL are linear equalities or inequalities about expected rewards. The current implementation of the iLTL model checker uses linear programming against these atomic propositions to find a counterexample and may have numerical errors due to rounding. For example, the checker may fail to detect that $\mathrm{P}[D = s] > 0.1 \vee \mathrm{P}[D = s] \leq 0.1$ is a tautology. However, in most cases, one can add small margins in the specification to avoid this numerical problems.

Euclidean model checking does not directly address the problem of interactions between nodes, and it does not model the behavior of a system in response to rare environmental events (e.g., the effect of a typhoon on the behavior of a bridge's structural health monitoring system). However, for the sort of aggregate properties we are interested in, this need not be a serious limitation. While individual interactions are asynchronous events, over larger time scales, the behavior may nevertheless be

[5]A steady state, is a state, or a probability distribution in this article, that does not change over time. With the unique limiting probability distribution property, it is the probability distribution at the infinite future.

[6]SAN is a model for interacting Markov processes. Specifically, SAN uses synchronized events to model the interactions between systems. Because CTMC models are used in SAN, one can convert DTMC subsystems to CTMCs in order to combine them and convert the combined model back to a DTMC. Please refer to Section 3.1 for the conversion between CTMCs and DTMCs.

Markovian. Moreover, rare events are by definition, rare. This means that the effect of rare events on the aggregate properties of a system over a longer period of time may be negligible. Alternately, one can also simulate rare events and analyze their effect by building a model of the system response to such events, thus establishing both properties in the immediate aftermath of a rare event and determining whether their impact over a longer period is negligible.

3. MARKOV CHAIN MODELS

In this section, we formally define Markov chain models and their computation paths. We also explain some properties of these models that are important for the analysis we describe in this article. We conclude this section by building a DTMC model for a *time difference of arrival* (TDoA) distance measurement protocol.

3.1. Markov Chain Models

We model real-world systems as Markov chains and evaluate the performances of the systems by checking the properties of the model. In this section, we formally define Markov chains and their computation paths. The computation paths are state trajectories of Markov chains described in difference equations. We solve the equations into their closed form such that any future states can be expressed in terms of their initial states. We also define Markov reward chains to express the properties in expected reward forms and multiple DTMC models to compare the computation paths of several DTMCs together. Some interesting operations on Markov chains used in this article are explained here. Some of them are the discretization of CTMCs and a probability aggregation operator. We show that the matrix diagonalization plays a key role in these operations.

Let Ω be a universal set called *sample description space*. Then a class of subsets of Ω, called $\mathcal{F}$, is a *σ-field* if it has the empty set $\emptyset$ and the universal set Ω and is closed under countable intersections, unions, and complements. Measures can be defined on σ-fields and the σ-fields that have a measure that is consistent with the probability axioms[7]—also known as a *probability measure*—are called sets of *events*. A *probability space* $\mathcal{P}$ is a triple $(\Omega, \mathcal{F}, \mathrm{P})$, where Ω is a sample description space, $\mathcal{F} \subseteq 2^{\Omega}$ is a σ-field of events, and $\mathrm{P} : \mathcal{F} \to [0, 1]$ is a probability measure. A *random variable* (rv) $X : \Omega \to \mathbb{R}$ is a mapping from the sample description space to the real line such that the set $\{\zeta \in \Omega : X(\zeta) \leq x\}$ is an event for every $x \in \mathbb{R}$, but in this article, we are only interested in mappings from Ω to a finite set of system states $S = \{s_1, \ldots, s_n\}$. We write $\mathrm{P}[X = s_i]$ for the probability $\mathrm{P}(\{\zeta \in \Omega : X(\zeta) = s_i\})$. A *stochastic process* $X : \mathbb{R}^+ \to (\Omega \to \mathbb{R})$ is a mapping from the time to rvs, but a mapping $X : \Omega \to (\mathbb{R}^+ \to \mathbb{R})$ from the sample description space to time functions called a *sample function* is also a stochastic process. Since we are interested in a finite set of system state S, we write $\mathrm{P}[X(t) = s_i]$ for the probability $\mathrm{P}(\{\zeta \in \Omega : X(\zeta)(t) = s_i\})$.

Definition 1. For a discrete rv $X : \Omega \to S$, the probability mass function (pmf) of X is a function $P_X : S \to [0, 1]$ that associates a probability to each system state $s \in S$ as $P_X(s) = \mathrm{P}[X = s]$ such that $\sum_{s \in S} P_X(s) = 1$.[8]

Following the *frequency interpretation* of probability, one can heuristically estimate a pmf as the number of outcomes that certain states appeared divided by the total number of experiments. That is, $\mathrm{P}[X = s_i] \simeq \frac{n_i}{n}$, where n is the number of experiments

[7]Probability axioms: for every event $E, F, E_i \in \mathcal{F}$, (1) $\mathrm{P}(E) \geq 0$, (2) $\mathrm{P}(\Omega) = 1$, (3) $\mathrm{P}(E \cup F) = \mathrm{P}(E) + \mathrm{P}(F)$ if $E \cap F = \emptyset$, (4) $\mathrm{P}(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mathrm{P}(E_i)$ if $E_i \cap E_j = \emptyset$ for all $i \neq j$ when the number of events is infinite.

[8]If we consider a discrete rv X that ranges over $\mathbb{R}$, the pmf $P_X : \mathbb{R} \to [0, 1]$ is defined as $\mathrm{P}[X \leq x] - \mathrm{P}[X < x]$.

and n_i is the number of outcomes s_i appeared.[9] From the law of large numbers, for any $\epsilon > 0$, the probability $\left|\mathrm{P}[X = s_i] - \frac{n_i}{n}\right| < \epsilon$ tends to 1 as n increases.

A Markov process is a stochastic process that satisfies the memoryless property, that is, the future probability distributions of a Markov process depend only on its present state. Markov processes with a countable[10] number of states are called Markov chains. Markov chains are further categorized into *continuous-time Markov chains* (CTMCs) and *discrete-time Markov chains* (DTMCs).

Definition 2. Let $(\Omega, \mathcal{F}, \mathrm{P})$ be a probability space. A discrete-time Markov chain (DTMC) is a stochastic process $D : \mathbb{N} \to (\Omega \to S)$ that satisfies the following memoryless property.

$$\mathrm{P}[D(k) = s_k \mid D(k-1) = s_{k-1}, \ldots, D(0) = s_0] \;=\; \mathrm{P}[D(k) = s_k \mid D(k-1) = s_{k-1}],$$

for all $s_k, \ldots, s_0 \in S$ and for all $k \in \mathbb{N}$. Similarly, a continuous-time Markov chain (CTMC) is a stochastic process $C : \mathbb{R}^+ \to (\Omega \to S)$ that satisfies the following memoryless property.

$$\mathrm{P}[C(t+s) = s_a \mid C(s) = s_b, C(r) = s_c] \;=\; \mathrm{P}[C(t+s) = s_a \mid C(s) = s_b],$$

for all $s_a, s_b, s_c \in S$ and for all $t, s, r \in \mathbb{R}^+$ such that $s \geq r$.

Definition 3. Let D be a DTMC and $\mathbf{M} \in [0,1]^{n \times n}$ be a probability transition matrix such that $\mathbf{M}_{ij} = \mathrm{P}[D(k+1) = s_i \mid D(k) = s_j]$ for $k \geq 0$. A state transition diagram (STD)[11] $\dot{D} = (S, \mathbf{M})$ of D then is a labeled directed graph whose nodes are the states in S and whose edges are the state transitions labeled with their probabilities: the set of edges is $\{(s_j, s_i, \mathbf{M}_{ij}) \in S \times S \times [0,1] : \mathbf{M}_{ij} > 0\}$.

For a CTMC C, let $\mathbf{R} \in \mathbb{R}^{n \times n}$ be an infinitesimal generator matrix such that $\mathbf{R}_{ij}$ is the state transition rate from s_j to s_i and $\mathbf{R}_{ii} = -\sum_{j=1, j \neq i}^{n} \mathbf{R}_{ji}$. An STD $\dot{C} = (S, \mathbf{R})$ of C then is a labeled directed graph whose nodes are the states in S and whose edges are the state transitions labeled with their rates. The set of edges is $\{(s_j, s_i, \mathbf{R}_{ij}) \in S \times S \times \mathbb{R} : \mathbf{R}_{ij} \neq 0\}$.

Note that although the formal model of our specification logic is a DTMC, systems modeled as CTMCs can still be analyzed after discretizing them through periodic sampling. We discuss this topic in detail later in this section.

Definition 4. A computation path of a DTMC $\dot{D} = (S, \mathbf{M})$ is a probability vector function $\mathbf{x} : \mathbb{N} \to [0,1]^n$ such that $\mathbf{x}(k)_i = \mathrm{P}[D(k) = s_i]$ for $i = 1, \ldots, n$. A computation path of a CTMC $\dot{C} = (S, \mathbf{R})$ is a probability vector function $\mathbf{y} : \mathbb{R}^+ \to [0,1]^n$ such that $\mathbf{y}(t)_i = \mathrm{P}[C(t) = s_i]$ for $i = 1, \ldots, n$.

The pmf vectors at a certain time $t \in \mathbb{R}^+$ or $k \in \mathbb{N}$ can be expressed in terms of their initial pmf vector by solving the Kolmogorov forward equation $\frac{d}{dt}\mathbf{y}(t) = \mathbf{R} \cdot \mathbf{y}(t)$.

$$\mathbf{y}(t) \;=\; e^{\mathbf{R} \cdot t} \cdot \mathbf{y}(0), \quad \text{where } e^{\mathbf{R} \cdot t} = \sum_{k=0}^{\infty} \frac{(\mathbf{R} \cdot t)^k}{k!}, \tag{1}$$

and by solving the difference equation $\mathbf{x}(k+1) = \mathbf{M} \cdot \mathbf{x}(k)$.

$$\mathbf{x}(k) \;=\; \mathbf{M}^k \cdot \mathbf{x}(0). \tag{2}$$

[9]The probability measure estimated this way is compatible with the probability axioms: $\mathrm{P}[E] \geq 0$ because $\mathrm{P}[E] = \sum_{s_i \in E} \frac{n_i}{n}$, $\mathrm{P}[\Omega] = 1$ because $\sum_{s_i \in S} n_i = n$, $\mathrm{P}[E \cup F] = \mathrm{P}[E] + \mathrm{P}[F]$ when E and F are disjoint because $\mathrm{P}[E \cup F] = \sum_{s_i \in E \cup F} \frac{n_i}{n} = \sum_{s_i \in E} \frac{n_i}{n} + \sum_{s_i \in F} \frac{n_i}{n} = \mathrm{P}[E] + \mathrm{P}[F]$, and the condition about countable union is unnecessary here because $|S|$ is finite.

[10]In this paper, we consider only Markov chains with a *finite* number of states.

[11]For simplicity, we call $\dot{D}$ a DTMC instead of an STD of DTMC. Similarly, we call $\dot{C}$ a CTMC.

Using these closed form equations, we can rewrite future states in terms of their initial states and can effectively reduce the model checking space to that of a single initial pmf vector.

Conventionally, the solutions $\mathbf{y}$ and $\mathbf{x}$ of the differential and difference equations are referred to as *state trajectories*, pmf vectors $\mathbf{y}(t)$ and $\mathbf{x}(k)$ in the state trajectories as *states*, the changes of states as *state transitions*, and $\mathbf{M}$ and $e^{\mathbf{R}}$ as *state transition matrices*. In the view of classical model checking, these uncountable sets of pmfs are analogous to the finite set of states of Kripke structures [Hughes and Creswell 1997].

It is often necessary to examine the state transitions of multiple systems together in order to compare their respective performances or to evaluate their combined effects. To support this capability, we extend the DTMC model to a *multiple DTMC* model. A multiple DTMC model is simply a multiset of *noninteracting* DTMCs denoted by a disjoint sum[12] of individual DTMCs. However, we arrange them in the following structure that is similar to the DTMC model so that the analysis and the explanations we made on the DTMC model can be applied similarly [Kwon and Agha 2011].

Definition 5. Let $\dot{D}^{(i)} = (S^{(i)}, \mathbf{M}^{(i)})$ for $i = 1, \ldots, m$ be noninteracting individual DTMCs with (pairwise) disjoint states $S^{(i)}$.[13] Then their multiple DTMC, denoted by $\dot{D}^{(1)} + \cdots + \dot{D}^{(m)}$ or by $\sum_{i=1}^{m} \dot{D}^{(i)}$, is a tuple $(S, \mathbf{M})$, where

$$S = S^{(1)} \cup \cdots \cup S^{(m)} \text{ and } \mathbf{M} = \begin{bmatrix} \mathbf{M}^{(1)} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{M}^{(m)} \end{bmatrix}.$$

The block diagonal matrix $\mathbf{M}$ is a probability transition matrix: all elements are nonnegative, and each column adds up to one. Thus, many theorems valid for a DTMC are still valid for the multiple DTMC model. However, we would like to emphasize that its state is not a single pmf but a set of pmfs—the trajectory of each pmf is determined by the block diagonal matrix $\mathbf{M}$.

We define a *computation path* of a multiple DTMC model as an extended probability vector function: let $\mathbf{x}^{(i)}$ be the probability vector function of $D^{(i)}$. Then the extended vector function $\mathbf{x} : \mathbb{N} \to [0,1]^{|S^{(1)}|+\cdots+|S^{(m)}|}$ is $\mathbf{x}(k) = [\mathbf{x}^{(1)}(k)^T, \ldots, \mathbf{x}^{(m)}(k)^T]^T$ for $k \geq 0$. With the extended vector function $\mathbf{x}$, the following difference equations still hold.

$$\mathbf{x}(k+1) = \mathbf{M} \cdot \mathbf{x}(k) \text{ and } \mathbf{x}(k) = \mathbf{M}^k \cdot \mathbf{x}(0) \quad \text{for } k \geq 0.$$

Because of the similarities between the DTMC models and the multiple DTMC models, we call them both DTMCs.

A *Markov reward chain* (MRC) [Ciardo et al. 1990] is a Markov chain extended with a reward function. An *atomic proposition* of iLTL is an (in)equality about the expected reward of an MRC. Because a typical iLTL formula contains multiple different atomic propositions, an iLTL formula can be thought of as specifying properties on the parallel transitions of multiple MRCs with the same underlying Markov chain. However, we would like to clarify that the main purpose of defining MRCs here is to simplify the explanation of the specifications on the computation paths, not to use MRCs as a model of the specification. Specifically, we use MRCs to specify regions in the pmf space: an (in)equality about the expected reward of an MRC forms a half space or hyper plane in the pmf space, which are the building blocks to specifying complicated regions in the pmf space.

[12]This type of construction is well known in universal algebra. Also, interacting DTMCs can be modeled as a single DTMC by SAN [Plateau and Atif 1991].

[13]This condition can be easily met by renaming the common states with fresh names.

Definition 6. A Markov reward chain R is a triple $R = (S, \mathbf{M}, \rho)$, where $\dot{D} = (S, \mathbf{M})$ is a DTMC and $\rho : S \to \mathbb{R}$ is a reward function. An *expected reward* of R at time k, denoted by $\mathrm{E}[R(k)]$, is

$$\mathrm{E}[R(k)] \;=\; \sum_{i=1}^{n} \rho(s_i) \cdot \mathrm{P}[D(k) = s_i].$$

A reward can be regarded as a value earned when the process visits a state. In this article, we consider only constant rewards. Thus, a reward function can be represented by a row vector $\mathbf{r} \in \mathbb{R}^{|S|}$ with $\mathbf{r}_i = \rho(s_i)$. With this vector representation, the expected reward can be written in a vector form as $\sum_{i=1}^{n} \rho(s_i) \cdot \mathrm{P}[D(k) = s_i] = \mathbf{r} \cdot \mathbf{x}(k)$.

