Redesign of the LMST Wireless Sensor Protocol through Formal Modeling and Statistical Model Checking

Michael Katelman, José Meseguer, and Jennifer Hou

Department of Computer Science
University of Illinois at Urbana-Champaign, U.S.A.
{katelman,meseguer,jhou}@uiuc.edu

Abstract. The local minimum spanning tree (LMST) topology control protocol tries to maintain connectivity in an ad-hoc wireless sensor network while minimizing power consumption and maximizing data bandwidth. Our formal, statistical model checking analysis of LMST under realistic deployment conditions shows that the invariant of maintaining network connectivity is easily lost. We then propose a formally-based system redesign methodology in which quantitative temporal logic formulas and further statistical model checking can be used to identify the causes of bugs, and to reach a correct system redesign. We show this methodology effective in the redesign of a version of LMST that ensures network connectivity under realistic deployment conditions.

1 Introduction

The design of wireless sensor network protocols presents many challenges. On the one hand, it is infeasible to comprehensively evaluate an ad-hoc wireless sensor network protocol based solely on deployment in the field. On the other, faithfully modeling such a protocol is far from trivial, because this requires a precise model of communication in which physical distance, location, power, and time must all be taken into account. Simulation is a widely used analysis method; but it falls short of formal analysis in its capacity to verify in a more conclusive way desired requirements. Formal modeling and analysis itself is nontrivial, because of the need for faithfully capturing the communication model, real time, and the often probabilistic algorithms (e.g. 802.11 MAC contention), or probabilistic phenomena (e.g. quartz clock drift).

The best way of using such formal modeling and analysis is not *a posteriori*, after a wireless protocol has been designed, but as a powerful method to *design* and *redesign* several times such a protocol, using the insights gained from the formal analysis to meet the desired requirements in a final design. In this work we do exactly this for the *local minimum spanning tree* (LMST) topology control protocol [18]. Only a high-level design of such a protocol under idealized circumstances existed prior to our work. At that idealized level, the key property that the protocol always maintains network connectivity had been shown by mathematical analysis in [18]. The nontrivial challenge has been to refine this high-level, idealized design into an implementable protocol version that can deal in practice with unavoidable issues such as clock drift, MAC contention, and transmission delay. The challenge has been nontrivial because our formal analysis has shown that the key invariant of maintaining network connectivity fails rather badly when these additional conditions are accounted for.

Our starting point has been the work of Ölveczky and Thorvaldsen [23, 24]. They show how to formally model and analyze the OGDC wireless sensor network protocol using Real-Time Maude [22], an extension of the rewriting logic language Maude [5] for real-time and hybrid systems. Real time is of the essence for wireless sensor network protocols such as OGDC and LMST; and we have adopted their elegant way of faithfully modeling all relevant aspects of a wireless communication model, such as its broadcast nature, plus its sensitivity to location, distance, and transmission range; and of specifying message sending and receiving events by rewrite rules, proposed in [23, 24]. We begin by specifying in this way the idealized LMST protocol as a real-time rewrite theory and analyzing it in Real-Time Maude, thus confirming by model checking the connectivity maintenance property established analytically in [18]. This serves as our base specification and provides key infrastructure on which to tackle the important challenge of arriving at a realistic (re-)design of the LMST protocol.

As soon as we introduce into the protocol model more realistic implementation details and environmental pressures, two important things happen. First, since various probabilistic phenomena naturally appear at this more realistic level, our formal specifications of the various refinements of the original model now become real-time *probabilistic* rewrite theories [13]. Probabilistic rewrite theories can be not only simulated in Maude using standard sampling techniques [3], they can also be formally analyzed by statistical model checking using the VeStA tool [26]. Second, our analysis shows that the idealized design fails quite badly to maintain network connectivity when such realistic issues are made explicit in the model; and therefore LMST requires a nontrivial *redesign*.

This work makes two main contributions. The first is to show, using a concrete state-of-the-art wireless sensor protocol like LMST, how the very successful Real-Time Maude approach to modeling and analysis of wireless sensor protocols initiated in [23, 24] can be seamlessly extended to the probabilistic setting, both at the level of specifications (passing from real-time rewrite theories to probabilistic real-time rewrite theories), and at the level of formal analysis (passing from LTL model checking in Real-Time Maude to statistical model checking in VeStA). We believe that this extension is quite useful because: (i) wireless sensor networks must operate in a probabilistic environment and often include probabilistic algorithms in some protocol components (e.g. 802.11 MAC contention); and (ii) performance issues are of the essence, and it is therefore very useful to generalize the absolute Boolean-valued guarantees of LTL requirements to probabilistic real-valued guarantees associated to probabilistic temporal logic requirements in a logic like QuaTEx [3]. In QuaTEx, the evaluation of a temporal logic formula yields a real number (not necessarily between 0 and 1) corresponding to some quantitative measurement of the system.

Our second contribution, also illustrated in the context of LMST for the sake of concreteness, but of general applicability, is to show how this style of probabilistic real-time formal specification and analysis can be the basis of a very useful *design* and *redesign methodology*. In our methodology, probabilistic real-time formal specifications and quantitative statistical model checking analysis are used throughout the design process to support three mutually-reinforcing tasks: (i) to uncover flaws in a given design; (ii) to conjecture the *causes* of the various malfunctions and to *confirm* such conjectures by means of statistical correlations between further analyses; and (iii) to then use the confirmed conjectures of the hypothesized causes of flaws to *redesign* the protocol several times and ultimately to *verify* by statistical model checking that the final design satisfies the desired requirements. In addition to LMST, the methodology is widely applicable for other wireless protocols and to other probabilistic systems, such as DoS protection protocols [2] and stochastic hybrid systems [20]. Our application of the methodology in this paper results in a new, implementable design of the LMST protocol that satisfies desired requirements in the face of realistic operating conditions.

2 The Idealized Local Minimum Spanning Tree Protocol

We begin by briefly reviewing the idealized version of the local minimum spanning tree (LMST) topology control protocol presented in [18]. The function of a topology control protocol is to define which nodes in an ad-hoc wireless sensor network communicate with each other, and with what transmission power they communicate. The goal is to minimize power consumption, prolong network lifetime, and maximize data bandwidth while maintaining network connectivity. In the case of the LMST protocol, a distributed algorithm is employed whereby each sensor node periodically updates its own *local topology*. The local topology of a node is the set of neighbors to which it routes data.