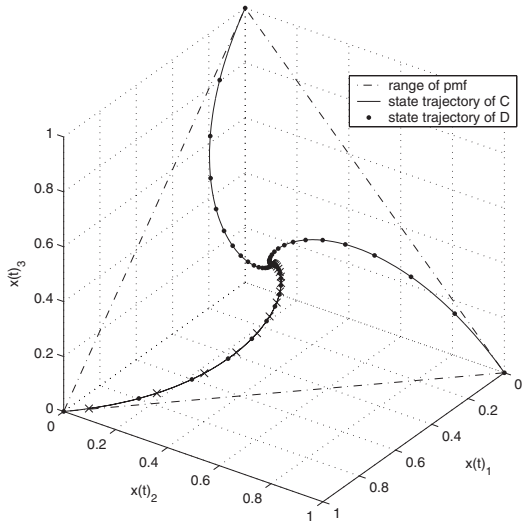
Operations on DTMCs. Many interesting properties about DTMCs can be expressed in terms of expected rewards. For example, there is a reward function corresponding to a time offset in the probability: let d be a time offset. Then $\mathrm{P}[D(k+d) = s_i] = (\mathbf{M}^d)_{i*} \cdot \mathbf{x}(k) = \mathrm{E}[R(k)]$, where $\mathbf{M}_{i*}$ is the i^{th} row of $\mathbf{M}$ and $R = (S, \mathbf{M}, \rho)$ with $\rho(s_j) = (\mathbf{M}^d)_{ij}$ for $1 \le j \le n$. Another interesting example is an accumulated reward, such as the expected amount of total energy required for an application. Because such accumulated rewards are frequently used as a performance criterion, we define a special *aggregation operator* $\mathbf{Q}$ for transient states s_i (i.e., such that $\lim_{t\to\infty} \mathrm{P}[D(t) = s_i] = 0$) as

$$\mathrm{Q}[D(k) = s_i] \;=\; \sum_{t=k}^{\infty} \mathrm{P}[D(t) = s_i].$$

The aggregation operator Q can be expressed as an expected reward of an MRC $R = (S, \mathbf{M}, \rho)$ with $\rho(j) = \left(\sum_{t=0}^{\infty} \mathbf{M}^t\right)_{ij}$ for $1 \le j \le n$, because $\mathrm{Q}[D(k) = s_i] = \left(\sum_{t=0}^{\infty} \mathbf{M}^t\right)_{i*} \cdot \mathbf{x}(k) = \mathrm{E}[R(k)]$. With the preceding definition of Q, the total expected energy consumption of a protocol modeled by an MRC $(S, \mathbf{M}, e)$ can be simply written as $\sum_{s\in S} e(s) \cdot \mathrm{Q}[D(0) = s]$, where D is a DTMC $(S, \mathbf{M})$ and $e : S \to \mathbb{R}$ is a function for the expected energy consumption of a state during a sampling period. Q also represents the expected number of times a state is visited. For example, let I be the initial state of a retrial-based error recovery system, then $\mathrm{Q}[D(0) = I]$ is the total expected number of retrials.

The diagonalization (also called *spectral decomposition*) of a probability transition matrix is one of the prerequisites for Euclidean model checking. Such diagonalization also ensures numerical stability and eases the computation of the matrix power, the infinite matrix sum in the aggregation operator Q, and the matrix exponential.[14] Let $\mathbf{diag}(\mathbf{v})$ be a diagonal matrix whose diagonal is $\mathbf{v}$ and let $\mathbf{M} = \mathbf{P} \cdot \mathbf{diag}([\lambda_1, \ldots, \lambda_n]) \cdot \mathbf{P}^{-1}$ be the diagonalization of $\mathbf{M}$.[15] Then, a numerically stable solution for the matrix power $\mathbf{M}^t$ can be computed as $\mathbf{P} \cdot \mathbf{diag}([\lambda_1^t, \ldots, \lambda_n^t]) \cdot \mathbf{P}^{-1}$. The conversion between CTMC and DTMC can be easily performed as $e^{\mathbf{M}} = \sum_{k=0}^{\infty} \frac{\mathbf{M}^k}{k!} = \mathbf{P} \cdot \mathbf{diag}([\sum_{k=0}^{\infty} \frac{\lambda_1^k}{k!}, \ldots, \sum_{k=0}^{\infty} \frac{\lambda_n^k}{k!}]) \cdot \mathbf{P}^{-1} = \mathbf{P} \cdot \mathbf{diag}([e^{\lambda_1}, \ldots, e^{\lambda_n}]) \cdot \mathbf{P}^{-1}$, where $\mathbf{M}$ here is an infinitesimal generator matrix instead of a probability transition matrix. For computing the matrix exponential, there are computationally more efficient techniques, such as uniformization. However, because the Euclidean model checking algorithm already requires diagonalization, we use the technique described here. The infinite

[14]Note that using the Jordan form, these operations can be performed on 'defective' matrices as well.
[15]Hence, $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $\mathbf{M}$.



Let a CTMC C and its discretized DTMC D be

$$\dot{C} = (\{s_1, s_2, s_3\}, R),$$
$$\dot{D} = (\{s_1, s_2, s_3\}, M),$$
$$R = \begin{bmatrix} -0.2 & 0 & 0.2 \\ 0.2 & -0.2 & 0 \\ 0 & 0.2 & -0.2 \end{bmatrix},$$
$$M = e^R.$$

The state trajectories of C and D are

$$y(t) = e^{R \cdot t} \cdot y(0),$$
$$x(k) = M^k \cdot x(0),$$

where $y(t)_i = \mathrm{P}[C(t) = s_i]$ and $x(k)_i = \mathrm{P}[D(k) = s_i]$ for $i = 1, 2, 3$.

Fig. 1. State trajectories of CTMC C (solid lines) and of its sampled DTMC D (dots and crosses).

matrix sum $\sum_{k=0}^{\infty} \mathbf{M}^k$ of the aggregation operator Q can be computed similarly as $\mathbf{P} \cdot \mathbf{diag}([\sigma_1, \ldots, \sigma_n]) \cdot \mathbf{P}^{-1}$, where $\sigma_i = \frac{1}{1-\lambda_i}$ if $|\lambda_i| < 1$ and $\sigma_i = 0$ otherwise.[16]

Discretization of CTMCs. When the infinitesimal generator matrix of a CTMC C is diagonalizable, one can get its discretized DTMC D such that its state trajectories correspond to a series of states obtained by periodically sampling on the state trajectories of C. Specifically, let T be a sampling period, $t = T \cdot k$, and $\mathbf{R}$ be an infinitesimal generator matrix. Then

$$\mathbf{y}(t) = e^{\mathbf{R} \cdot t} \cdot \mathbf{y}(0) = (e^{\mathbf{R} \cdot T})^k \cdot \mathbf{x}(0) = \mathbf{x}(k).$$

Thus, the DTMC obtained by sampling a CTMC $\dot{C} = (S, \mathbf{R})$ with a period T is $\dot{D} = (S, e^{\mathbf{R} \cdot T})$. Also, the generator matrix of a CTMC that can produce pmf trajectories that pass through the discrete pmf trajectories of a DTMC can be obtained by taking the log on the eigenvalues of the probability transition matrix. This conversion is useful when we need to combine DTMCs obtained with different sampling periods. Moreover, techniques such as Stochastic PEPA [Calder et al. 2006] can generate a CTMC from a high-level description about the system. Thus, one can get a sampled DTMC from the CTMC.

As an example of this conversion, Figure 1 shows state trajectories of a CTMC C and of its discretized DTMC D. The three solid lines are state trajectories of C starting from three initial states $\mathbf{y}(0) = [1, 0, 0]^T$, $\mathbf{y}(0) = [0, 1, 0]^T$, and $\mathbf{y}(0) = [0, 0, 1]^T$. These trajectories converge to a unique limiting pmf $\mathbf{y}(\infty) = [1/3, 1/3, 1/3]^T$. The dots in the figure are state trajectories of D starting from the same initial states as their continuous counterparts. As the figure shows, the discrete state trajectories are exactly on the continuous state trajectories of the same starting point. In fact, for any point in a continuous state trajectory, there is a sampled state trajectory that steps on the

[16]Some readers may be curious why $\sigma_i = 0$ when $|\lambda_i| = 1$. Suppose that s_i is a transient state and let $\mathbf{I}_{i*}$ be the i^{th} row of the identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$. Then $\mathbf{I}_{i*}$ is orthogonal to the columns $\mathbf{P}_{*j}$ of $\mathbf{P}$ corresponding to the eigenvalues $|\lambda_j| = 1$; otherwise $\lim_{t \to \infty} \mathbf{x}(t)_i$ will never vanish for some $\mathbf{x}(0)$, as can be inferred from $\mathbf{x}(t)_i = \mathbf{I}_{i*} \cdot \mathbf{P} \cdot \mathbf{diag}([\lambda_1^t, \ldots, \lambda_n^t]) \cdot \mathbf{P}^{-1} \cdot \mathbf{x}(0)$. Because $\mathbf{I}_{i*} \cdot \mathbf{P}_{*j} = 0$ when $|\lambda_j| = 1$, we can safely set $\sigma_j = 0$.

point. Specifically, for any point $\mathbf{y}(t)$ on $\mathbf{y}$, there is a discrete state trajectory $\mathbf{x}$ such that $\mathbf{x}(k) = \mathbf{y}(t)$ for some $k \in \mathbb{N}$ and $\mathbf{x}(0) = \mathbf{y}(t_0)$ with some $t_0 \in [0, T)$. For example, the crosses in Figure 1 show a state trajectory of D when its initial state $\mathbf{x}(0)$ equals $\mathbf{y}(1/3)$. In Section 4, we show how we examine all state trajectories starting from any point on the range of pmf (the dot dashed triangle).

When discretizing a CTMC to a DTMC, the granularity of the sampling is an important parameter which significantly affects the model checking process. Intuitively, fast sampling will produce a sequence of pmfs that does not change much, and slow sampling may miss some important changes in pmfs. To elaborate, suppose that the infinitesimal generator matrix of a CTMC is diagonalized as $\mathbf{R} = \mathbf{P} \cdot \mathbf{diag}([\lambda_1, \ldots, \lambda_n]) \cdot \mathbf{P}^{-1}$ and that the initial pmf $\mathbf{y}(0) = \mathbf{P}_{*i}$ is the eigenvector corresponding to eigenvalue λ_i, then the state trajectory, called *mode* corresponding to eigenvalue λ_i, lies on a line $\mathbf{y}(t) = \mathbf{P}_{*i} \cdot e^{\lambda_i \cdot t}$ for $t \geq 0$. Observe that a state trajectory can be expressed as a linear combination of the modes. Its discretized state trajectory sampled at a period T is $\mathbf{x}(k) = \mathbf{P}_{*i} \cdot e^{\lambda_i \cdot T \cdot k}$ for $k \geq 0$. Thus, fast sampling (small T) produces slowly changing modes which could increase the number of steps required for model checking without adding any meaningful information. Moreover, the sampling itself could change the characteristics of the system. On the other hand, slow sampling (large T) may lose the details of the evolutionary behavior of a system. For example, during a sampling period, a system's output may exceed a safety boundary and then return within the boundary—without the safety violation being detected in the model checking process. Slow sampling could also dilate the accuracy of the quantitative evaluations, especially when the quantities are accumulated rewards, such as the total energy consumption. In this case, approximating the curve by the piecewise linear model[17] could improve the accuracy compared to using the piecewise constant models. Fortunately, the expected rewards for the piecewise linear model can be easily expressed, for example, if $E = \sum_{i=1}^{n} \mathbb{Q}[D(k) = s_i]$ is an accumulated reward for a piecewise constant model, then its piecewise linear model counterpart is $E' = \sum_{i=1}^{n} 0.5 \cdot (\mathbb{Q}[D(k) = s_i] + \mathbb{Q}[D(k+1) = s_i])$. Observe that E' can be written as an expected reward for $D(k)$. Although the piecewise linear model could improve the accuracy, it will not eliminate the discretization errors entirely. In control theory, a sampling rate of nine samples during the rise time of a system is accepted as a 'rule of thumb' [Franklin et al. 1994].

3.2. DTMC Model of a TDoA Protocol

We now model a time difference of arrival (TDoA) distance measurement protocol as a DTMC. We first build a CTMC model using state transition rates and reliabilities of the modules. We then discretize the CTMC model to get a DTMC model. The performance metrics of the system will be evaluated in the next section using the DTMC model obtained.

TDoA is a commonly used protocol for WSNs, particularly in localization services (e.g., see [Kwon et al. 2010]). The TDoA protocol involves three nodes: a pair of sender/receiver nodes and a central node which initiates the protocol and collects the result. Although TDoA can be run efficiently by collectively measuring the distances between a single sender node and all of its neighboring receiver nodes, in our example, we assume that the TDoA protocol measures the distance between a single sender node and a single receiver node. The protocol may run many times with

[17]When constructing a continuous signal from a discretized signal, the piecewise linear model (i.e., first-order hold model) approximates the curve by pieces of line segments connecting the sampled values, whereas the piecewise constant model (i.e., zero-order hold model) assumes constants between the samples and jumps at the sample points.

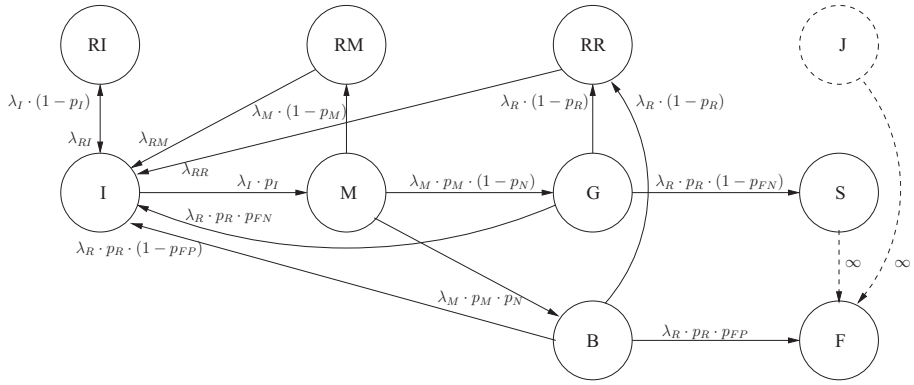


Fig. 2. A state transition diagram of a TDoA protocol.

different sender/receiver pairs. To get their average behavior, state transitions of these executions are chronologically juxtaposed so that their starting times are aligned.[18]

TDoA measures the distance between two nodes by measuring the delay between receiving the radio signal and receiving the sound signal. Figure 2 shows a state transition diagram of the TDoA protocol with timeout-based retrials. We define the states as phases of the protocol using the following four steps.

(1) A central node sends an *init* command to a sender node (*I* state).
(2) The sender node transmits a radio signal and a sound signal to a receiver node (*M* state).
(3) The receiver makes either a good measurement (*G* state) or a bad measurement (*B* state), depending on the presence of a noise, and reports the result back to the central node.
(4) The central node checks the result and either accepts it or discards it to restart the protocol.

If a good measurement is accepted, the process is a success (*S* state), and if a bad measurement is accepted, the process is a failure (*F* state). This TDoA protocol relies on the timeout mechanism to recover from various failures: if the central node does not receive a result within a timeout period, it restarts the protocol. The *RI, RM,* and *RR* states represent the restart states from different states of the protocol. The *J* state and the infinite rates of Figure 2 are explained later.

Table I summarizes the parameters of the STD. We assumed that (a) sending the *init* command takes 1 sec ($1/\lambda_I$) and has *reliability* (i.e., the probability of success) $p_I = 0.9$; (b) the measurement takes 2 sec ($1/\lambda_M$) and has reliability $p_M = 0.7$; (c) reporting takes 1 sec ($1/\lambda_R$) and has reliability $p_R = 0.9$; and (d) the timeout limit is 9 sec. Hence, the restart rates from the *RI, RM,* and *RR* states are $\lambda_{RI} = 1/(9-1)$, $\lambda_{RM} = 1/(9-1-2)$, and $\lambda_{RR} = 1/(9-1-2-1)$, respectively.

The performance of the measurement depends on the sensitivity of the sensors: increasing the sensitivity will increase the probability of making a measurement (p_M), but it will also increase the probability of picking up noise (p_N) instead of the signal. We assumed that $p_N = 0.5$. The central node checks the consistency of the reported measurements; the error probabilities are $p_{FP} = 0.3$ for the false positives and $p_{FN} = 0.3$ for the false negatives.

[18]This sample collection method may look different than the example of Section 6, where the samples are collected from multiple nodes and are spatially juxtaposed.

Table I. Transition Rates and Probabilities of Events for the TDoA Example

Parameter	Rate (Hz)	Parameter	Probability
λ_I	1/1	p_I	0.9
λ_M	1/2	p_M	0.7
λ_R	1/1	p_R	0.9
λ_{RI}	$1/(9-1)$	p_N	0.5
λ_{RM}	$1/(9-3)$	p_{FP}	0.3
λ_{RR}	$1/(9-4)$	p_{FN}	0.3

Note: The timeout period is nine sec.

$$\mathbf{R} = \begin{bmatrix} -\lambda_I & 0 & \lambda_R \cdot p_R \cdot p_{FN} & \lambda_R \cdot p_R \cdot (1-p_{FP}) & 0 & 0 & \lambda_{RI} & \lambda_{RM} & \lambda_{RR} \\ \lambda_I \cdot p_I & -\lambda_M & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_M \cdot p_M \cdot (1-p_N) & -\lambda_R & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_M \cdot p_M \cdot p_N & 0 & -\lambda_R & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_R \cdot p_R \cdot (1-p_{FN}) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_R \cdot p_R \cdot p_{FP} & 0 & 0 & 0 & 0 & 0 \\ \lambda_I \cdot (1-p_I) & 0 & 0 & 0 & 0 & 0 & -\lambda_{RI} & 0 & 0 \\ 0 & \lambda_M \cdot (1-p_M) & 0 & 0 & 0 & 0 & 0 & -\lambda_{RM} & 0 \\ 0 & 0 & \lambda_R \cdot (1-p_R) & \lambda_R \cdot (1-p_R) & 0 & 0 & 0 & 0 & -\lambda_{RR} \end{bmatrix}$$

Fig. 3. Based on the state transition rates and the probabilistic choices between states, an infinitesimal generator matrix $\mathbf{R}$ is built for the TDoA protocol.

A CTMC $\dot{C} = (S, \mathbf{R})$ is built from the STD of Figure 2 and the parameters of Table I, where $S = \{I, M, G, B, S, F, RI, RM, RR\}$[19] and $\mathbf{R}$, as given in Figure 3. Sampling C with a period T produces a DTMC $\dot{D} = (S, \mathbf{M})$, where $\mathbf{M} = e^{\mathbf{R} \cdot T}$. In Figure 2, there is a dashed arrow from state S to state F with infinite rate. This arrow is added to ensure that D has a unique limiting state, which is one of the requirements of Euclidean model checking.[20] Although there are difficulties in handling the infinite rate in a CTMC, the rate becomes 1 in a DTMC: $\mathrm{P}[D(k+1) = S | D(k) = S] = \mathbf{M}_{5,5} = 0$ and $\mathrm{P}[D(k+1) = F | D(k) = S] = \mathbf{M}_{6,5} = 1$. That is, D certainly makes a transition from state S to state F in the next step. With this transformation, we read F state as a *finished* state instead of a *failure* state.

Now let us consider some examples of rewards. First, let the *reliability* of the protocol be the probability that the TDoA protocol eventually accepts a good measurement. Without the infinite rate from state S to state F, the reliability is the probability that the process ends in state S ($\mathrm{P}[D(\infty) = S]$). After adding the infinite rate, it is the accumulated probability of state S ($\mathrm{Q}[D(0) = S]$). Similarly, the expected number of trials to finish the process is the accumulated probability at state I ($\mathrm{Q}[D(0) = I]$). As a final example, let us consider the energy consumption level. We assume that sending the *init* command consumes one unit of energy, the measurement takes three units of energy, and reporting the result consumes one unit of energy. Then, the expected energy consumption during a sampling period is $\mathrm{P}[D(k) = I] + 3 \cdot \mathrm{P}[D(k) = M] + \mathrm{P}[D(k) = G] + \mathrm{P}[D(k) = B]$, and the expected energy consumption to finish the process from time k is $\mathrm{Q}[D(k) = I] + 3 \cdot \mathrm{Q}[D(k) = M] + \mathrm{Q}[D(k) = G] + \mathrm{Q}[D(k) = B]$.

4. SPECIFICATION LOGIC

In this section, we explain the syntax and semantics of our property specification logic iLTL. iLTL has the same set of logical and temporal operators as LTL. However, unlike

[19]For convenience, we assume that the states s_1 to s_9 correspond to the states I to RR of S in this order.

[20]We believe this conversion can be done automatically. However, the interpretation of the transformed states can be difficult.

the classical LTL, the atomic propositions of iLTL are equalities or inequalities about expected rewards. Thus, the truth values of the atomic propositions are evaluated in the *uncountable* space of pmfs, instead of being evaluated in each of the possible states of the system. We briefly explain our iLTL model checking algorithm with a simple model checking example. We conclude this section with several performance evaluation examples on the TDoA model we built in Section 3.2.

4.1. Syntax

The syntax of an iLTL formula ψ is as follows.

$$\begin{aligned}\psi \ ::= \ & \mathtt{T} \mid \mathtt{F} \mid ap \mid (\ \psi\ ) \mid \\ & \neg\,\psi \mid \psi\ \wedge\ \psi \mid \psi\ \vee\ \psi \mid \psi \rightarrow \psi \mid \psi \leftrightarrow \psi \mid \\ & \mathtt{X}\,\psi \mid \Box\,\psi \mid \Diamond\,\psi \mid \psi\ \mathtt{U}\ \psi \mid \psi\ \mathtt{R}\ \psi.\end{aligned}$$

An atomic proposition ap is an equality or inequality between expected rewards. For $S = \{s_1, \ldots, s_n\}$ and a DTMC $\dot{D} = (S, \mathbf{M})$, the atomic propositions are defined as follows.

$$ap \ ::= \ r_1 \cdot \mathrm{P}_{\mathrm{op}}\left[D(t_1) = s_1\right] + \cdots + r_n \cdot \mathrm{P}_{\mathrm{op}}\left[D(t_n) = s_n\right] \bowtie r,$$

where P_{op} is either P or Q, $t_i \in \mathbb{N}$ is a time offset, $r_i \in \mathbb{R}$ is a reward associated with state s_i, and $\bowtie$ is one of $<$, $\leq$, $=$, $\geq$, or $>$. Because the Q operator and the time offset can be replaced with a reward vector, we write "$\sum r_i \cdot \mathrm{P}[D = s_i] \bowtie r$" as a representative syntax for the atomic propositions. Observe that this atomic proposition is a comparison between r and the expected reward of a MRC $(S, \mathbf{M}, \rho)$ with $\rho(s_i) = r_i$. We denote the set of all atomic propositions by AP.

4.2. Semantics

We first explain the semantics of iLTL informally, and then describe its formal semantics using satisfiability relations. For the informal semantics, we categorize the elements in three groups: atomic propositions, logical operators and temporal operators.

An *atomic proposition* "$\sum_i r_i \cdot \mathrm{P}[D = s_i] \bowtie r$" is true at time t if and only if (iff) $\sum_i r_i \cdot \mathrm{P}[D(t) = s_i] \bowtie r$. The value of t is provided based on the interpretation of temporal operators. Note that by Equation (2), the truth value of ap depends only on the time t and the initial pmf $\mathbf{x}(0)$.

The meaning of *logical operators* is as follows. $\neg\,\psi$ is true at t if and only if ψ is false at t; $\psi\ \wedge\ \phi$ is true at t if and only if ψ is true at t and ϕ is true at t; and $\psi\ \vee\ \phi$ is true at t if and only if ψ is true at t or ϕ is true at t. $\psi \rightarrow \phi$ is equivalent to $\neg\,\psi\ \vee\ \phi$ and $\psi \leftrightarrow \phi$ is equivalent to $(\psi \rightarrow \phi)\ \wedge\ (\phi \rightarrow \psi)$.

The meaning of *temporal operators* is as follows. $\mathtt{X}\,\psi$ is true at t if and only if ψ is true at the next time step $t + 1$; $\Box\,\psi$ is true at t if and only if ψ is always true from t; and $\Diamond\,\psi$ is true at t if and only if ψ eventually becomes true at some time $t' \geq t$. $\psi\ \mathtt{U}\ \phi$ is true at t if and only if ψ is true before ϕ eventually becomes true, that is, there is a time $t' \geq t$ when ϕ is true and ψ is true at τ for $t \leq \tau < t'$. $\psi\ \mathtt{R}\ \phi$ is true at t if and only if ϕ holds up to and, including the first time, $(\geq t)$ ψ becomes true, but ψ is not required to hold eventually. Note that the always ($\Box$) and the eventually ($\Diamond$) operators can be written in terms of the until ($\mathtt{U}$) and the release ($\mathtt{R}$) operators: $\Box\,\psi \equiv \mathtt{F}\ \mathtt{R}\ \psi$ and $\Diamond\,\psi \equiv \mathtt{T}\ \mathtt{U}\ \psi$.

Formally, the semantics of iLTL is defined by a binary satisfaction relation $\models \subset D \times \psi$ and a ternary satisfaction relation $\models \subset (\mathbb{N} \rightarrow [0, 1]^n) \times \mathbb{N} \times \psi$, where D is a DTMC and ψ is an iLTL formula. For simplicity, we write $D \models \psi$ for $(D, \psi) \in \models$ and likewise $\mathbf{x}, t \models \psi$ for $(\mathbf{x}, t, \psi) \in \models$. Figure 4 defines the ternary satisfaction relation. This ternary relation concerns a single computation path; it states whether the pmf transitions given by the computation path $\mathbf{x}$ satisfy the formula ψ at time t. The binary

$$
\begin{array}{lcl}
\mathbf{x}, t \models \mathrm{T} & & \\
\mathbf{x}, t \not\models \mathrm{F} & & \\
\mathbf{x}, t \models \text{“}\Sigma_i r_i \cdot \mathrm{P}[D(t_i) = s_i] \bowtie r\text{”} & \Leftrightarrow & \Sigma_i r_i \cdot \mathbf{x}(t_i + t)_i \bowtie r \\
\mathbf{x}, t \models \neg \psi & \Leftrightarrow & \mathbf{x}, t \not\models \psi \\
\mathbf{x}, t \models \psi \wedge \phi & \Leftrightarrow & \mathbf{x}, t \models \psi \text{ and } \mathbf{x}, t \models \phi \\
\mathbf{x}, t \models \psi \vee \phi & \Leftrightarrow & \mathbf{x}, t \models \psi \text{ or } \mathbf{x}, t \models \phi \\
\mathbf{x}, t \models \mathrm{X}\, \psi & \Leftrightarrow & \mathbf{x}, t+1 \models \psi \\
\mathbf{x}, t \models \psi \,\mathrm{U}\, \phi & \Leftrightarrow & \text{there is } t' \geq t \text{ such that } \mathbf{x}, t' \models \phi \text{ and for all } t \leq \tau < t', \mathbf{x}, \tau \models \psi \\
\mathbf{x}, t \models \psi \,\mathrm{R}\, \phi & \Leftrightarrow & \text{for all } t' \geq t, \text{if for every } t \leq \tau < t' \ \mathbf{x}, \tau \not\models \psi \text{ then } \mathbf{x}, t' \models \phi
\end{array}
$$

Fig. 4. The semantics of the ternary satisfaction relation $\models$.

satisfaction relation concerns all computation paths; it tells whether all computation paths of D satisfy the formula ψ at time *0*.

$$D \models \psi \ \Leftrightarrow\ \mathbf{x}, 0 \models \psi \text{ for all computation path } \mathbf{x}.$$

Because $\mathbf{x}(t)$ for $t \geq 0$ is determined solely by the initial pmf $\mathbf{x}(0)$, the binary satisfaction relation expresses that ψ is true for every initial pmf.

4.3. Euclidean Model Checking

We now sketch our algorithm to check iLTL formulas. Specifically, we explain how the iLTL model checking problem is converted to a feasibility checking problem and demonstrate the overall process with a simple example.

Given a DTMC D and an iLTL formula ψ, Euclidean model checking is a process that determines whether $D \models \psi$. Also, if $D \not\models \psi$, the model checking process outputs a witness $\mathbf{x}(0)$ such that $\mathbf{x}, 0 \not\models \psi$. We now provide a brief sketch of the Euclidean model checking algorithm.

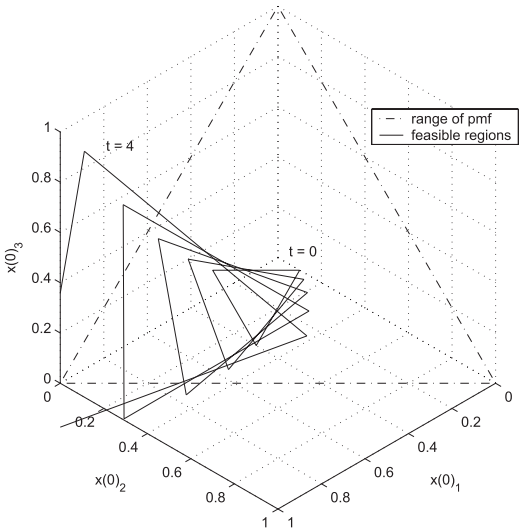
One of the key ideas of iLTL model checking algorithm is the *search depth*, that is, a time step from which the truth values of atomic propositions do not change. Hence, one can determine the truth value of the given iLTL formula instantly. As an example, let a be an atomic proposition “$\mathrm{P}[D = s_1] + \mathrm{P}[D = s_2] \leq 0.5$” with $\mathrm{P}[D(\infty) = s_1] = 0.1$ and $\mathrm{P}[D(\infty) = s_2] = 0.2$. Then, as $\mathrm{P}[D(t) = s_1] + \mathrm{P}[D(t) = s_2]$ converges to its limiting value 0.3, a moment will come when this sum never become larger than 0.5 regardless of its initial condition $0 \leq \mathrm{P}[D(t) = s_i] \leq 1$ for $i = 1, 2$, and hence the value of a remains at its limiting value, *true*. We compute an upper bound of such moment by finding a monotonically decreasing upper bound and a monotonically increasing lower bound of the LHS of a that converge toward each other. Observe that once one of these bounds crosses the RHS of a, the value of a does not change henceforth.

The model checking algorithm involves the following steps.

(1) Build a Büchi automaton for the negated specification.
(2) Find the search depth.
(3) Using the search depth, find an accepting run of the Büchi automaton such that the conjunction of the atomic propositions in the run is feasible.

Note that in Step (3), the atomic propositions of different time steps are rewritten in terms of the initial pmf using Equation (2). Any feasible initial pmf is the counterexample that we are seeking. Interested readers should refer to Kwon and Agha [2011] for further details.

As a simple model checking example, let a DTMC $\dot{D}$ be $(\{a, b\}, \mathbf{M})$, where $\mathbf{M} = \left[\begin{smallmatrix} 0.5 & 0 \\ 0.5 & 1 \end{smallmatrix}\right]$. Suppose we have a specification ψ of $10 \cdot \mathrm{P}[D = b] < 2 \vee \mathrm{X}\ 10 \cdot \mathrm{P}[D = b] < 3$.



Let D be the DTMC defined in Figure 1 and let three inequalities be

$$a_1 \ : \ \mathrm{P}[D = s_1] + \mathrm{P}[D = s_2] < 0.7,$$
$$a_2 \ : \ \mathrm{P}[D = s_1] + \mathrm{P}[D = s_3] < 0.6,$$
$$a_3 \ : \ \mathrm{P}[D = s_2] + \mathrm{P}[D = s_3] < 0.5.$$

Then,

$$\mathbf{x}, t \models a_1 \ \Leftrightarrow \ [1,1,0] \cdot \mathbf{M}^t \cdot \mathbf{x}(0) < 0.7,$$
$$\mathbf{x}, t \models a_2 \ \Leftrightarrow \ [1,0,1] \cdot \mathbf{M}^t \cdot \mathbf{x}(0) < 0.6,$$
$$\mathbf{x}, t \models a_3 \ \Leftrightarrow \ [0,1,1] \cdot \mathbf{M}^t \cdot \mathbf{x}(0) < 0.5.$$

The interiors of the solid edged triangles are the feasible regions of $\mathbf{x}, t \models a_1 \wedge a_2 \wedge a_3$ in terms of $\mathbf{x}(0)$ for $t = 0, \ldots, 4$, where $\mathbf{x}(0)_i = \mathrm{P}[D(0) = s_i]$ for $i = 1, 2, 3$.