In the protocol, each wireless node is presumed to have internal quartz clock timers, a memory for buffering messages, and a wireless transmitter which is adjustable to different power levels. The periodic, real-time nature of the protocol is governed by a global constant called the *round time*, denoted rd, and is approximately $10s$. Each node constantly employs one of its timers to count the time between round boundaries, at which point the node may adjust its local topology by changing its wireless transmission strength. We refer to this timer as the *round timer*. There are therefore two notions of a round, one *global* and one *local*. A global round is any interval $[t, t + rd]$ where t is a multiple of rd. A local round is determined with respect to a particular node, and is defined as any interval between successive round timer expirations. The protocol is then defined by what happens when the local round timer of a node expires:

1. The node first broadcasts a message, called a *hello message*, at *maximum transmission strength*. The hello message contains a unique identifier of the

node and its current physical location. Hello messages are buffered by any *visible neighbor*, that is, any node within wireless transmission range.
2. The node reads from its message buffer all hello messages received during the previous round and distills from these a graph of its visible neighbors weighted by distance.
3. Taking the local graph of visible neighbors just distilled by the node, it then calculates the minimum spanning tree of that graph.
4. The nodes in the local minimum spanning tree which are directly connected (one-hop away) are selected to be the node's new *neighbors*, meaning those to which it will transmit data during this local round.
5. The node resets its round timer for rd, and waits for the timer to expire.

As shown in [18], LMST has a number of advantageous properties, including low power usage, and a provably small number of neighbors for each node, which reduces medium contention and increases bandwidth. Furthermore, it is also shown that LMST satisfies the crucial property of *maintaining network connectivity.* That is, if the graph whose edges link the sensor nodes within wireless reach of each other is connected, then the considerably smaller subgraph computed by LMST is also connected. However, LMST is an *idealized* design, which does not take into account crucial issues that must be faced in a real implementation. As we show later, the crucial requirement of maintaining network connectivity is soon lost when such issues are modeled. It thus remains an open question how LMST can be refined into a realistic design where such realistic issues are addressed and where network connectivity is still maintained.

3 Idealized LMST Model in Real-Time Maude

This section describes our formal specification and analysis in Real-Time Maude (RTM) [22] of the idealized version of LMST summarized in Section 2. We do this following the general methodology for formal modeling of wireless sensor networks proposed in [23, 24], incorporating many modeling constructions essentially unchanged. This first step of modeling and analysis serves two purposes. First, since probabilistic phenomena are not yet modeled at this idealized level, it serves as a warm-up exercise to later see how real-time specifications of wireless sensor network protocols in RTM can be naturally extended to probabilistic real-time specifications. Second, since the network connectivity invariant was only shown by high-level analytic arguments in [18] but never formally verified, it also serves as a sanity check to obtain independent, model checking evidence that the invariant holds, and to indirectly gain further confidence that our formal RTM specification faithfully captures the idealized LMST design. We first recall some background on real-time rewrite theories and their use in modeling wireless network protocols. We then summarize the specification of LMST as a real-time rewrite theory and its formal analysis in RTM.

3.1 Modeling Wireless Networks in Real-Time Maude

The key idea of rewriting logic [19] is to model a concurrent system as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) is an equational theory whose types and function symbols are described by a signature Σ, and having a set of equations E; and where R is a collection of rewrite rules. The basic idea is that the *states* of the concurrent system thus specified are elements of the algebraic data type (initial algebra) $T_{\Sigma/E}$ associated to the equational theory (Σ, E), whereas the *concurrent transitions* of the system are the possible rewrites allowed by the rewrite rules R. For our purposes in this paper it is useful to instantiate this general idea to the case of *object-based distributed systems*. For such systems it is often useful to model the distributed state as a "soup" or multiset of *objects* and *messages*, of types (sorts) *Object* and *Msg*. Such distributed, soupy states can be called *configurations* and belong to a sort *Configuration*, which contains *Object* and *Msg* as *subsorts*. The soup-like nature of the state is algebraically modeled by declaring a binary multiset union operator (which we can describe with empty, juxtaposition syntax) satisfying the axioms of *associativity* and *commutativity*, and having the empty multiset *null* as its *identity* element. Thus, if O_1 and O_2 are objects, and M_1, M_2, and M_3 are messages, then the juxtaposition $O_1\, O_2\, M_1\, M_2\, M_3$ is a configuration made up of those objects and messages, where the associativity and commutativity axioms mean that no parentheses are needed, and the order in which the objects and messages appear is immaterial. In the Maude rewriting logic language [5] and in its RTM extension, objects of a given class C in a given state are represented as terms of the form

$$\texttt{<}\ O : C\ \ \texttt{|}\, attr_1 : val_1, \ldots, attr_n : val_n \texttt{>}$$

where O is the object's name or identifier, C is the object's class, and where val_1 to val_n are the current values of the attributes $attr_1$ to $attr_n$, respectively, which in a given class C are required to have specified sorts $s_1, \ldots, s_n$. This requirement is specified in a class declaration of the form

$$\texttt{class}\ C\ \ \texttt{|}\ \ attr_1 : s_1, \ldots, attr_n : s_n.$$

The *dynamics* of an object-based distributed system are then specified by multiset rewrite rules, where one or more objects and/or messages in a configuration are rewritten to other objects and/or messages.