Fig. 5. A visualization of the iLTL model checking process. Feasible regions of $a_1 \wedge a_2 \wedge a_3$ for $t = 0, \ldots, 4$ in terms of the initial pmf, $\mathbf{x}(0)$, are the interiors of the solid edged triangles.

Then

$$\begin{aligned} D \models \psi \ &\Leftrightarrow \ \text{there is no computation path } \mathbf{x} \text{ such that } \mathbf{x}, 0 \models \neg\psi \\ &\Leftrightarrow \ \{\mathbf{v} \in [0,1]^2 : [0,10] \cdot \mathbf{M}^0 \cdot \mathbf{v} \geq 2 \text{ and } [0,10] \cdot \mathbf{M}^1 \cdot \mathbf{v} \geq 3 \text{ and } \mathbf{v}_1 + \mathbf{v}_2 = 1\} = \emptyset \\ &\Leftrightarrow \ \{\mathbf{v} \in [0,1]^2 : [0,10] \cdot \mathbf{v} \geq 2 \text{ and } [5,10] \cdot \mathbf{v} \geq 3 \text{ and } \mathbf{v}_1 + \mathbf{v}_2 = 1\} = \emptyset, \end{aligned}$$

where $\mathbf{v}_0$ and $\mathbf{v}_1$ correspond to $\mathrm{P}[D(0) = a]$ and $\mathrm{P}[D(0) = b]$, respectively. Any feasible $\mathbf{v}$ is an initial pmf whose computation path would violate ψ. For example, $\mathbf{v} = [0.5, 0.5]^T$ is a feasible vector. Thus, the computation path defined as $\mathbf{x}(t) = \mathbf{M}^t \cdot [0.5, 0.5]^T$ violates the specification ψ (i.e., $\mathbf{x}, 0 \models \neg\psi$).

Figure 5 visualizes how the feasible regions of $\Box(a_1 \wedge a_2 \wedge a_3)$ with respect to the DTMC model D of Figure 1 are changing over time (only the first five steps are shown) in terms of the initial pmf. The feasible regions of $a_1 \wedge a_2 \wedge a_3$ at time t in the coordinates of $\mathbf{x}(t)_1$, $\mathbf{x}(t)_2$, and $\mathbf{x}(t)_3$ are always like the smallest solid triangle of Figure 5. However, when they are transformed in the coordinates of $\mathbf{x}(0)_1$, $\mathbf{x}(0)_2$, and $\mathbf{x}(0)_3$, they are changing like the triangles in this figure. Observe that as the state trajectory $\mathbf{x}$ converges to its unique limiting pmf $\mathbf{x}(\infty)$, the (in)equalities become less and less dependent on the initial pmf $\mathbf{x}(0)$. As a result, the edges of the triangle move infinitely away from the range of pmf (the dot dashed triangle). A search depth is an upper bound of the moment when all edges permanently leave the range of pmf. Let a specification ψ be $\Box(a_1 \wedge a_2 \wedge a_3)$, then $D \not\models \psi$ if and only if the intersection of the feasible regions $\bigcap_{t=0}^{sd}\{\mathbf{x}(0) \in [0,1]^3 : \mathbf{x}, t \models a_1 \text{ and } \mathbf{x}, t \models a_2 \text{ and } \mathbf{x}, t \models a_3\} \cap \{\mathbf{x}(0) \in [0,1]^3 : \mathbf{x}(0)_1 + \mathbf{x}(0)_2 + \mathbf{x}(0)_3 = 1\}$ is empty, where sd is a search depth. In other words, if the intersection of these $sd + 2$ triangles is not empty, then any point in the intersection is a counterexample of $D \models \neg\psi$. In general, we are checking the feasibility of such interactions of constraints for each run of a Büchi automaton.

Observe that despite the uncountably large state space and the infinite length of computation paths, there are only a finite number of linear (in)equalities to check. Hence, we can find a feasible solution or declare its nonexistence up to the precision

of the numerical operations involved in the model checking algorithm.[21] However, by solving the equations symbolically, the algorithm would guarantee correct results.

There are two main applications of Euclidean model checking. First, it quantitatively verifies whether the model satisfies the specification; if a specification is not satisfied, an initial pmf witnessing a violation of the specification will be found. The second use is to compute system parameters that can achieve a desired behavior ψ: specifically, let ψ be a desired system behavior, then any counterexample $\mathbf{x}$ for the negated specification $\neg\psi$ would satisfy the desired behavior, that is, $\mathbf{x}, 0 \models \psi$. The initial state $\mathbf{x}(0)$ of the counterexample often contains important information about the system design.

One of the salient examples of the second usage is to find a scheme that probabilistically chooses an operation mode from a set of possible modes to achieve the desired goal. Observe that the Euclidean model checking process looks for an initial pmf whose trailing pmf trajectory would violate the specification. Thus, when a specification is violated, the counterexample contains information about how many portions of the nodes are in certain operation modes. First, observe that DTMCs are linear systems (Equation (2)) that respect the superposition principle. That is, for $\mathbf{x}(0) = \alpha \cdot \mathbf{x}^{(1)}(0) + (1-\alpha) \cdot \mathbf{x}^{(2)}(0)$, the trailing pmf transitions satisfy $\mathbf{x}(t) = \alpha \cdot \mathbf{x}^{(1)}(t) + (1-\alpha) \cdot \mathbf{x}^{(2)}(t)$ for $t \geq 0$ and $\alpha \in [0, 1]$. Suppose now that $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ are computation paths of DTMCs $\dot{D}^{(1)} = (S^{(1)}, \mathbf{M}^{(1)})$ and $\dot{D}^{(2)} = (S^{(2)}, \mathbf{M}^{(2)})$. Then $\mathbf{x}$ is a computation path of a mixed system that uses $D^{(1)}$ with fraction α and $D^{(2)}$ with fraction $1-\alpha$. In order to compute α, we extend the DTMCs $D^{(1)}$ and $D^{(2)}$ with noninteracting sink states[22] $f^{(1)}$ and $f^{(2)}$ to: $\dot{D}'^{(i)} = (S^{(i)} \cup \{f^{(i)}\}, \left[\begin{smallmatrix} \mathbf{M}^{(i)} & \mathbf{0} \\ \mathbf{0} & 1 \end{smallmatrix}\right])$ for $i = 1, 2$. Next, suppose that $D^{(1)}$ and $D^{(2)}$ have single initial states $i^{(1)}$ and $i^{(2)}$, respectively, and let ψ be our design goal. Then any counterexample witnessing

$$D'^{(1)} + D'^{(2)} \not\models \left(\begin{array}{l} \mathrm{P}\big[D'^{(1)} = i^{(1)}\big] + \mathrm{P}\big[D'^{(1)} = f^{(1)}\big] = 1 \wedge \\ \mathrm{P}\big[D'^{(2)} = i^{(2)}\big] + \mathrm{P}\big[D'^{(2)} = f^{(2)}\big] = 1 \wedge \\ \mathrm{P}\big[D'^{(1)} = i^{(1)}\big] + \mathrm{P}\big[D'^{(2)} = i^{(2)}\big] = 1 \end{array} \right) \rightarrow \neg\psi, \tag{3}$$

satisfies the design goal ψ and is of the form $\mathrm{P}[D'^{(1)}(0) = i^{(1)}] = \alpha$, $\mathrm{P}[D'^{(2)}(0) = i^{(2)}] = 1 - \alpha$, $\mathrm{P}[D'^{(1)}(0) = f^{(1)}] = 1 - \alpha$, $\mathrm{P}[D'^{(2)}(0) = f^{(2)}] = \alpha$, and zero for the other states. Observe that α is the parameter we are seeking: if we randomly choose $D^{(1)}$ with probability α and $D^{(2)}$ with probability $1 - \alpha$, then the desired goal ψ can be satisfied. The determination of parameters, such as α in this example, can be considered as a form of *statistical quantitative analysis*. This type of application is quite useful and has been employed in other places: for example, Kwon and Agha [2011], evaluated the reliability and the performance of a software system in a similar way[23], and in Kwon and Kim [2010], the drug-disposition changes in the body were model checked by considering the drug concentration levels as expected rewards.

Our model checking algorithm is sound, but it is complete under the following sufficient conditions. Let $\dot{D} = \sum_{i=1}^{m} \dot{D}^{(i)} = (S, \mathbf{M})$ be a DTMC and let $\lambda_1, \ldots, \lambda_{|S|}$ be the

[21]Note that because we are working with quantitative accumulative properties in systems, the likelihood of numerical error causing model checking giving faulty results can be reduced by adding some margin in the specification.

[22]If we choose mode $D^{(i)}$ with probability α, then the rest of the initial probability $1 - \alpha$ can be put at the noninteracting sink state $f^{(i)}$ so that the probability assigned to it does not affect the other states.

[23]Because the TDoA example of this article and the software reliability example in Kwon and Agha [2011] are both dealing with the reliability, methodologically they share some similarities. However, the model structure, the parameters, the source of errors, and the specifications of this article are carefully designed for the TDoA protocol.

eigenvalues of $\mathbf{M}$ such that $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_{|S|}|$, then these conditions are as follows.

(1) For all ap, "$\sum_i r_i \cdot \mathrm{P}[D(t_i) = s_i] \bowtie r$," $\sum_i r_i \cdot \mathrm{P}[D(\infty) = s_i] \neq r$.
(2) $|\lambda_{m+1}| < 1$.
(3) $\mathbf{M}$ is diagonalizable.

Condition (1) prevents the transient oscillating modes[24] of $\mathbf{M}$ from changing the truth value of the atomic propositions forever. This condition can be easily satisfied by slightly (e.g., less than the required accuracy) changing the atomic propositions. In practice, distinguishing the energy consumption level of 2.0 mJ and 2.000001 mJ may be meaningless in most measurement devices. Furthermore, the precision can be raised as necessary. Condition (2) ensures the existence of a unique limiting probability distribution. Although this is a commonly used assumption in the analysis of Markov processes, occasionally this condition is violated. However, in many cases, by adjusting the structure of the Markov process, (e.g., by merging multiple sink states into one), this condition can be satisfied. Regarding the last condition, matrices that are not diagonalizable are called defective matrices because they lack the full set of independent eigenvectors. Again, by slightly modifying the matrices, the lost eigenvectors can be regained [Strang 1988].

4.4. Performance Evaluation of a TDoA Protocol

We conclude this section with several performance evaluation metrics for the TDoA Markov chain model, as defined in Section 3.2.

An effective way to tune the performance metrics of the TDoA protocol is to change the filtering characteristics of the central node. In Figure 2, a good measurement can be accepted (the edge from state G to state S) or rejected (the edge from state G state to state I) depending on the filter. Similarly, a bad measurement can be accepted (the edge from state B to state F) or rejected (the edge from state B to state I). In general, using a stronger filter decreases the probability of false positives and increases the probability of false negatives. Thus, a stronger filter improves the reliability[25] of the protocol at the cost of increasing the time required and the energy consumed because of the increased number of retrials required to achieve the improved reliability. Another practical concern of the TDoA protocol is that one cannot indefinitely wait until all measurements are completed. Instead, the measurement has to be stopped when a certain percentage of the measurements have been completed in order to satisfy timeliness requirements.

In this example, we find a probabilistic strategy that increases the reliability of the protocol above a certain level and limits the retrials and the energy consumption levels to within certain limits. Specifically, we choose the original filter ($p_{FP} = 0.3$, $p_{FN} = 0.3$) with probability α and choose a strong filter ($p_{FP} = 0.2$, $p_{FN} = 0.4$) with probability $1 - \alpha$ and check the reported measurement using the chosen filter. We find probability α through Euclidean model checking from a counterexample of the negated goal. The desired performance metrics for the system are as follows.

(1) The reliability of the protocol is larger than 0.72, and at least 70% of the measurements are successful when the measurements are 85% complete.
(2) The expected total number of trials is less than 4.5, and 85% of the measurements use less than four trials on average.

[24]These modes correspond to complex eigenvalues whose absolute values are less than 1. Their imaginary parts are canceled by the modes corresponding to their complex conjugates. Also, because their absolute values are less than 1, they tend to 0 as the time proceeds.
[25]In this example, we define the reliability as the probability of accepting a good measurement.

(3) The expected total energy consumption is less than 28 units, 85% of the measurements use less than 25 units of energy, and the expected energy consumption during a sampling period never exceeds 1.8 units.

Let us begin the specification with a description about the probabilistic choices between the original system $\dot{D}^{(1)} = (S, \mathbf{M}^{(1)})$ and the system with the strong filter $\dot{D}^{(2)} = (S, \mathbf{M}^{(2)})$. Suppose that the probabilistic scheme is to choose the original filter with probability α and the strong filter with probability $1-\alpha$. As explained in the previous section, we can write a specification,[26] similar to the precondition of Equation (3), as

$$\psi_{init} = \big(\mathrm{P}[D^{(1)} = I] + \mathrm{P}[D^{(2)} = I] = 1\big) \wedge \big(\mathrm{P}[D^{(1)} = I] + \mathrm{P}[D^{(1)} = F] = 1\big) \wedge \big(\mathrm{P}[D^{(2)} = I] + \mathrm{P}[D^{(2)} = F] = 1\big),$$

and let the model checker find α. Unfortunately, this specification violates the first completeness condition, because $\mathrm{P}[D^{(k)}(\infty) = I] + \mathrm{P}[D^{(k)}(\infty) = F] = 1$ for $k = 1, 2$. To address this problem, we extended the DTMCs with state J, as in Figure 2, a noninteracting state that is sunk directly into state F with probability 1. That is, $\dot{D}^{(k)} = (S', \mathbf{M}^{(k)'})$ for $k = 1, 2$, where $S' = \{I, M, G, B, S, F, RI, RM, RR, J\}$ and $\mathbf{M}^{(k)'}_{i,j} = \mathbf{M}^{(k)}{}_{i,j}$ for $1 \le i, j \le 9$, $\mathbf{M}^{(k)'}{}_{6,10} = 1$, and $\mathbf{M}^{(k)'}_{i,j} = 0$ in other cases. With the extended DTMCs, the specification about the initial condition is

$$\psi_{init} : \big(\mathrm{P}[D^{(1)} = I] + \mathrm{P}[D^{(2)} = I] = 1\big) \wedge \big(\mathrm{P}[D^{(1)} = I] + \mathrm{P}[D^{(1)} = J] = 1\big) \wedge \big(\mathrm{P}[D^{(2)} = I] + \mathrm{P}[D^{(2)} = J] = 1\big).$$

Next, we model the reliability condition. It specifies the overall reliability and the reliability when 85% of the measurements are complete. The overall reliability can be simply written as $\psi_{r-limit} : \mathrm{Q}[D^{(1)} = S] + \mathrm{Q}[D^{(2)} = S] > 0.72$. That is, if $\mathbf{x}, 0 \models \psi_{r-limit}$, then $\sum_{\tau=0}^{\infty} \mathbf{x}(\tau)_5 + \mathbf{x}(\tau)_{15} > 0.72$. To specify the second reliability condition, let us first specify the 85% complete condition as $\psi_{85} : \mathrm{P}[D^{(1)} = F] + \mathrm{P}[D^{(2)} = F] \ge 1.85$. The reason the RHS of ψ_{85} is 1.85 instead of 0.85 is that the initial probabilities in state J are 1. The condition that at least 70% of the measured distances are good measurements can be written as $\psi_{r-85} : \mathrm{Q}[D^{(1)} = S] + \mathrm{Q}[D^{(2)} = S] < 0.125$. This means that if $\mathbf{x}, t \models \psi_{r-85}$, then $\sum_{\tau=t}^{\infty} \mathbf{x}(\tau)_5 + \mathbf{x}(\tau)_{15} < 0.125$. Thus, if also $\mathbf{x}, 0 \models \psi_{r-limit}$, then $\sum_{\tau=0}^{t-1} \mathbf{x}(\tau)_5 + \mathbf{x}(\tau)_{15} > 0.595$, meaning that more than 70% of the measurements (0.595 out of 0.85) are good measurements. With ψ_{85} and ψ_{r-85}, the second condition can be specified as $\neg\psi_{85}\,\mathtt{U}\,\psi_{r-85}$. That is, until the remaining reliability gain becomes less than 0.125, the measurement is not complete more than 85%. To summarize, the reliability condition can be written as

$$\psi_r : \psi_{r-limit} \wedge (\neg\psi_{85}\,\mathtt{U}\,\psi_{r-85}).$$

Let us turn to the condition about the expected number of trials. It states that the total expected number of trials is less than 4.5 and that the 85% complete condition can be reached before the fourth trial. The condition about the total expected number of trials can be specified as $\psi_{t-limit} : \mathrm{Q}[D^{(1)} = I] + \mathrm{Q}[D^{(2)} = I] < 4.5$. For the second condition, let us define an atomic proposition $\psi_{t-85} : \mathrm{Q}[D^{(1)} = I] + \mathrm{Q}[D^{(2)} = I] > 0.5$. Now, suppose that $\mathbf{x}, 0 \models \psi_{t-limit}$ and $\mathbf{x}, t \models \psi_{t-85}$ at some time t. Then $\sum_{k=0}^{t-1} \mathbf{x}(k)_1 + \mathbf{x}(k)_{11} < 4$. In other words, the expected number of trials up to $t-1$ is less than four.