How about *real-time* object-based distributed systems? They can be formally specified by *real-time rewrite theories* [21], where the rewrite rules now have *time duration* information, that is, they are conditional rewrite rules of the form

$$t \xrightarrow{\tau} t' \quad if \quad cond$$

with t and t' multiset expressions involving objects and messages, and with τ a *time* expression, that is, a term of sort *Time*, where the time domain can be chosen to be either discrete or dense. If $\tau = 0$, then we call the rule an *instantaneous* rule, and we omit the 0 label. Otherwise we call the rule a *tick* rule, because time is advanced. Since time should advance not just for a local state, but for the whole system, the global configuration of a system is encapsulated by a bracket operator, so that the global state has the form $\{t\}$, with t a configuration of objects and messages. theories can be *desugared* into ordinary rewrite theories [22]. The trick is to model the global state $\{t\}$ as a pair $(\{t\}, \alpha)$, where α is the current value of a global clock. Then a tick rule $\{t\} \xrightarrow{\tau} \{t'\} \quad if \quad cond$ is desugared into an ordinary rewrite rule

$$(\{t\}, \alpha) \longrightarrow (\{t'\}, \alpha + \tau) \;\; if \;\; cond$$

Real-Time Maude is an extension of Maude that directly supports the specification of real-time systems as real-time rewrite theories. It performs the above desugaring to execute such rewrite theories in the underlying Maude system. It also supports breadth-first search and model checking of LTL properties by compilation into Maude, where now such LTL properties may involve predicates that inspect the global clock or the state of particular timers in some objects, and where the model checking can specify a time bound [22]. In [23, 24], Real-time Maude has been shown to be very well suited to model and formally analyze wireless network protocols, in particular we utilize their models of standard wireless communication types. The first is unicast messaging:

```
sort DirectedMsg .   subsort DirectedMsg < Configuration .
op directed-msg : Receiver Msg -> DirectedMsg .
```

Broadcast messages are used generally to send a message to all nodes within a certain physical radius of the transmitting node. In our specification we use them to model both wireless data transmission as well as part of our 802.11 MAC model. They use the following syntax:

```
sort BroadcastMsg .  subsort BroadcastMsg < Configuration .
op broadcast-msg : Sender Msg -> BroadcastMsg .
```

Message broadcasting is modeled as in [24], by defining a set of equations to turn each broadcast message into a set of unicast messages, one such message for each node within transmission range.

The last wrapper is used to delay the reception of a message with respect to real time. If the message is a true wireless transmission, then this delay corresponds to the transmission delay; in other cases it may model different things. We use delayed messages extensively to model timers like the one used by each node to count time between successive rounds.

```
sort DelayedMsg   .  subsort DelayedMsg   < Configuration   .
sort Delay        .  subsort TimeInf      < Delay           .
op delayed-msg : DirectedMsg  Delay -> DelayedMsg [right id: 0] .
op delayed-msg : BroadcastMsg Delay -> DelayedMsg [right id: 0] .
```

Finally, [22] shows how all of the time-elapsing events of the system should be distilled into a single tick rewrite rule of the form

```
crl [tick] :
   {Cx:Configuration}  => {delta(Cx, Tx)} in time Tx:Time
if Tx <= mte(Cx) .
```

using two operator symbols `delta` and `mte`. The first operator defines the effect of the time elapse for the system configuration, and the second operator defines the maximum time that can elapse before the next instantaneous event. Both are defined over the *Configuration* sort, and act to distribute over the objects and messages in the configuration. For example, the `delta` function is partially defined by (we indicate the sort of each variable with its first use)

```
eq delta(delayed-msg(DMx:DirectedMsg, Dx:Delay)
      Cx:Configuration
      , Tx:Time)
=  delayed-msg(DMx, Dx monus Tx)
   delta(Cx, Tx) .
```

3.2 Idealized Model of LMST in Real-Time Maude

Our definition of the LMST protocol hand specifies the description given in [18] through a set of (mostly) localized rewrite rules governing state changes of the nodes individually. In order to make our version conform to the idealized definition of the protocol, we have to define guards for each rewrite rule that consider the entire state before firing. For example, we guard the rule for updating a node's local topology to ensure that all outstanding hello messages are received prior to doing the update calculation. This corresponds to a tacit assumption made by the protocol that before a topology update occurs, a hello message has been received from every visible neighbor.

The protocol is idealized in the sense that it is free from real-world conditions such as message loss, node mobility, imperfect quartz clock timers, and so on. In addition, we assume that all the round timers are synchronized, so that all of them always signal a new round in unison. The following class definition defines wireless sensor nodes:

```
class SensorNode |
   location          : Location
   , received-HMs    : HelloMsgValueSet
   , neighbor-set    : NodeIDSet
   , transmit-radius : Float .
```

The attributes give the node's physical location, the buffered hello messages received since the start of the current round, the current set of neighbors, and the current transmission radius. Since the goal of a topology control protocol is to determine each sensor node's wireless transmission strength, it may seem curious that this information is missing from the above definition. The reason is that our analysis will only be concerned with connectivity, for which having the transmission radius is simply more convenient. We note that transmission strength can be calculated from the transmission radius and knowledge about the radio propagation model.

There are three message constructors: one to represent wireless communication of hello messages, and two others representing other state-changing actions that a node must take itself.

```
msg round-timer-msg  :                  -> Msg .
msg update-msg       :                  -> Msg .
msg hm-msg           : HelloMsgValue    -> Msg .

op <_,_> : NodeID Location -> HelloMsgValue .
```

The payload of a hello message is the sending node's identifier plus its location. This message is broadcast by each node at the beginning of each new round (see Section 2). There are exactly three rewrite rules in the model, one for each of the messages above. The idea is that every concurrent event in the system has an associated message, and is acted upon when received by the node it is directed to. This idea is similar to what is suggested in [3].

Instead of presenting the rules verbatim from our idealized model, we simply indicate what each rule does using prose. This saves space and omits syntactic overhead which does not directly contribute to the interesting actions being performed. Our model, which can be downloaded [1], contains the exact definitions.

The events associated with the above messages are as follows:

1. When a `round-timer-msg` is consumed, the associated node performs three actions. It first broadcasts a hello message identifying itself and giving its current location. Second, it schedules an `update-msg` for itself. Third, it resets its round timer, emitting a delayed `round-timer-msg`.
2. When a `hm-msg` is consumed, the receiving node simply adds the message payload to the `received-HMs` set.
3. When a `update-msg` is consumed, the receiving node updates its local topology (setting `neighbor-set`). The updated topology is calculated from the minimum spanning tree of the graph defined by the hello messages received during the previous round. One slight complication is that for the topology to conform to the idealized specification, we must ensure that all hello messages be received during the current time instant before a topology update is executed. This is checked for using an equationally defined guard and a conditional rewrite rule.