[26] We do not need to extend the DTMCs here with a non-interacting sink state because they already have one (F).

Because this should occur before the measurement is 85% complete, this condition can be expressed as $\psi_{85}\,\mathrm{R}\,\psi_{t-85}$. In other words, before the measurement is 85% complete, the number of trials is less than four. Thus, the conditions about the number of trials can be expressed as

$$\psi_t \;:\; \psi_{t-limit} \wedge (\psi_{85}\,\mathrm{R}\,\psi_{t-85}).$$

Finally, the condition about the expected energy consumption has three subconditions: (1) the total expected energy consumption to finish the measurement is less than 28 units of energy; (2) the expected energy consumption until 85% of the measurement is complete is less than 25 units; and (3) the energy consumption level during a sampling is always less than 1.8 units. The first two conditions can be expressed similarly, as in the previous specification of the number of trials. To simplify the descriptions, let us define an auxiliary function: $e : \{\mathrm{P}, \mathrm{Q}\} \times \{D^{(1)}, D^{(2)}\} \to \mathbb{R}$ such that $e(\mathrm{P_{op}}, D) = \mathrm{P_{op}}[D = I] + 3 \cdot \mathrm{P_{op}}[D = M] + \mathrm{P_{op}}[D = G] + \mathrm{P_{op}}[D = B]$, where $\mathrm{P_{op}} \in \{\mathrm{P}, \mathrm{Q}\}$. Let two atomic propositions $\psi_{e-limit}$ and ψ_{e-85} be $\psi_{e-limit} : e(\mathrm{Q}, D^{(1)}) + e(\mathrm{Q}, D^{(2)}) < 28$ and $\psi_{e-85} : e(\mathrm{Q}, D^{(1)}) + e(\mathrm{Q}, D^{(2)}) > 3$. Using the auxiliary function definition, the first two conditions can be written as $\psi_{e-limit} \wedge (\psi_{85}\,\mathrm{R}\,\psi_{e-85})$. The third condition can be specified, using the always operator, as $\Box\,\psi_{e-sample}$, where $\psi_{e-sample}$ is $e(\mathrm{P}, D^{(1)}) + e(\mathrm{P}, D^{(2)}) < 1.8$. Thus, requirements for the expected energy consumption can be specified as

$$\psi_e \;:\; \psi_{e-limit} \wedge (\psi_{85}\,\mathrm{R}\,\psi_{e-85}) \wedge \Box\,\psi_{e-sample}.$$

Now, with all subconditions specified, we can write an iLTL formula whose counterexample, if it exists, tells us the probability for choosing the strong filter. Combining the subspecifications together, we can build an iLTL formula

$$\psi \;:\; \psi_{init} \to \neg(\psi_r \wedge \psi_t \wedge \psi_e).$$

Observe that any counterexample $\mathbf{x}$ that violates ψ satisfies the original requirements. That is, $\mathbf{x}, 0 \models \psi_{init} \wedge \psi_r \wedge \psi_t \wedge \psi_e$. Model checking ψ shows that $D^{(1)} + D^{(2)} \not\models \psi$ with a counterexample of $\mathrm{P}[D^{(1)}(0) = I] = \mathrm{P}[D^{(2)}(0) = J] = 0.6003$, $\mathrm{P}[D^{(2)}(0) = I] = \mathrm{P}[D^{(1)}(0) = J] = 0.3997$, and zeros for the other states. That is, if we randomly choose the original filter at 60.03% of the time and the strong filter at 39.97% of the time, then the conditions ψ_r, ψ_t, and ψ_e, are all satisfied.

Figure 6 shows the transitions of the expected rewards beginning from the initial state found from the model checking. In this figure, the vertical-axis for the dashed lines is on the left, for the solid lines on the right-hand side of the graphs. The solid line of the first graph is the sum of the probabilities that $D^{(1)}$ and $D^{(2)}$ are in F state. The jump at Step 0 is due to the remaining probabilities initially put in state J. From this graph, we see that the measurement is 85% complete at Step 48. The dashed line shows the expected number of trials. This count converges to a number below 4.5 and is less than 4 at Step 48. The second graph shows how the probability of success changes. It converges to a value larger than 0.72 and is larger than 0.595 at Step 48. The expected energy consumption levels are plotted in the last graph. It shows that the total accumulated energy consumption (solid line) is less than 28 units and is less than 25 units when the measurement is 85% complete. The dashed line confirms that the expected energy consumption during a sampling period never exceeds 1.8 units.

5. SYSTEM MODEL ESTIMATION

To make the performance evaluation results reflect the behaviors of a real system accurately, the mathematical model used in the performance evaluation must adequately reflect the parameters of the real system. In this section, we present a DTMC parameter estimation method from the samples of a real system so that performance evaluation

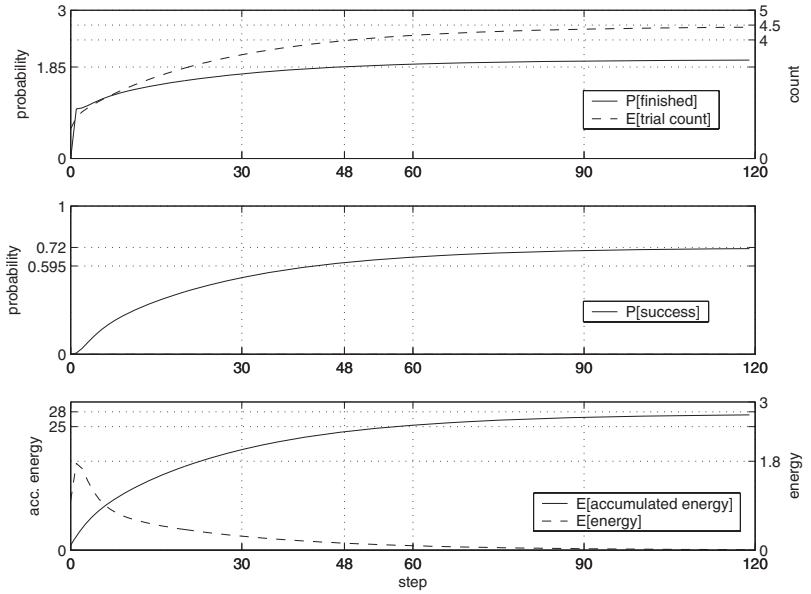


Fig. 6. The changes of the expected rewards beginning from the counterexample.

can be performed on a realistic model. We propose a Markov transition matrix estimation method based on quadratic programming (QP) [Luenberger 1989]. We also develop a statistical test that can reject estimated models which do not represent the behavior of a real system.

5.1. DTMC Model Estimation

A typical WSN comprises a large number of nodes, which makes the fractions of the nodes in certain states an accurate estimation of the global state as represented by a pmf. In this section, we find the state transition dynamics of the system from a sampled series of the pmf estimations. Note that the memoryless property of DTMCs ensures that the current pmf estimate depends only on its immediate past. Thus, the probability transition matrix we are seeking is the one that can optimally[27] match every pair of consecutive pmf estimates.

Let $\dot{D} = (S, \mathbf{M})$ be the true DTMC of a system, then like the standard system identification techniques, we compute an estimated DTMC $\hat{\dot{D}} = (S, \hat{\mathbf{M}})$ such that $\hat{\mathbf{M}}$ minimizes the squared sum of differences between the sampled pmf $\bar{\mathbf{x}}(t+1)$ and the predicted pmf $\hat{\mathbf{x}}(t+1)$ given $\bar{\mathbf{x}}(t)$. We estimate the probability $\bar{\mathbf{x}}(t)_i = \mathrm{P}[D(t) = s_i]$ as the fraction of the number of nodes in s_i state over the total number of nodes ns.

Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ be a matrix of the point estimates such that $P_{ij} = \bar{\mathbf{x}}_j(i)$, where m is the number of pmf samples and n is the number of states. We estimate a Markov matrix $\hat{\mathbf{M}} \in \mathbb{R}^{n \times n}$ such that it minimizes the Euclidean distances between a sampled pmf $\bar{\mathbf{x}}(t+1)$ and a one-step predicted pmf $\hat{\mathbf{x}}(t+1)$ given $\bar{\mathbf{x}}(t)$, which is $\hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t)$.

$$E = \min_{\hat{\mathbf{M}}} \sum_{t=0}^{m-2} |\bar{\mathbf{x}}(t+1) - \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t)|^2.$$

[27] In the sense that it minimizes an error function.

Let $\mathbf{P}_c$ and $\mathbf{P}_f$ be the submatrices of $\mathbf{P}$ with the first and the last $m-1$ rows respectively. Then, from the conditions $\frac{\partial E}{\partial \hat{\mathbf{M}}_{ij}} = 0$ for $i, j = 1, \ldots, n$, the matrix $\hat{\mathbf{M}}$ that minimizes E is

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

However, to be a Markov matrix, $\hat{\mathbf{M}}$ should satisfy the constraints that (1) $0 \leq \hat{\mathbf{M}}_{ij} \leq 1$ for all $1 \leq i, j \leq n$ and (2) $\sum_{i=1}^{n} \hat{\mathbf{M}}_{ij} = 1$ for all $1 \leq j \leq n$. As a relatively straightforward solution, we minimize E subject to these constraints by quadratic programming [Luenberger 1989]. That is, $\hat{\mathbf{M}}$ can be obtained by

$$\mathbf{z} = \operatorname{argmin}\left(\frac{1}{2} \cdot \mathbf{z}^T \cdot \mathbf{H}^T \cdot \mathbf{H} \cdot \mathbf{z} - \mathbf{f}^T \cdot \mathbf{H} \cdot \mathbf{z}\right)$$
$$\text{subject to} \quad \mathbf{A} \cdot \mathbf{z} \leq \mathbf{b} \text{ and } \mathbf{C} \cdot \mathbf{z} = \mathbf{d},$$

where

$$\mathbf{z} \in \mathbb{R}^{n^2 \times 1} = \left[\hat{\mathbf{M}}_{*1}^T, \ldots, \hat{\mathbf{M}}_{*n}^T\right]^T, \qquad \mathbf{A} \in \mathbb{R}^{2n^2 \times n^2} = \left[\mathbf{I}_{n^2}, -\mathbf{I}_{n^2}\right]^T,$$
$$\mathbf{b} \in \mathbb{R}^{2n^2 \times 1} = \left[\mathbf{1}_{1,n^2}, \mathbf{0}_{1,n^2}\right]^T,$$
$$\mathbf{H} \in \mathbb{R}^{n(m-1) \times n^2} = \begin{bmatrix} \mathbf{P}_c & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{P}_c \end{bmatrix}, \qquad \mathbf{C} \in \mathbb{R}^{n \times n^2} = \begin{bmatrix} \mathbf{1}_{1,n} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{1}_{1,n} \end{bmatrix},$$
$$\mathbf{f} \in \mathbb{R}^{n(m-1) \times 1} = \left[\mathbf{P}_{f*1}^T, \ldots, \mathbf{P}_{f*n}^T\right]^T, \qquad \mathbf{d} \in \mathbb{R}^{n \times 1} = \mathbf{1}_{n,1},$$

where $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is an identity matrix, $\mathbf{1}_{m,n} \in \mathbb{R}^{m \times n}$ is a matrix of ones, $\mathbf{0}_{m,n} \in \mathbb{R}^{m \times n}$ is a matrix of zeros, and $\hat{\mathbf{M}}_{*i}$ and $\mathbf{P}_{f*i}$ are the ith columns of $\hat{\mathbf{M}}$ and $\mathbf{P}_f$.

It is known that if the $\mathbf{Q}$ matrix (corresponding to $\mathbf{H}^T \cdot \mathbf{H}$) of QP is positive semidefinite, QP becomes a convex optimization problem, and the computational complexity of QP becomes polynomial if $\mathbf{Q}$ is positive definite when the ellipsoid method [Khachiyan 1979] is used. However, if $\mathbf{Q}$ is indefinite or has at least one negative eigenvalue, its computational complexity becomes NP-hard [Pardalos and Vavasis 1991]. In the preceding estimation algorithm, because $\mathbf{H}^T \cdot \mathbf{H}$ is at least positive semidefinite,[28] the estimation problem is a convex optimization problem. Furthermore, if $\mathbf{P}_c$ has full rank, then $\mathbf{H}^T \cdot \mathbf{H}$ becomes positive definite[29] and the problem can be solved in polynomial time. Observe that when enough samples are collected, if random noise is present in the sample, it is likely that $\mathbf{P}_c$ will have the full rank even though the matrix corresponding to the true pmf samples does not.

When estimating the probability transition matrix, we assume that (1) the states of the DTMC model are already known and (2) the underlying systems are identical. Regarding the first assumption, there are different intuitive choices of the states that work well. Some possible choices are the states of a protocol, the modules of a program, or the process states of a running application. Depending on which choice is made, different types of analysis can be performed. With respect to Assumption (2), the interpretation of the term *underlying systems* is not as restrictive as it may seem: we do not necessarily refer to individual sensor nodes or to the applications running on them. As can be seen in the TDoA protocol in Section 3.2, the underlying system may involve several processes running on different nodes. Moreover, the identicalness does

[28]Suppose that $\mathbf{H}^T \cdot \mathbf{H} \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$, then $\mathbf{v}^T \cdot \mathbf{H}^T \cdot \mathbf{H} \cdot \mathbf{v} = (\mathbf{H} \cdot \mathbf{v})^T \cdot (\mathbf{H} \cdot \mathbf{v}) = \lambda \cdot \mathbf{v}^T \cdot \mathbf{v}$. Because $(\mathbf{H} \cdot \mathbf{v})^T \cdot (\mathbf{H} \cdot \mathbf{v}) \geq 0$ and $\mathbf{v}^T \cdot \mathbf{v} \geq 0$, $\lambda \geq 0$.

[29]$\mathbf{H}^T \cdot \mathbf{H}$ is a block diagonal matrix of $\mathbf{P}_c^T \cdot \mathbf{P}_c$. If $\mathbf{P}_c$ is full rank, because the columns of $\mathbf{P}_c$ are independent, $\mathbf{P}_c \cdot \mathbf{v} \neq \mathbf{0}$ and $\mathbf{P}_c^T \cdot \mathbf{P}_c \cdot \mathbf{v} \neq \mathbf{0}$ unless $\mathbf{v} = \mathbf{0}$. Thus, if $\mathbf{P}_c$ is full rank, 0 is not an eigenvalue of $\mathbf{H}^T \cdot \mathbf{H}$.

not mean that all the nodes are running the same task at the same time; it means that the runs of the processes, possibly at different times, are identically generated. For example, the TDoA protocol may run several times between different sets of nodes while other nodes are running other tasks. When the samples of the runs are collected, our assumption is that they were generated by identical stochastic processes.