The three rules described above essentially define the entire model. Most of the specification focuses on defining the minimum spanning tree calculation.

The formal model can be analyzed through *time-bounded LTL model checking* in RTM. When we analyze the model, we start with an initial configuration that is primed by inserting one directed round timer message for each node in the configuration, set to be consumed instantly at time 0. The property that we are interested in is *total network connectivity*, which means that multi-hop connections exist from every node to every other node in the network. As an LTL safety formula, the property is expressed as the invariant $\Box$ `connected`, with `connected` an equationally-defined predicate (see [1]). It computes the strongly connected components of the graph defined by the union of all neighbor sets.

The model checking that we have done is to check that the connectedness property holds in our model *with respect to a set of randomly selected initial configurations*, each involving a small number (4) of nodes and with a time bound of one round. It is important to make clear exactly what this result means with respect to the idealized protocol. The non-determinism exhibited in our model comes from selecting the order in which nodes broadcast hello messages, receive messages, *etc.* However, due to the guard on the topology update rule, the non-determinism that was added does not result in many interesting interactions.

4 Probabilistic Modeling and Analysis of LMST

A major issue with the analysis of the previous section is that standard model checking algorithms do not apply to *probabilistic phenomena*, such as the uniform distribution of nodes in a sensing area. Indeed, many probabilistic phenomena naturally exist for wireless protocols. In this section we describe two realistic refinements of the idealized LMST model: one with unsynchronized local round timers, and the other with quartz clock drift, 802.11 MAC contention, and message delay. Due to space limitations, we have omitted two other refinements – one with node mobility and another with probabilistic message loss – that we have also modeled and analyzed (see [11]). The refinements are modified versions of the idealized model, transformed into a standard rewrite theory, with new rules and probabilistic annotations. The annotations take the formalization outside the realm of standard rewrite theories and into the realm of *probabilistic rewrite theories* [13, 3]. After the probabilistic models are defined, we show how to automatically analyze them using the *logic of quantitative temporal expressions* (QuaTEx) [3] and the statistical model checking algorithms implemented by the VeStA tool [3, 26]. The analysis reveals significant disconnectedness, or *bugs*, under the conditions imposed by each of the refinements.

4.1 Probabilistic Rewrite Theories, QuaTEx, and VeStA

A *probabilistic rewrite theory* [13, 3] replaces the usual rewrite rules with probabilistic ones of the form

$$l(\boldsymbol{x}) \longrightarrow r(\boldsymbol{x}, \boldsymbol{y}) \texttt{ with probability } \boldsymbol{y} := p(\boldsymbol{x})$$

Such a rule is *non-deterministic*, because the term r has new variables $\boldsymbol{y}$ disjoint from the variables $\boldsymbol{x}$ appearing in l. Therefore, a substitution θ for the variables $\boldsymbol{x}$ appearing in l that matches a subterm of a term t at position q does not uniquely determine the next state after the rewrite: there can be many different choices for the next state, depending on how we instantiate the extra variables $\boldsymbol{y}$ in r. In fact, we can denote the different next states by expressions of the form $t[r(\theta(\boldsymbol{x}), \sigma(\boldsymbol{y}))]_q$, where θ is fixed as the given matching substitution, but σ ranges over all possible substitutions for the new variables $\boldsymbol{y}$. The probabilistic nature of the rule is expressed by the notation: `with probability` $\boldsymbol{y} := p(\boldsymbol{x})$, where $p(\boldsymbol{x})$ is a probability measure on the set of substitutions σ (modulo the equations E in a given rewrite theory). However, the probability measure $p(\boldsymbol{x})$ *may depend on the matching substitution* θ. We sample $\boldsymbol{y}$, that is, the substitution σ, probabilistically according the probability measure $p(\theta(\boldsymbol{x}))$.

PMaude [3] is an extension of Maude for probabilistic rewrite theories. Like Real-Time Maude, PMaude is implemented as a theory transformation using Maude's reflection capabilities. This involves desugaring the probability annotation in a probabilistic rewrite rule and giving rewriting definitions for the various probability distributions. The desugaring makes the probability measure $\boldsymbol{y} := p(\boldsymbol{x})$ a rewriting condition, $p(\boldsymbol{x}) \longrightarrow \boldsymbol{y}$, and goes into a plain conditional rewrite rule. The probability distributions are implemented using special features in Maude for generating random numbers according to a *uniform* distribution (see [5, §9.3]). The uniform distribution is then used to sample into other distributions, for example

```
rl BERNOULLI(R) => if rand / rand-max < R then true else false fi .
```

is used to sample a Bernoulli distribution with success probability R, given as a rational number. The operator `rand` is treated specially by Maude; it returns a natural number between 0 and the constant `rand-max`.

As we noted above, both Real-Time Maude and PMaude are implemented as theory transformations. So via Real-Time Maude we can go from a real-time rewrite theory to a plain rewrite theory, and via PMaude we can go from a probabilistic rewrite theory to a plain rewrite theory. To combine the two paradigms what we have done is to manually apply the desugaring described above for PMaude within our larger real-time rewrite theory.

For analyzing probabilistic rewrite theories, it is often desirable to state logical queries *quantitatively*, that is, not with a *true* or *false* answer, but with a real number corresponding, for example, to a probability or, more generally, to some quantitative measurement of our system. For this reason, we use the QuaTEx probabilistic temporal logic of *Quantitative Temporal Expressions* proposed in [3]. This language is supported by the VeStA tool [26], which has an interface to Maude. The key idea of QuaTEx is to generalize probabilistic temporal logic formulas from Boolean-valued expressions to real-valued expressions. The Boolean interpretation is preserved as a special case using the real numbers 0 and 1. As usual, QuaTEx has *state expressions*, evaluated on states, and (real-valued) *path-expressions* evaluated on computation paths. The notion of state predicates is now generalized to that of *state functions*, which can evaluate quantitative properties of a state. QuaTEx is particularly expressive because of the possibility of defining recursive expressions. In this way, only the next operator (`#` in VeStA syntax) and conditional branching (`if` $Bexp$ `then` $Pexp$ `else` $Pexp'$ `fi`, with $Bexp$ a Boolean expression and $Pexp, Pexp'$ path expressions) are needed to define more complex operators, such as "until". Model checking queries are given as expected values of path expressions. We refer to [3] for a detailed account of QuaTEx and its semantics. The VeStA tool [26] then performs *statistical model checking* on a probabilistic system by evaluating a QuaTEx expression on computation paths obtained by Monte Carlo simulation. The model checking query is parameterized by two values, α and δ, which are user-provided. VeStA responds to a query with a $(1 - \alpha) \cdot 100\%$ confidence interval bounded by δ for the expected value of the random variable defined by the QuaTEx formula. Depending on the slack allowed by the given parameters, VeStA may need greater or fewer sample runs to compute this interval.

Using VeStA requires that the PMaude model be free from unrestrained non-determinism that does not come from sampling a probability distribution. In particular, for the tool to work correctly it should never be the case that two rules apply to the same term. One way of solving this is described in [3]. There, a uniform distribution is used to give each event a (probabilistically) unique identifier and an event queue then orders the events by identifier. Each rewrite rule can only fire if the event associated with it is the one indicated by the front of the event queue. In our case this matching is done by adding a third field to the directed message construct and adding a special object with the event queue

```
op directed-msg : Receiver Msg EventID -> DirectedMsg .
class Control |
   event-queue : EventQueue .
```

Finally, we change each of the rules to match the directed message being consumed by the rule (all of our rules are of this form) with the the event at the front of the queue. We found that the details of handling events and the event queue are tricky to do cleanly. We did not want the intent of the rules to be obscured by a mass of syntax that simply deals with the event queue mechanism. The details of our eventual solution can be found in [11, 1].

4.2 Refinement 1: Unsynchronized Timers

Our first refinement changes the round timers so that they are no longer globally synchronized. Instead, the timers are initially set to expire somewhere in the time interval $[0s, 10s]$, rather than at time 0 as in the idealized case. This is accomplished by adding a new message type , `init-msg`, and an associated *probabilistic rewrite rule*. In the starting configuration the only messages are the initialization messages, one for each sensor node, and we execute the rule below to get to the desired starting state where the protocol can be applied. The rule draws a value uniformly from $[0, 10]$ and uses this value to initialize the node's round timer. Note that we use a simplified syntax below, but the intent should be clear. In addition, we use $(\ldots)$ to indicate omitted code.

```
rl [init] : init-msg ... => delayed-msg(round-timer-msg, y) ...
      with probability y = uniform-dist(0(s), 10(s)) .
```

This rule is also used to initialize the node's location (not shown), therefore overcoming one of the problems with the analysis done in Section 3, namely, the inability to have the model checking analysis directly consider all (probabilistically) possible starting positions for the nodes.

Although this is a very simple change, it exposes a bug in the protocol. To see this, we first recast the connectedness property as the QuaTEx formula below. The key idea is to calculate the percentage of an interval, [`lower`, `upper`], when the network is totally connected. Whenever the tick rule is applied `PrctCon` looks at the interval stepped over (using `#`, QuaTEx's *next* operator), determines if the network was connected, determines if the time interval overlaps [`lower`, `upper`], and concisely records this information.

```
PrctCon(x:Bool, y:Float, z:Float) =
   if time > lower and y < lower then
      #PrctCon(connected, time, z + Start(x, y));
   else if y >= lower and time <= upper then
      #PrctCon(connected, time, z + Mid(x, y))
   else if y > upper then
      100.0 * (z + End(x, y) / (upper - lower));
   else
      #PrctCon(connected, time, z);
   fi; fi; fi;
```

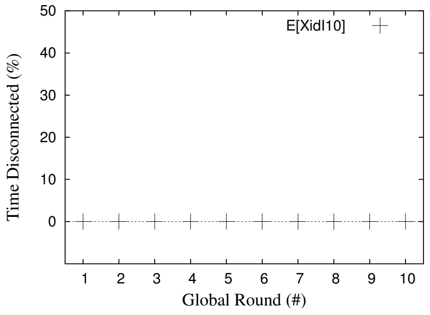
The `x` argument of `PrctCon` records whether or not during the previous time step the network was connected, `y` records the starting time of the previous time step, and `z` enumerates the time that the network is connected in the interval. The state expressions, such as `connected` and `time`, have the obvious meanings. The value returned by model checking the above QuaTEx formula in VeStA is an interval describing, statistically, the expected value of the *percentage of time the network was totally connected during the time interval* [*`lower`*, *`upper`*].

In Figure 1b we have plotted the expected value of `PrctCon` for the first ten global rounds of network operation. The result shows endemic disconnectedness during the first two global rounds. For comparison, we plot the same value for a probabilistic version of the idealized model in Figure 1a. In Section 5 we use a novel redesign methodology to fix the bug causing the disconnectedness.

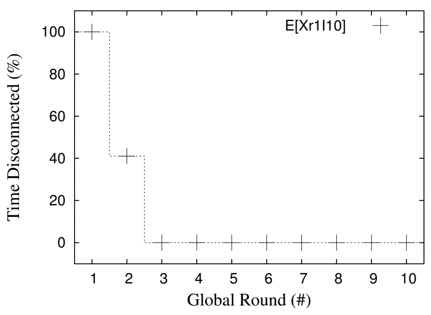
4.3 Refinement 2: Delay Uncertainty

The LMST protocol is highly localized in the sense that very little collaboration between nodes takes place. Therefore it is especially important to investigate the inter-node ordering of events and messages. Our second refinement addresses: (1) wireless transmission delays, (2) imperfections in quartz clock timers, and (3) the 802.11 MAC contention procedure [4]. Item (1), wireless transmission delay, is easily implemented using the `delayed-msg` construct defined in Section 3.1.