Our proposed estimation method does not work when the constituent processes are not identical. For example, when different operation modes are present in a WSN, the set of states may need to be extended. A common situation is that the nodes on the edge of a WSN behave differently than the nodes inside a WSN. One solution for estimating these multimode systems is to estimate separate DTMCs for different modes and combine them together, for example, using the Kronecker operators or SAN [Langville and Stewart 2004; Lam et al. 2004]. However, in general, identifying the hidden modes could be a complicated process. In the next section, we propose a statistical test method that could reject the incorrectly estimated DTMCs.

A pth-order DTMC is a Markov chain whose current state depends on its past p states. That is, $\mathrm{P}[D(k) = s_k | D(k-1) = s_{k-1}, \ldots, D(0) = s_0] = \mathrm{P}[D(k) = s_k | D(k-1) = s_{k-1}, \ldots, D(k-p) = s_{k-p}]$. With a little modification, the proposed estimation method can also be applied to high-order DTMCs. First, the set of system states needs to be extended to its p cross products. For example, the extended states of a second-order DTMC with two states a and b are (a,a), (a,b), (b,a), and (b,b). The sample matrix $\mathbf{P}$ can be easily reconstructed according to the new definition of the states. Second, there are constraints about prohibited state transitions. For example, in the second-order DTMC example, state transitions like $(a,a) \to (b,b)$ or $(a,a) \to (b,a)$ are prohibited: a should appear in the past state of the new states. This restriction can be easily enforced by simply removing these elements from the $\mathbf{z}$ vector(they are zeros in $\mathbf{M}$)and adjusting the other matrices accordingly.

5.2. DTMC Test Method for Estimated DTMCs

The Markov chain estimation method of the previous section always finds a probability transition matrix that optimally matches the transitions of two consecutive samples of pmfs. However, we still do not know whether the optimal model can accurately represent the real system. In this section, we propose a statistical test method for 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 H_0: the system is a DTMC whose probability transition matrix is the same as the estimated matrix,[30] and an alternative hypothesis H_a: the system is not a DTMC or the estimated probability transition matrix is different from the real matrix. Using these hypotheses, the outline of the proposed DTMC test procedure is as follows. (1) Under H_0, the sampled pmfs $\bar{\mathbf{x}}(t)$ can be accepted or rejected by the χ^2 test with the predicted pmfs $\hat{\mathbf{x}}(t) = \hat{\mathbf{M}} \cdot \bar{\mathbf{x}}(t-1)$ and a significance level β, and (2) if there is a β by which the number of rejected samples is too high for a given significance level α, we reject H_0 and accept H_a; otherwise we accept H_0.

Let $Y_i(t)$ and $K_i(t)$ be *random variables* (rvs) defined as

$$Y_i(t) = \begin{cases} 1 & \text{if } D(t) = s_i \\ 0 & \text{otherwise} \end{cases}, \quad K_i(t) = \sum_{j=1}^{ns} Y_i^{(j)}(t),$$

[30]This hypothesis is rather strong and it may be possible to further relieve it as an ϵ–approximation of the estimated matrix with respect to a matrix norm, such as the Frobenius norm.

where $Y_i^{(j)}(t)$ for $j = 1, \ldots ns$ are independent and identically distributed (iid) rvs whose distributions are identical to $Y_i(t)$. Then, $K_i(t)$, representing the number of nodes in s_i state, has a binomial distribution: $\mathrm{P}[K_i(t) \leq n] = \mathrm{Bin}\left(n, ns, \mathbf{x}(t)_i\right)$, where $\mathrm{Bin}\,(n, m, p) = \sum_{i=0}^{n} \binom{m}{i} \cdot p^i \cdot (1-p)^{m-i}$.

A point estimator $q(t)$, known as Pearson's test statistic, and its point estimate $\bar{q}(t)$ is as follows.

$$q(t) = \sum_{i=1}^{n} \frac{\left(K_i(t) - ns \cdot \hat{\mathbf{x}}(t)_i\right)^2}{ns \cdot \hat{\mathbf{x}}(t)_i}, \quad \bar{q}(t) = \sum_{i=1}^{n} \frac{\left(ns \cdot \bar{\mathbf{x}}(t)_i - ns \cdot \hat{\mathbf{x}}(t)_i\right)^2}{ns \cdot \hat{\mathbf{x}}(t)_i}.$$

It is well known that the rv $q(t)$ has a χ^2 distribution with $n-1$ degrees of freedom[31] [Papoulis 1991]. Thus, for a significance level β, we reject the sample $\bar{\mathbf{x}}(t)$ if $\bar{q}(t)$ is in the critical region of $q(t)$. In other words, let χ_z^2 be the value of y when $\chi^2(y) = z$, then we reject $\bar{\mathbf{x}}(t)$ with the probability of the type–I error β when $\bar{q}(t) \geq \chi_{1-\beta}^2$.

In order to bring the whole m samples into the consideration, we define another rv B_β-an estimator for the number of rejected samples—as follows.

$$B_\beta = \sum_{t=1}^{m-1} \delta_{\chi_{1-\beta}^2}\left(q(t)\right) \ , \text{ where } \delta_\theta(x) = \begin{cases} 1 & \text{if } x \geq \theta; \\ 0 & \text{otherwise.} \end{cases} .$$

Observe that $\mathrm{E}[\delta_{\chi_{1-\beta}^2}\left(q(t)\right)] = \mathrm{P}[q(t) > \chi_{1-\beta}^2] = \beta$, that is, the probability that a sample $\bar{\mathbf{x}}(t)$ is erroneously rejected. Because there are $m-1$ samples, B_β has a binomial distribution of order $m-1$ and probability β. Let its point estimate $\bar{B}_\beta$ for the number of rejected samples be $\bar{B}_\beta = \sum_{t=1}^{m-1} \delta_{\chi_{1-\beta}^2}\left(\bar{q}(t)\right)$. Then, given a significance level α, we reject the hypothesis H_0 and take H_a if and only if $\mathrm{Bin}\left(\bar{B}_\beta, m-1, \beta\right) \geq 1-\alpha$ for some $\beta \in (0, 1)$.

One difficulty here is that β is a real number which we cannot enumerate. However, because B_β takes only finite number of values, $0 \cdots m-1$, we can still perform the DTMC test. For all $k \in \{1, \ldots, m-2\}$, let β_k be the least upper bound of the significance level for the χ^2 test such that H_0 is still acceptable with a significance level α when k samples are rejected. That is, $\beta_k = \sup(\{\beta : \mathrm{Bin}\,(k, m-1, \beta) < 1-\alpha\})$. The two extreme cases are $k = 0$, meaning no samples are rejected, and $k = m-1$, meaning all samples are rejected. These are trivially true because the significance levels in these cases are, respectively, 1 and 0. Now we check if there is a $k \in \{1, \ldots, m-2\}$ such that the χ^2 test rejects more than k samples with a significance level β_k. In other words, we accept H_0 if and only if $|\{t : \bar{q}(t) \geq \chi_{1-\beta_k}^2\}| \leq k$ for $k = 1, \ldots, m-2$.

6. EXPERIMENTS

Now we demonstrate the usefulness of our performance evaluation framework by analyzing performance metrics of a real WSN consisting of 90 nodes. We estimate a DTMC from the samples of the WSN and statistically test the validity of the estimated model. Against the estimated model, we systematically check the performances of the system through Euclidean model checking. Our experimental platform comprises 90 Mica-2 motes.

Each mote has an ATmega 128L low-power 8-bit CPU running at 8MHz clock speed, 4KByte of SRAM, 128KByte of program flash, and 512KByte of serial flash memory.

[31]In this section, we omit the degrees of freedom term in the χ^2 distribution to simplify the notation. That is, χ^2 means χ^2 with $n-1$ degrees of freedom.

Table II. Energy Consumption Level of a Mica-2 Mote

	Processor	Radio	
		transmit	receive
Active	8 mA	25 mA	8 mA
Sleep	$< 15\ \mu$A	$< 1\ \mu$A	

Although a mote has relatively large serial flash memory, this memory is slow, especially for write operations. Mica-2 is also equipped with a CC1000 radio transceiver. At the lower-level communication layer, Manchester encoding is used, and we can achieve a theoretical throughput of 38.4Kbps. In order to save energy, an application can go to sleep mode. The amount of energy spent in various modes is summarized in Table II.[32]

TinyOS is an operating system running on Mica-2 nodes.[33] 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 for a long time should be written in a *task* block. TinyOS schedules tasks by managing a queue of nonpreemptable task blocks. Thus, spin-looping in a task block will block all operations of TinyOS.

6.1. Application Scenario and Program

We implemented the program in Figure 7. This application program samples a microphone and stores the results in a buffer (sampling code is not shown). Whenever the `run` task is scheduled, it computes a Fourier transform coefficient of the mic sample to detect the buzzer. After the coefficient is computed, in order to reduce collisions, the application sends the result to a base station with probability 0.05. To make the 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[34] 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. Considering the small SRAM size, we compress the sample data by a run length encoding algorithm [Sedgewick 1990].

As one can see from the $\hat{\mathbf{M}}$ matrix (next section), 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 the sampling speed increases so does the probability that a process remains in its current state. That means, the run length encoding algorithm performs better with high-frequency sampling. For this experiment, we configure every node within a radio communication range of a base station. Because we do not need multihop message forwarding, we turn off the radio communication channel except when a node is sending a message. This saves energy which might otherwise be consumed by unnecessary messages.

[32]Crossbow Technology, Inc. `http://xbow.com`.
[33]TingOS. `http://thingos.net`.
[34]It is about 1/4th of the maximum sampling rate. We did not choose the maximum sampling rate in order to not affect the system characteristics.

```
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);
}
```

Fig. 7. Abbreviated experimental program. The `state` variable keeps track of the state of the system. There is a timer interrupt routine that samples the `state` value at a regular interval.

6.2. 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 7. The state of a process 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 the same time and compute a sequence of estimated pmfs.

The solid lines of Figure 8 show the transitions of a sampled probability distribution. The jitters in this figure are due to the small sample population size. From this sequence of pmf estimations, we estimate a DTMC $\dot{A} = (S, \hat{\mathbf{M}})$, where $S = \{Ready, Run, Wait\}$ and

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

The dashed lines of Figure 8 show pmf transitions of the estimated DTMC A. We plotted the estimated pmf transition graphs starting from the initial sample pmf. We applied the DTMC test method of Section 5 to the estimated DTMC A and the pmf samples. Our estimated model passed the test with a significance level of $p < 0.05$.

6.3. Performance Evaluation

We developed a model checking tool called *iLTLChecker* that implements our Euclidean model checking algorithm [Kwon and Agha 2005]. It takes a text description about a list of DTMC models and an iLTL specification and checks if the model satisfies the specification. A list of initial pmfs corresponding to each individual DTMC is printed out as a counterexample when the model does not satisfy the specification. In order to prevent users from waiting indefinitely for the model checking results, the checker computes and notifies a search depth before the main model checking process begins. Specific implementation details about iLTLChecker are as follows: a scanner and a parser generator for the text input processing are generated from a Lex and a Yacc tools;

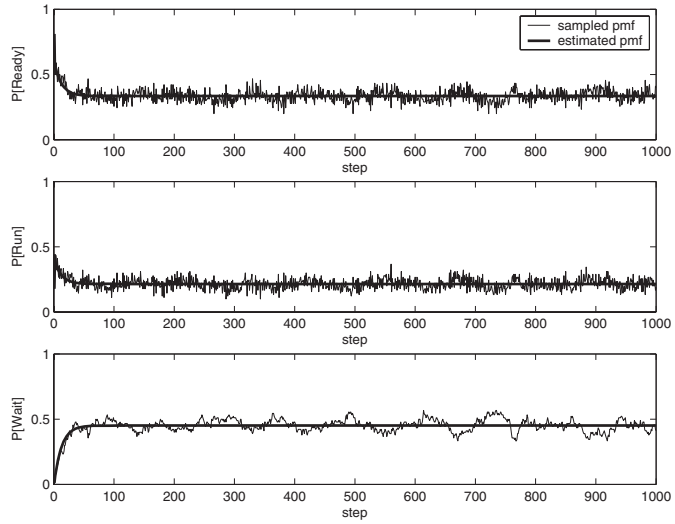


Fig. 8. Sampled pmf transitions from a 90-node experiment (thin lines) and predicted pmf transitions of an estimated Markov chain (thick lines).

the LAPACK library is imported to compute the matrix diagonalization; and a simplex method that can handle equality and strict and nonstrict inequality constraints is implemented in the C language. The checker builds a Büchi automaton for the negated specification and traverses the automaton in a depth first search manner. It tries to find a run of length of the search depth while checking the feasibility of the (in)equalities collected along the run on each step.

Figure 9 shows an iLTLChecker description for this experiment. In the description, any contents from # to the end of the line are considered as comments and are ignored. The description has two main blocks of a model description block that begins with a `model:` tag and a specification block that begins with a `specification:` tag. The description can also have an optional identifier definition block that begins with a `var:` tag. In the model description block, a multiple DTMC model is defined as a ',' separated list of DTMCs, where each DTMC has a name, a set of states, and a probability transition matrix. In the specification block, an optional list of atomic propositions is defined first. Each atomic proposition is an equality or inequality about expected rewards of the DTMCs defined in the model description block. Finally, using atomic propositions, an iLTL specification ends the description.

In Figure 9, we define three DTMCs: A for normal mode, B for slow mode, and C for combined mode. A is the model we estimated from the real system in the previous section. The states *Re, Ru,* and *Wa* stand for the *Ready, Run,* and *Wait* states. B is a computed model obtained from A by reducing the probabilities from the *Ready* state to the other states by half. That is, this task yields more to the other tasks. From this change we can expect that B itself will consume less energy.[35] C is a combined system of A and B such that during the *Run* state, the task can change its mode to the normal mode with probability toA or to the slow mode with probability toB $(= 1 - toA)$. The states *Rea, Rua*, and *Waa* are *Ready, Run*, and *Wait* states of the normal mode, and the states *Reb, Rub*, and *Wab* are the corresponding states of the slow mode.

[35]While a task is in *Ready* state, other tasks are running. Thus, the energy consumption in this state is not due to this task.

```
var:
  # transition to normal mode
    toA = 1, toB = 1-toA,
  # transition to slow mode
  # toA = 0, toB = 1-toA,
  # transition to combined mode
  # toA = 0.683, toB = 1-toA,
  # time bounds for transitions
    ta = 37, tb = 99, tc = 38

model:
  # Normal mode
  Markov chain A
    has states:
      { Re, Ru, Wa},
    transits by :
      [  .4691, .7383, .0435;
         .4827, .2455, .0000;
         .0482, .0162, .9565   ],

  # Slow mode
  Markov chain B
    has states:
      { Re, Ru, Wa},
    transits by :
      [  .7346, .7383, .0435;
         .2413, .2455, .0000;
         .0241, .0162, .9565   ],

  # Combined mode
  Markov chain C
    has states:
      { Rea, Rua, Waa, Reb, Rub, Wab},
    transits by :
[.469,toA*.738, .044, .0,  toA*.738, .0;
 .483,toA*.246, .000, .0,  toA*.246, .0;
 .048,toA*.016, .956, .0,  toA*.016, .0;
 .0,  toB*.738, .0,   .735,toB*.738, .044;
 .0,  toB*.246, .0,   .241,toB*.246, .000;
 .0,  toB*.016, .0,   .024,toB*.016, .956]
```

```
specification:
  ### Availability
  aa1: P[A=Ru] > 0.214,
  aa2: P[A=Ru] < 0.215,
  ab1: P[B=Ru] > 0.160,
  ab2: P[B=Ru] < 0.161,

  ### Energy
  ea1: 8*P[A=Ru] + 33*P[A=Wa] > 16.591,
  ea2: 8*P[A=Ru] + 33*P[A=Wa] < 16.593,
  eb1: 8*P[B=Ru] + 33*P[B=Wa] > 12.429,
  eb2: 8*P[B=Ru] + 33*P[B=Wa] < 12.431,

  ### Mode transition
  # Is availability close to normal mode?
  toa: P[C(ta)=Rua] + P[C(ta)=Rub] > 0.2,
  # Is energy consumption close to slow mode?
  tob: 8*P[C(tb)=Rua]+33*P[C(tb)=Waa]+
       8*P[C(tb)=Rub]+33*P[C(tb)=Wab]<12.5,
  # Initial modes
  a: P[C=Rea] + P[C=Rua] + P[C=Waa] = 1,
  b: P[C=Reb] + P[C=Rub] + P[C=Wab] = 1,

  ### System design
  # Required availability
  ac:  P[C(tc)=Rua] + P[C(tc)=Rub] > 0.18,
  # Required energy consumption
  ec:  8*P[C=Rua] + 33*P[C=Waa] +
       8*P[C=Rub] + 33*P[C=Wab] < 15,

  ### Specifications
  #1. A's availability
      <> [] ~(aa1 /\ aa2)
  #2. B's availability
  #   <> [] ~(ab1 /\ ab2)
  #3. A's energy consumption
  #   <> [] ~(ea1 /\ ea2)
  #4. B's energy consumption
  #   <> [] ~(eb1 /\ eb2)

  #5. slow mode to normal mode
  #   b -> [] toa
  #6. normal mode to slow mode
  #   a -> [] tob

  #7. required availability
  #   [] ac
  #8. required energy
  #   <> [] ~ec
```

Fig. 9. An iLTLChecker description of the estimated DTMC model and specifications.