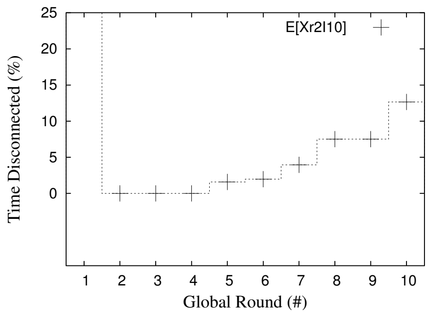
For realistic quartz clocks, additional infrastructure is needed. First, the wireless sensor node state gets a new attribute, `clock-drift`, and the initialization rule is modified so that each sensor node gets a value, uniformly distributed in



(a) Connectedness in idealized model.



(b) Connectedness in refinement 1.



(c) Connectedness in refinement 2.

Fig. 1: Quantified connectedness of the idealized model and refinements 1 and 2.

the $[-5.0, 5.0]$ interval, for its clock drift. A slow clock is represented with negative values, and a fast clock with positive values. The absolute value gives the drift in *parts per million.*

In addition, we introduce a *Timer* sort, which we treat as a new type of *Delay*, defined by constructors

```
sort Timer  .  subsort Timer < Delay   .
op ACTIVE   : Float  -> Timer .
op SUS      : Float  -> Timer .
```

The floating point arguments indicate the amount of time remaining according to the node's internal clock, which may be fast or slow. The two constructs differentiate between an active timer and one that has been suspended. Suspended timers are used as part of the 802.11 MAC contention protocol. Quartz clock drift is then modeled by replacing all event delays associated with timers with these constructs; plus a few auxiliary operations to update the constructs to account for drift. The main change that we need to make is in the definition of `delta` for the tick rule, which now must scale time decrements by clock drift. To do this we add a new equation to the definition of `delta`, exactly like the one in Section 3.1 but with

```
... delayed-msg(DMx, ACTIVE(Fx1 monus Tx with-scale Fx2))) ...
eq Fx monus Tx with-scale Fy =
      if Fy < 0.0 then
         Fx + float(Tx) - (float(Tx) * ((- Fy) / 1000000.0))
      else
         Fx + float(Tx) + (float(Tx) * ((Fy) / 1000000.0)) fi .
```

substituted for delayed messages having a regular delay. Note that we omit matching the drift (`Fx2`) in appropriate sensor node to save space. For suspended timers, the `delta` function leaves the timer unchanged. There are some complications to deal concurrently with the event queue (see [11, 1]).

The 802.11 MAC contention procedure defines how sensor nodes try to arbitrate access for the transmission medium. Too much noise in the medium can cause messages to be dropped, so the basic idea of the protocol is that each node measures the noise level and waits when it is too noisy. To model the 802.11 MAC contention procedure we define four new events:

```
msg difs-timer-msg      : -> Msg .
msg backoff-timer-msg   : -> Msg .
msg medium-busy-msg     : -> Msg .
msg medium-clear-msg    : -> Msg .
```

and a modification of the round timer rule. Under the 802.11 regime [4] when a sensor node wants to transmit a message (e.g. a hello message) it sets two timers. The DIFS timer is set for a fixed time period and is made active rightaway. The backoff timer is set *probabilistically* and waits for the DIFS period to finish before it becomes active. Therefore we change the round timer message event to emit two delayed messages according to the protocol (again, simplified syntax below)

```
rl [round-timer-msg] : round-timer-msg ... =>
      delayed-msg(difs-timer-msg    , ACTIVE(128(mu-s)))
      delayed-msg(backoff-timer-msg , SUS(y * 50(mu-s)) ...
         with probability  y := uniform-dist(0, 15) .
```

The interaction between the two timers is governed by the *carrier sense mechanism*, which is implemented using the other two new message types defined above. In addition to these messages we add a new field to our sensor node class called `medium-busy`, which takes a natural number value and records when the medium is busy. The value is 0 when the medium is not busy, gets incremented on every `medium-busy-msg`, and decremented on every `medium-clear-msg`.

Due to space limitations we cannot give all of the details of our 802.11 model (see [11] for full details). The part of the protocol that we have not specified here defines how the two timers respond to changes in the carrier sense mechanism. The basic idea is that whenever the medium becomes busy (`medium-busy` field goes from 0 to 1), the backoff timer is suspended and the DIFS timer gets reset. When the backoff timer finishes, then the node can finally transmit its message. Since we are only concerned with hello messages, whenever a backoff timer message is consumed, the node transmits its hello message.

Analysis results are plotted in Figure 1c and show real disconnectedness during portions of the first ten rounds. The significance of such a level of disconnectedness depends on many variables. What we found though is that an expected disconnectedness value of $x\%$ usually corresponds to an $x\%$ chance that the network is disconnected for the entire round. The bugs causing the disconnectedness are identified and fixed in Section 5.

5 A New Realistic Design of LMST

The methods of statistical quantitative analysis have so far been used to diagnose serious issues with the LMST protocol, but, unfortunately, the problems uncovered remain otherwise idiopathic. Determining the *causes* of the undesired behavior is a crucial part of the formal analysis process. In this section we propose a new method for redesigning probabilistic systems by statistical quantitative analysis. Instead of calculating expected values $E[X]$ directly, we calculate *mathematical correlations*, $\rho(X, Y)$, of two random variables. The idea is that one of the random variables represents the symptoms of a bug, and the other a hypothesized cause. This method can save significant time during the debugging and redesign process by providing a way to isolate the cause of the a bug *without first implementing a hypothesized fix*, which might require substantial effort, and only afterward trying again to detect the bug in the "fixed" model.

5.1 A Formally-Based System Redesign Methodology

Debugging and redesigning a complex protocol is rarely a trivial task, particularly when the protocol is concurrent and/or probabilistic, because specific symptoms are usually difficult to reproduce. For concurrent finite state systems one can witness bugs directly using standard model checking algorithms to generate counter-examples. These can then be used to diagnose the cause of a bug and guide the effort required to actually fix it. However, for probabilistic systems and approximate quantitative analysis no analogue seems to exist.

Moreover, the process of determining the cause of a bug is extremely important, because otherwise, without a known cause, any effort used to modify the model can easily be wasted if the changes do not happen to hit on the cause. This wasted effort can be substantial if, for example, a fix does not fit cleanly within the current architecture of the model specification. Therefore it is essential to know beforehand that such efforts will likely result in success.

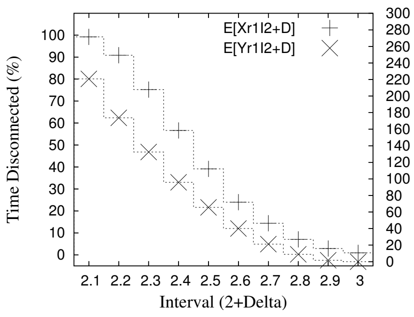
Our proposed methodology to address these issues begins with the idea of using statistical quantitative analysis to calculate *mathematical correlations*. The correlation of two random variables, X representing the *symptoms*, and Y the *hypothesized cause*, is calculated to establish the likelihood of a causal relationship between X and Y. Of course, X and Y being correlated does not *prove* a causal relationship, but it is a necessary condition for causality and indicative of it. Since specifying X and Y, for example as QuaTEx formulas, usually requires much less effort than directly modifying the model, the process can lead to reduced debugging and redesign time. However, unlike the finite state concurrent system case, some extra work must be done to formulate Y. The proposed system redesign methodology assumes two initial items:

- A probabilistic model $\mathcal{M}$, for example a PMaude module.
- An observable value measuring, for any given sample from the probability space defined by $\mathcal{M}$, fit or disagreement with required behavior. This takes the form a random variable X over the sample probability space and can be defined by a QuaTEx formula.

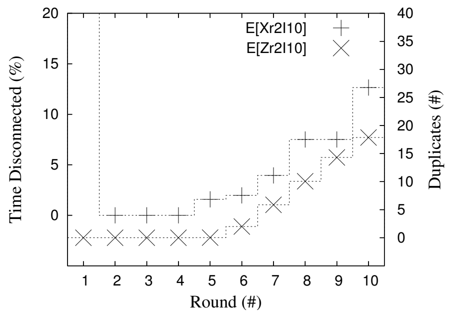
The methodology then proceeds through the following steps:

1. Use statistical quantitative analysis, using tools such as by VeStA or PRISM [14], to calculate $E[X]$. If the value indicates buggy behavior, then go on with the remaining steps, otherwise there is no need to go further.
2. Hypothesize a cause for the bug. The hypothesized cause can be formulated also as a QuaTEx formula, yielding a second random variable, Y, on the sample probability space, $\mathcal{M}$.
3. The next step is to calculate the *correlation coefficient* (see [25]), $\rho(X, Y)$, which is easily accomplished by observing the following expansion, which turns $\rho(X, Y)$ into a simple expression of expected values
$$\rho(X,Y) = \frac{\mathrm{Cov}\, XY}{\sigma(X)\sigma(Y)} = \frac{E[XY] - E[X]E[Y]}{\sqrt{E[(X - E[X])^2]}\sqrt{E[(Y - E[Y])^2]}}$$
Therefore, statistical quantitative analysis is used to first calculate $E[X]$, $E[Y]$, and $E[XY]$; and then secondarily using these values we calculate $E[(X - E[X])^2]$ and $E[(Y - E[Y])^2]$. The final value of $\rho(X, Y)$ is easily calculated from these values. If the correlation is not significant (i.e. close to 0) then this step is repeated with a new hypothesis, otherwise we go on.
4. Modify the model $\mathcal{M}$ to remove the cause articulated through Y, yielding a new model $\mathcal{M}'$. For $\mathcal{M}'$ calculate, via statistical quantitative analysis, the expected value of X', which captures the same observable value as X but is a random variable over the probability space defined by $\mathcal{M}'$ instead of $\mathcal{M}$. If $E[X']$ shows no buggy behavior then we finish, otherwise repeat.

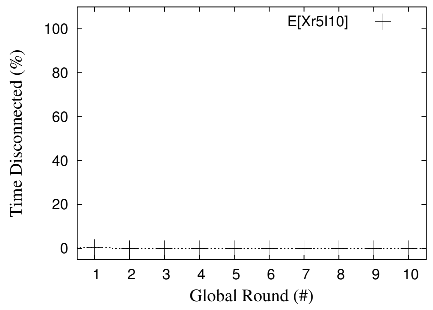
We feel that this method adds rigor to the process of redesigning a probabilistic system, and does so without a lot of auxiliary, extraneous work that would otherwise be needed. Compared to a more informal method, the extra burden on the designer is only in expressing Y concretely. While certainly not always trivial, writing down the hypothesized cause has a number of benefits outside of just fixing the immediate bug. For example, it provides concrete documentation of the bug, assurances that the bug has truly been fixed, and an easy regression test to make sure the bug is not re-introduced in the future. In Section 5.2 we apply this methodology to determine the causes of the bugs uncovered in Section 4. Using this knowledge we then redesign the LMST protocol to reach a realistic, implementable design which ensures connectedness.



(a) Debugging of refinement 1.



(b) Debugging of refinement 2.



(c) Connectedness in our LMST redesign.

Fig. 2: Causal analyses used in our LMST redesign.

5.2 Redesign of the LMST Protocol

Let us now consider applying the redesign methodology to our model of the LMST protocol. The analysis results for refinements 1 and 2 (Figure 1) show significant disconnectedness behavior. With the first refinement the disconnectedness appears to be transient, but the second refinement results in pervasive disconnectedness across global rounds. We fix both bugs in this section.

According to the methodology outlined in the previous section, we need to define a random variable X describing the correctness property that we are interested in. This poses a subtle problem, given the data presented in Figure 1, because we evaluated *multiple separate* QuaTEx formulas, one for each global round, to create the plot. Our solution to making this set of random variables into a single one is to, for each simulation run, encode a probabilistically chosen global round to evaluate the disconnectedness property on.

We first look at refinement 1, and the set of intervals partitioning global round 2 into 10 parts. We still need to hypothesize and quantify a *cause* for the undesired behavior witnessed through X. The hypothesis that we made was that the bug involved something we will call *late neighbors*. Late neighbors are visible neighbors that do not make themselves known to other nodes within their transmission range during global round 1 because their local round *begins later*. Since they send their hello messages late, they are not received in time to be incorporated into the (early) neighbor's first local topology update. Due to space limitations, we again refer to [11, 1] for full details.