In this experiment, we assume that when an interesting event has occurred, the process switches its mode to the normal mode by setting $toA = 1$, and later, when the alert period is over, it switches back to its usual operation mode—the slow mode ($toA = 0$) or the combined mode ($0 < toA < 1$).

The specification block begins with the definitions of the atomic propositions to be used in the specification. First, we define the *availability* of a process, say X, as the probability that the process is in *Run* state: $\mathrm{P}[X = Run]$.[36] From Table II, the expected energy consumption values are 33 (mA) in *Wait* state (25 for the radio + 8 for the processor), 8 (mA) in *Run* state, and zero in *Ready* state, as the energy is consumed by other tasks. Thus, the expected energy consumption during a sampling period is proportional to $8 \cdot \mathrm{P}[X = Run] + 33 \cdot \mathrm{P}[X = Wait]$. Finally, to evaluate the delay in the mode switches, we define two atomic propositions about the initial conditions that (1) all tasks are in the normal mode $\mathrm{P}[C = Rea] + \mathrm{P}[C = Rua] + \mathrm{P}[C = Waa] = 1$, and (2) all tasks are in the slow mode $\mathrm{P}[C = Reb] + \mathrm{P}[C = Rub] + \mathrm{P}[C = Wab] = 1$.

As a first performance evaluation example, we find the availability in the steady state. This can be done by a binary search guided by the model checking results. The first specification of Figure 9, $\diamond\,\square\,\neg(aa1 \wedge aa2)$, specifies that the availability of A is not in the range (0.214, 0.215) in the steady state ($\diamond\,\square$). The model checking result is false. Thus, as a next step, we examine the interval (0.214, 0.2145) and so on. In general, model checking finishes early when the result is false, because once the checker finds a counterexample, it can skip the rest of the search space. Therefore, if the checking does not finish fast enough, one can switch the interval or remove the negation operator. The actual model checking result of the specification is as follows.

```
Depth:  84
Result: F
counterexample:
    pmf(A(0)): [ 0.335 0.214 0.451 ]
    pmf(B(0)): [ 1.000 0.000 0.000 ]
    pmf(C(0)): [ 1.000 0.000 0.000 0.000 0.000 0.000 ].
```

In the result, the first line `Depth:84` shows the search depth, that is, the maximum number of time steps iLTLChecker needs to search to determine the result. Because iLTLChecker prints this value before actual search begins, we can modify the specification if this value is too large. The second line `Result:F` means that the multiple DTMC model is not a model of the specification, and the last three lines are the counterexample: the initial pmfs for A, B, and C. Similarly, from the second specification (#2 of Figure 9), the steady state availability of B is in [0.160, 0.161]. As expected, the availability is reduced in the slow mode. We further compute the expected energy consumption levels of the two modes in the steady state by the third and the fourth specifications (#3 and #4 of Figure 9). These values are 16.592 (mA) for A and 12.430 (mA) for B.

As a second set of examples, we examine several time bounds for the mode switches. First, we compute the maximum delay to switch from the slow mode to the combined mode, whose availability is larger than 0.2. That is, we find the maximum lower bound t such that $\forall t' \geq t\ .\ \mathrm{P}[C(t') = Rua] + \mathrm{P}[C(t') = Rub] > 0.2$ with `toA=1`. The precondition that all tasks are in the slow mode can be written as $\mathrm{P}[C(0) = Reb] + \mathrm{P}[C(0) = Rub] + \mathrm{P}[C(0) = Wab] = 1$. Observe that this constraint forms a surface in the space of initial pmfs and the mode transition time depends on each point on the surface. The time bound can be found by the binary search on `ta` based on the model checking results for $b \rightarrow \square\, toa$ (#5 of Figure 9). In the specification, the always operator $\square$ and the time offset `ta` in `toa` are used to specify "from `ta` onward `toa` is true". After several trials, we found that the longest mode transition time is 37 steps (148 ms). The first graph of Figure 10 shows the availability (solid line) and the energy consumption level (dashed

[36]Only the processes in *Run* state can handle events reliably: in *Wait* state, the high-priority radio will block low-priority events, and in *Ready* state, event handling depends on the other processes running at the moment.

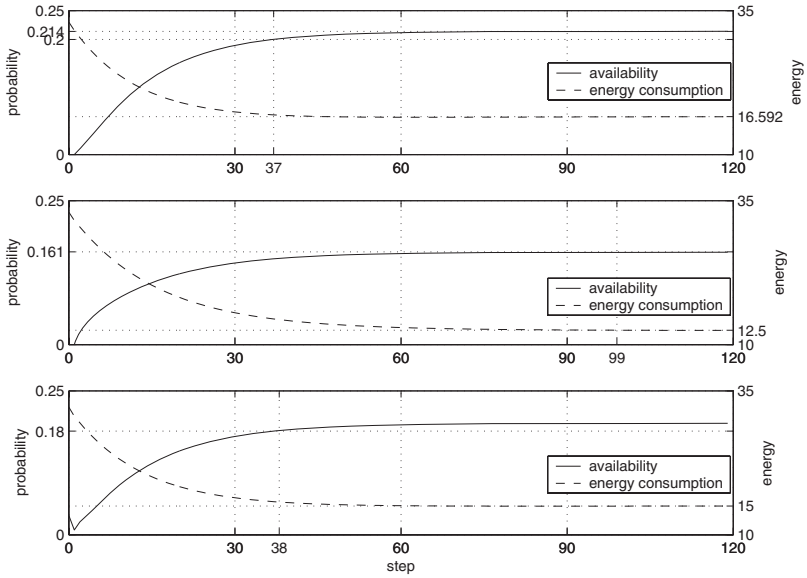


Fig. 10. The availability and the expected energy consumption level changes during mode switches.

line) changes during this mode switch from the counterexample when we set `ta` to 36. In the graph, availability becomes greater than 0.2 from Step 37 onward. Also, the availability and the energy consumption level converge to their normal mode limiting values: 0.214 and 16.592 (mA), as we computed in the first set of examples.

Similarly, we computed the longest mode switching time from the normal mode to a combined mode that consumes less than 12.5 (mA) of energy. That is, we find the maximum lower bound t such that $\forall t' \geq t \;.\; 8 \cdot \mathrm{P}[C(t') = Rua] + 33 \cdot \mathrm{P}[C(t') = Waa] + 8 \cdot \mathrm{P}[C(t') = Wab] + 33 \cdot \mathrm{P}[C(t') = Wab] < 12.5$ with `toB=1`. From the binary search based on $a \rightarrow \Box\, tob$ (#6 of Figure 9), we found that the time bound is 99 steps (387 ms). To check the availability change, we plot the energy consumption level changes from a counterexample when we set `tb` to 98. The second graph of Figure 10 shows the availability (solid line) and the energy consumption level (dashed line) changes from the counterexample. The graph shows that the energy consumption level remains under 12.5 (mA) from Step 99 onward. Again, the availability and the energy consumption level converge to their slow mode limiting values: 0.161 and 12.430 (mA).

As a final example, we determine the design parameter `toA` such that (1) in any state, the availability can be increased to 0.18 in 38 steps (148 ms), and (2) the energy consumption level in the steady state is less than 15 (mA). The design parameter `toA` can be searched with specifications (1) $\Box\, ac$ for the availability and (2) $\Diamond \Box \neg ec$ for the steady state energy consumption level (#7 and #8 of Figure 9).

We found that by setting `toA` to 0.683, these conditions can be satisfied. That is, if the process switches to the normal mode with probability 0.683 and to the slow mode with probability 0.317, the desired performance characteristics can be achieved. The third graph of Figure 10 shows how the availability (solid line) and the energy consumption level (dashed line) change from the counterexample when we set `tc` to 37. The graphs confirm that the availability is larger than 0.18 from Step 38 onward, and the energy consumption level converges to a value less than 15 (mA).

Regarding the performance of the iLTLChecker, all examples of this section were model checked in less than 1 sec on a Pentium III 500 MHz machine. However, it took

about 34 sec to find the counterexample of the TDoA example of Section 4. Although the number of states is larger in the TDoA example (20) than in the examples of this section (12), these numbers alone may not explain the 34-fold performance gap. To identify the cause, we examined the number of times the feasibility solver is called, but the differences were small: 190 for the TDoA example and 161 for the slowest example of this section. We found the cause from the number of pivot exchanges of the simplex method to check the feasibility of the (in)equality constraints collected along the paths of Büchi automaton. They were 17,470 for the TDoA example and 804 for the slowest example of this section. Although the fact that the time complexity of the Euclidean model checking algorithm is exponential in terms of the search depth is a significant performance factor,[37] we often find that the simplex method is another performance bottleneck (as is the case in the TDoA example). As future work, we are considering polynomial time optimization algorithms, such as the *interior point methods* [Karmarkar 1984]. A concern in adopting these methods is *Slater's condition* that requires all inequality constrains to be strict inequalities. One can replace a nonstrict inequality with a disjunction of an equality and a strict inequality, but each disjunction can result in a split node in the Büchi automaton and can exponentially increase the number of runs to check.

7. RELATED WORK

PCTL (Probabilistic Computation Tree Logic) and PCTL* are probabilistic extensions of the temporal logics CTL and CTL* such that the the universal or existential path quantifiers of CTL are replaced with with a 'degree of satisfaction' specified by a probability [Clarke and Emerson 1981; Clarke et al. 1983; Holzmann 1997; Hansson and Jonsson 1994; Aziz et al. 1995; Baier et al. 1999; Kwiatkowska et al. 2004]. Because PCTL-like logics check the specification by assigning probabilities to a set of computation paths generated from a single Markov chain model, it is difficult to evaluate aggregate properties of a large WSN expressed in such a logic. The difficulty is as follows: checking properties in a PCTL-like logic requires the crossproduct of the local states; such a crossproduct leads to the state space explosion problem. Two approaches that have been proposed to address this state space explosion are lumping symmetric states together [Bucholz 1994; Kemeny and Snell 1976], and using efficient data structures such as multivalued decision diagrams (MDDs) or matrix diagrams (MDs) [Miner and Ciardo 1999; Ciardo and Miner 1999]. However, in our case, lumping all symmetric states would end up with the original single Markov chain model without providing a way of analyzing the aggregate properties. Moreover, it is not practical to add processes simply to increase the precision in the specification. For example, it would be impractical to build a crossproduct of all drug molecules to describe the drug concentration level changes [Kwon and Kim 2010].

We take a different model checking approach in this article. Using *iLTL* to express the properties we are interested in addresses the state space explosion problem through a *statistical abstraction*. This makes iLTL formulas amenable to Euclidean model checking and obviates the need to run simulations and gather statistics, as statistical model checking approaches do, to provide probabilistic guarantees [Kwon and Agha 2011]. The main difference between the PCTL-based approaches and our iLTL-based approach is how the probability space is defined: while the PCTL-like logics compute the fraction of computation paths that satisfy each subformula of a specification, iLTL abstracts the portions of the nodes in certain states as a probability mass function (pmf) and verifies properties on the transitions of these pmfs. Because iLTL expresses properties

[37] One optimization is to check the common prefixes of a specification together with the worst case complexity on a rare occasion.

directly in terms of the pmfs, it is a suitable logic for describing the average (aggregate) behaviors of the entire system over Slogictime. As mentioned earlier, in the case of WSNs, such properties include expected energy consumption, expected throughput, and reliability. Our notion of state as a pmf vector has also been used in detecting emergent behavior in networks of cardiac mycocytes [Grosu et al. 2008].

The model checking approaches previously described seek to establish the correctness of the model checking results, a process that is computationally intensive. A different model checking approach for addressing the state space explosion problem is by providing a *statistical guarantee* based on Monte Carlo simulations and hypothesis testing [Sen et al. 2005]. VeStA is a statistical model checking tool which checks the properties of semi-Markov chains, CTMCs, and DTMCs against specifications written in Continuous Stochastic Logic (CSL) [Baier et al. 1999] or PCTL. Specifically, for each subformula of the specification, VeStA runs simulations and does statistical hypothesis testing to check if the system provides a probabilistic guarantee about the required properties. Statistical model checking has also been used to model denial of service attacks [Agha et al. 2005] and cell receptor pathways [Clarke et al. 2008]. These model checking methods solve the state space explosion problem at the cost of losing the preciseness of the model checking results. In other words, their result can only give statistical guarantees, and the result can possibly be different during its execution. Euclidean model checking directly evaluates the evolution of the pmfs, providing precise results—assuming the transition probabilities have been accurately estimated.

Besides performance evaluation approaches based on model checking, there are other formal modeling and formal analysis methods for WSNs. In Ölveczky and Thorvaldsen [2009] show how the real-time Maude language and tool [Ölveczky and Meseguer 2007, 2002] can be used to formally model, simulate, and model check WSN algorithms through an example of an optimal geographical density control (OGDC) algorithm. They demonstrate that a high-level, real-time Maude description obviates the need for implementing a complicated simulation tool and that model checking can be run against the model formally defined. Katelman et al. [2008] proposed a formally-based protocol design methodology for WSNs. In particular, the standard local minimum spanning tree (LMST) topology control protocol for WSNs has been analyzed, and an improved protocol was designed. Moreover, using real-time Maude, the preservation of the network connectivity in the standard protocol, specified as a real-time rewrite theory, is verified. One of the advantages of the formal modeling and analysis methods is that they can use a detailed description of the model (e.g., as in real-time Maude) models of the OGDC and LMST WSN algorithms [Ölveczky and Thorvaldsen 2009; Katelman et al. 2008]). On the other hand, the model description of iLTL takes a more abstract form: it is composed of differential equations on a few (typically around ten) state variables. Also, while the former examines the computation paths composed of system states, the latter explores the computation paths of expected behaviors. As a result, in Ölveczky and Thorvaldsen [2009], a WSN of six nodes was model checked, but our performance evaluation method is not limited by the number of nodes, as it works with aggregate values.

Another formal analysis method is *theorem proving*-based probabilistic analysis [Hasan and Tahar 2009]: this allows rigorous reasoning about systems (such as WSNs) using random behaviors. With the recent development in formalization of probability theory in higher-order logic, analyses of random components of a system using a higher-order logic theorem prover become possible. Evaluating the performance metrics of non-Markovian processes or of such statistical properties as the mean and the variance have been generally considered difficult with model checking techniques. Our Euclidean model checking approach cannot handle non-Markovian processes and such statistical properties as the variance: it deals with the change of the mean property over time. The ability to handle such properties is an advantage

of using theorem proving-based performance evaluation methods. However, unlike theorem proving methods which require human intervention to prove properties, our approach—as is common for model checking techniques—is fully automated.

In this article, we estimate the probability transition matrix of a Markov chain from the execution samples of a WSN, assuming that the chain's states are already known. There are more general approaches, called *stochastic grammatical learning*, that find the states as well as the probability transition matrix from system execution samples [Sen et al. 2004; Carrasco and Oncina 1994; Kermorvant and Dupont 2002; Ron et al. 1995]. Stochastic grammatical learning builds a prefix-tree automaton, that is, a Markov chain whose states are finite prefixes of states/labeled edges of the execution samples and the transition probabilities/rates are estimated as fractions of the transitions between these Markov states. A minimal state Markov chain is obtained from the prefix-tree automaton by merging the equivalent states, that is, states which have the same set of destination states and the same transition probabilities/rates to these destination states. However, statistical grammatical learning methods do not provide any semantic interpretation on the states that are inferred. Consequently, in order to apply the proposed performance evaluation methods in our work to a Markov chain model estimated this way, a semantic interpretation of the states would have to be done separately, which may be complicated because the states that are found this way may not have any clear meaning.

Finally, we summarize related work of our own. We have applied Euclidean model checking techniques to several real-world problems. Applying it to pharmaceutics, we computed drug dose by model checking a compartment model—specifically, a Markovian pharmacokinetic model—of a body against pharmaceutical requirements described using iLTL [Kwon and Kim 2010]. We evaluated the performance metrics and the reliability of software systems using iLTL model checking. A software system can be modeled as a Markov chain such that the states are the modules and the probability transition matrix is the control transition rates between the modules obtained from the user profiles. Performance criteria are expressed in iLTL and evaluated through iLTL model checking [Kwon and Agha 2007, 2011].

In a previous work related to this article [Kwon and Agha 2006], we evaluated the performance metrics of WSNs. Here, we obtain a DTMC from the execution samples of a WSN and evaluated its performance through iLTL model checking. Building on that work, we established a performance evaluation framework that could help in designing a new system as well as in evaluating the performance of an existing system. We extended the previous work with more examples, including the TDoA protocol design example, and with enriched details, such as the multiple DTMC model, the aggregation operator, and the conversion between CTMCs and DTMCs.

Another promising application area for Euclidean model checking technique is the scheduling in grid environments. Grid environments are composed of resources and resource managers which are good candidates for states when the environment is modeled as a Markov chain. In Entezari-Maleki and Movaghar [2012] compute the connection probabilities, called a scheduling policy, between the resource managers such that the mean response time of a given task can be minimized by computing the infinite sum of probability transition matrices. We believe it is possible to use Euclidean model checking to find a policy that can probabilistically mix well-defined schemes to meet complicated requirements expressed in iLTL.

Extending the ideas of iLTL model checking, we developed a temporal logic called LTLC and its model checking algorithm for linear time-invariant (LTI) systems [Kwon and Agha 2008]. LTI system models of LTLC are more general than the Markov chain models of iLTL in the sense that their state transition dynamics are affected not only by initial states but also by external inputs whereas the dynamics of Markov chain

models are dependent only on their initial states. However, having a more general model constrains the model checking algorithm to have a specialized purpose: it is suitable for synthesizing automatic controllers which ensure that a system reaches a steady state within finite steps. Specifically, the LTLC model checking algorithm finds an initial state and a finite sequence of inputs that can achieve a control objective specified in LTLC.

We have worked on extending iLTL to include nondeterminism in the model. Markov decision processes (MDPs) have a set of probability transition matrices that can be nondeterministically chosen on each step (i.e., an infinite string of the matrices is called a *scheduler*). This provides a standard model for systems with probabilistic and nondeterministic behaviors. We are interested in the model checking problem where the model is an MDP with a set of schedulers given in an ω-regular language over the probability transition matrices [Korthikanti et al. 2010].

8. CONCLUSIONS

This article develops a statistical performance evaluation method for large-scale WSNs. The statistical abstraction of a global system state as a probability mass function enables us to specify properties of the expected behaviors of the system. Many interesting properties about the expected behaviors of a system can be easily written in our probabilistic temporal logic iLTL and systematically verified through what we term Euclidean model checking. To minimize the differences between the model behavior and the system behavior, we also propose a method for estimating a DTMC model from samples of a real system. We also show how a statistical test could be used so that poor models could be rejected. The effectiveness of a our technique is supported by analyzing data from a real WSN experiment.

One of the difficulties in estimating the parameters of a system is that different DTMCs may be present in a system. For example, in a WSN, the nodes at the boundary of the system behave differently than the nodes at the center. To solve this problem, we are currently working on classifying the sample runs of a system so that runs of different characteristics can be modeled as different DTMCs. The performances of the multiple DTMCs can still be evaluated by Euclidean model checking: if they do not interact, we can check them as a multiple DTMC model; otherwise we can build a Cartesian product of the different DTMCs. In this case, the size of the model grows exponentially, but it is only in terms of the number of different types of DTMCs.

A related challenge occurs when the probability transition matrix is changing over time. For example, a WSN can switch its mode to an alert mode when there are events to monitor and it can switch back to a normal mode where the nodes are more likely to be in a sleep state. These systems can be modeled as a family of Markov chains, one for each mode and a scheduler which controls mode switches. Korthikanti et al. model such systems as Markov decision processes(MDP) and show a sufficient condition under which model checking MDPs against ω-automaton properties is decidable Korthikanti et al. [2010]. However, at this point, it is not clear how practical this approach is likely to be.

We believe that the performance evaluation framework we propose—model estimation, statistical testing, and Euclidean model checking—provides an effective tool for designing the parameters of large-scale systems like sensor networks and for systematically analyzing properties of their deployment.

ACKNOWLEDGMENTS

The authors would like to acknowledge the detailed comments from the anonymous reviewers. We also would like to thank the helpful feedback from Eunhee Kim, Timo Latvala, Kirill Mechitov, Koushik Sen, Peter Dinges, Vijay Korthikanti, and Ashish Vulimiri.

REFERENCES

AGHA, G., GUNTER, C., GREENWALD, M., KHANNA, S., MESEGUER, J., SEN, K., AND THATI, P. 2005. Formal modeling and analysis of DoS using probabilistic rewrite theories. In *Proceedings of the Workshop on Foundations of Computer Security (FCS'05)*.

ASLAM, J., BUTLER, Z., CONSTANTIN, F., CREPSI, V., CYBENKO, G., AND RUS, D. 2003. Tracking a moving object with a binary sensor network. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 150–161.

AZIZ, A., SINGHAL, V., AND BALARIN, F. 1995. It usually works: The temporal logic of stochastic systems. In *Proceedings of the 7th International Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science, vol. 939, Springer-Verlag, Berlin Heidelberg, 155–165.

BAIER, C., KATOEN, J., AND HERMANNS, H. 1999. Approximate symbolic model checking of continuous-time Markov chains. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR)*. vol. 1664, Springer-Verlag, Berlin Heidelberg, 146–162.

BUCHOLZ, P. 1994. Exact and ordinary lumpability in finite Markov chains. *J. Appl. Probab. 31*, 59–74.

CALDER, M., DUGUID, A., GILMORE, S., AND HILLSTON, J. 2006. Stronger computational modelling of signalling pathways using both continuous and discrete-state methods. In *Proceedings of the International Conference on Computational Methods in System Biology*. Lecture Notes in Computer Science, vol. 4210, Springer-Verlag, Berlin Heidelberg, 63–77.

CARRASCO, R. C. AND ONCINA, J. 1994. Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications*. Lecture Notes in Computer Science, vol. 862, Springer-Verlag, Berlin Heidelberg, 139–152.

CIARDO, G., MARIE, R. A., SERICOLA, B., AND TRIVEDI, K. S. 1990. Performability analysis using semi-Markov reward processes. *IEEE Trans. Comput. 39*, 1251–1264.

CIARDO, G. AND MINER, A. S. 1999. A data structure for the efficient Kronecker solution of GSPNs. In *Proceedings of the International Workshop on Petri Nets and Performance Models*. 22–31.

CLARKE, E., FAEDER, J., LANGMEAD, C., HARRIS, L., JHA, S., AND LEGAY, A. 2008. Statistical model checking in BioLab: Applications to the automated analysis of T-cell receptor signaling pathway. In *Proceedings of the 6th International Confernece on Computational Methods in Systems Biology*. Lecture Notes in Computer Science, vol. 5307, Springer-Verlag, Berlin Heidelberg, 231–250.

CLARKE, E., GRUMBERG, O., AND PELED, D. 2000. *Model Checking*. MIT Press, Cambridge, MA.

CLARKE, E. M. AND EMERSON, E. A. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the International Conference on 25 Years of Model Checking*. Lecture Notes in Computer Science, vol. 5000, Springer-Verlag, Berlin Heidelberg, 196–215.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1983. Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach. In *Proceedings of the 10th International ACM Symposium on Principles of Programming Languages*. 117–126.

ENTEZARI-MALEKI, R. AND MOVAGHAR, A. 2012. A probabilistic task scheduling method for grid environments. *Future Gen. Comput. Syst. 28*, 3, 513–524.

FRANKLIN, G. F., POWELL, J. D., AND EMAMI-NAEINI, A. 1994. *Feedback Control of Dynamic Systems* 3rd Ed. Addison Wesley, Boston, MA.

GROSU, R., BARTOCCI, E., CORRADINI, F., ENTCHEVA, E., SMOLKA, S. A., AND WASILEWSKA, A. 2008. Learning and detecting emergent behavior in networks of cardiac myocytes. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control (HSCC)*. Lecture Notes in Computer Science, vol. 4981, Springer-Verlag, Berlin Heidelberg, 229–243.

HANSSON, H. AND JONSSON, B. 1994. A logic for reasoning about time and reliability. *Formal Aspects Comput. 6*, 5, 512–535.

HASAN, O. AND TAHAR, S. 2009. Probabilistic analysis of wireless systems using theorem proving. *Electron. Notes Theor. Comput. Sci. 242*, 43–58.

HENZINGER, T. A., HO, P.-H., AND WONG-TOI, H. 1997. HyTech: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transfer 1*, 1–2, 110–122.

HOLZMANN, G. J. 1997. The model checker SPIN. *IEEE Trans. Softw. Eng. 23,* 279–295.

HUGHES, G. AND CRESWELL, M. 1997. *Introduction to Modal Logic*. Methuen, Methuen, MA.

KARMARKAR, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica 4*, 373–395.

KATELMAN, M., MESEGUER, J., AND HOU, J. 2008. Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In *Proceedings of the 10th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*. Lecture Notes in Computer Science, vol. 5051, Springer-Verlag, Berlin Heidelberg, 150–169.

KEMENY, J. G. AND SNELL, J. L. 1976. *Finite Markov Chains*. Springer-Verlag, Berlin Heidelberg.

KERMORVANT, C. AND DUPONT, P. 2002. Stochastic grammatical inference with multinomial tests. In *Proceedings of the 6th International Colloquium on Grammatical Inference*. Lecture Notes in Computer Science, vol. 2484, Springer-Verlag, Berlin Heidelberg, 149–160.

KHACHIYAN, L. G. 1979. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR 244*, 1093–1096.

KORTHIKANTI, V. A., VISWANATHAN, M., KWON, Y., AND AGHA, G. 2010. Reasoning about MDPs as transformers of probability distributions. In *Proceedings of the International Conference on Quantitative Evaluation of SysTems (QEST)*. IEEE Computer Society, 199–208.

KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. 2004. PRISM 2.0: A tool for probabilistic model checking. In *Proceedings of the International Conference on Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 322–323.

KWON, Y. AND AGHA, G. 2005. iLTLChecker: A probabilistic model checker for multiple DTMCs. In *Proceedings of the International Conference on the Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 245–246.

KWON, Y. AND AGHA, G. 2006. Scalable modeling and performance evaluation of wireless sensor networks. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. IEEE, 49–58.

KWON, Y. AND AGHA, G. 2007. A Markov reward model for software reliability. In *Proceedings of the Next Generation Software (NGS) Workshop at International Parallel and Distributed Processing Symposium (IPDPS)*. 1–6.

KWON, Y. AND AGHA, G. 2008. LTLC: Linear temporal logic for control. In *Proceedings of the 11the International Workshop on Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, vol. 4981, Springer-Verlag, Berlin Heidelberg, 316–329.

KWON, Y. AND AGHA, G. 2011. Verifying the evolution of probability distributions governed by a DTMC. *IEEE Trans. Softw. Engi. 37*, 1, 126–141.

KWON, Y. AND KIM, E. 2010. Specification and verification of pharmacokinetic models. In *Computational Biology and Bioinformatics, Advances in Experimental Medicine and Biology (AEMB)*. Lecture Notes in Computer Science, vol. 680, Springer, 463–472.

KWON, Y., MECHITOV, K., SUNDRESH, S., KIM, W., AND AGHA, G. 2010. Resilient localization for sensor networks in outdoor environments. *ACM Trans. Sen. Netw. 7*.

LAM, V. V., BUCHHOLZ, P., AND SANDERS, W. H. 2004. A structured path-based approach for computing transient rewards for large CTMCs. In *Proceedings of the International Conference on Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 136–145.

LANGVILLE, A. N. AND STEWART, W. J. 2004. The Kronecker product and stochastic automata networks. *J. Comput. Appl. Mathe.* 429–447.

LICHTENSTEIN, O. AND PNUELI, A. 1985. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*. 97–107.

LUENBERGER, D. G. 1989. *Linear and Nonlinear Programming* 2nd Ed., Addison Wesley, Boston, MA.

MECHITOV, K., SUNDRESH, S., KWON, Y., AND AGHA, G. 2003. Cooperative tracking with binary-detection sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM Press, 332–333.

MINER, A. S. AND CIARDO, G. 1999. Efficient reachability set generation and storage using decision diagrams. In *Proceedings of the International Conference Application and Theory of Petri Nets*. Lecture Notes in Computer Science, vol. 1639, Springer-Verlag, Berlin Heidelberg, 6–25.

ÖLVECZKY, C. AND MESEGUER, J. 2002. Specification of real-time and hybrid systems in rewriting logic. *J. Theor. Comput. Sci. 285*, 2, 359–405.

ÖLVECZKY, C. AND MESEGUER, J. 2007. Semantics and pragmatics of real-time maude. *Higher-Order Symbo. Comput. 20*, 1–2, 161–196.

ÖLVECZKY, P. C. AND THORVALDSEN, S. 2009. Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. *J. Theor. Comput. Sci. 410*. 2–3, 254–280.

PAPOULIS, A. 1991. *Probability, Random Variables, and Stochastic Processes* 3rd Ed. McGraw-Hill, New York, NY.

PARDALOS, P. M. AND VAVASIS, S. A. 1991. Quadratic programming with one negative eigenvalue is NP-hard. *J. Global Optimi. 1*, 15–22.

PLATEAU, B. AND ATIF, K. 1991. Stochastic automata network for modeling parallel systems. *IEEE Trans. Softw. Eng. 17*, 1093–1108.

Rice, J., Mechitov, K., Shim, S.-H., Nagayama, T., Jang, S., Kim, R., Spencer, B., Agha, G., and Fujino, Y. 2010. Flexible smart sensor framework for autonomous structural health monitoring. *Smart Struct. Syst. 6*, 5–6, 423–438.

Ron, D., Singer, Y., and Tishby, N. 1995. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*. ACM Press, 31–40.

Sedgewick, R. 1990. *Algorithms in C*. Addison Wesley, Boston, MA.

Sen, K., Viswanathan, M., and Agha, G. 2004. Learning continuous time Markov chains from sample executions. In *Proceedings of the International Conference on Quantitative Evaluation of Systems (QEST)*. IEEE, 146–155.

Sen, K., Viswanathan, M., and Agha, G. 2005. On statistical model checking of stochastic systems. In *Proceedings of the International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 3576. Springer-Verlag, Berlin Heidelberg, 311–324.

Strang, G. 1988. *Linear Algebra and Its Applications* 3rd Ed. Harcourt Brace Jovanovich, San Diego, CA.

Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., and Culler, D. 2004. An analysis of a large scale habitat monitoring application. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*. 214–226.

Xu, N., Rangwala, S., Chintalapudi, K. K., Ganesan, D., Broad, A., Govindan, R., and Estrin, D. 2004. A wireless sensor-network for structural monitoring. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*. 13–24.

Received August 2010; revised May, November 2011, April, August 2012; accepted September 2012