Figure 2a plots connectedness versus the total number of late neighbors for nodes that have not yet begun their second local round. It is clear that the two plots exhibit strong correlation, which can be calculated for the random variables just described using the methods of the previous section. The resulting correlation is 0.99, thus indicating that we should make sure that the problem of late neighbors is fixed.

The situation with refinement 2 is even more interesting, because, as seen in Figure 1c and re-printed in Figure 2b, the disconnectedness behavior of this model is persistent across rounds. We conjectured that what might be happening was something like the following: one node with a fast clock processes a round timer event so early, relative to a neighbor, that it sends its round $i + 1$ hello message before the second node processes its round i message. Therefore, the second node gets a duplicate message from the first node.

To precisely quantify this conjecture, we defined a QuaTEx formula to calculate the number of duplicate messages where the node sending the duplicate message has a clock that is fast relative to its neighbor. Figure 2b plots results for both random variables, and it is easy to see in this graph that the two are correlated. The exact value is -0.13, indicating significant correlation, but not as high as we would like for such a small number of intervals. However, with the outlier for global round 1 removed the correlation rises to 0.98. Therefore, it seems reasonable to try to fix the bug by removing duplicate messages.

We want to fix both bugs uncovered above: the one due to late neighbors and the other due to duplicated messages from clock drift. The late neighbors problem is easily solved by *each node broadcasting a hello message as soon as it turns on.* To prevent clock drift and MAC contention from causing duplicate messages, we take a two-pronged approach. Since drift and contention only cause small perturbations in the amount of time between successive hello messages, our solution is to *delay topology updates for a few milliseconds after a round timer event.* However, this does not completely solve the problem and in fact only delays it as the clocks drift farther apart. So the second component of our fix is to *apply an ultra-lightweight clock synchronization algorithm [16].* The overhead of the algorithm is that now each hello message gets time-stamped.

Our new design, with the three fixes explained above, completely removes the problems that we observed in refinements 1 and 2. Correctness results, based on quantitative statistical analysis of our new design, are given in Figure 2c.

6 Related Work and Conclusions

Alternatives to Real-Time Maude [22] include Uppaal [17] and HyTech [7], which use timed and linear hybrid automata as the underlying modeling formalisms. Compared with Real-Time Maude, Uppaal and HyTech trade expressibility in the modeling language for certain decidability results.

There are various alternatives to PMaude and VeStA. The PRISM tool [14] supports a BDD-based probabilistic model checking algorithm and also an approximate, statistical model checking analysis through Monte Carlo techniques. PRISM supports a large number of modeling formalisms [14]. In [12] the algorithm used by VeStA is refined so that an the sample size necessary to achieve normality in the data can be computed before analysis begins. The modeling language used is probabilistic rewrite theories. Other probabilistic model checking tools include APMC [8], $E \vdash MC^2$ [9], and Rapture [10]. As with Real-Time Maude, the most significant difference between PMaude/VeStA and other tools is the tradeoff between expressiveness and algorithmic power.

The combination of PMaude and VeStA has been used in multiple case studies, including the analysis of a DoS resistant TCP/IP protocol [2] and two case studies involving distributed object-based stochastic hybrid systems [20]. In addition there are other case studies on 802.11 [15] and sensor networks [6].

In conclusion, this work has presented two main contributions. First, we have extended the formal specification and analysis approach for wireless sensor networks advocated by Ölveczky and Thorvaldsen [23, 24] to the probabilistic and statistical model checking setting, demonstrating significant flaws in a realistic protocol. Second, we have presented a system redesign methodology applicable to probabilistic systems in general and wireless sensor networks in particular where QuaTEx-based quantitative measurement of bugs and their hypothesized causes can be correlated; and the knowledge thus gained can be used to redesign a system free of the given bugs.

Acknowledgment

This research was supported by The Boeing Company, Grant C8088-557395.

References

1. `http://peepal.cs.uiuc.edu/~katelman/fmoods_2008.tgz`.
2. G. Agha, C. Gunter, M. Greenwald, S. Khanna, J. Meseguer, K. Sen, and P. Thati. Formal Modeling and Analysis of DoS Using Probabilistic Rewrite Theories. In *Foundations of Computer Security (FCS)*, 2005.
3. G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. In *Proc. of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)*, 2005.
4. ANSI/IEEE. *ANSI/IEEE Std 802.11, 1999 Edition (R2003).*
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework.* Springer, 2007.
6. A. Demaille, T. Hérault, and S. Peyronnet. Probabilistic Verification of Sensor Networks. *Intl. Conf. on Research, Innovation and Vision for the Future*, 2006.
7. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. In *Proc. of the 9th Intl. Conf. on Computer Aided Verification (CAV 1997)*, pages 460–463. Springer-Verlag, 1997.
8. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *Proc. 5th Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*. Springer, 2004.

9. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov Chain Model Checker. In *Proc. of the 6th Intl. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2000)*, 2000.
10. B. Jeannet, P. R. D'Argenio, and K. G. Larsen. RAPTURE: A tool for verifying Markov Decision Processes. In *Tools Day'02, affiliated with 13th Int. Conf. on Concurrency Theory (CONCUR 2002)*, Technical Report. Faculty of Informatics, Masaryk University Brno, 2002.
11. M. Katelman, J. Meseguer, and J. Hou. Formal Modeling, Analysis, and Debugging of a Localized Topology Control Protocol with Real-Time Maude and Probabilistic Model Checking (In Preparation). Technical report, University of Illinois at Urbana-Champaign, 2008.
12. M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. A Probabilistic Formal Analysis Approach to Cross Layer Optimization in Distributed Embedded Systems. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS 2007)*, volume 4468 of *LNCS*. Springer, 2007.
13. N. Kumar, K. Sen, J. Meseguer, and G. Agha. A Rewriting Based Model for Probabilistic Distributed Object Systems. In *The 6th IFIP Intl. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS 2003)*, 2003.
14. M. Kwiatkowska, G. Norman, and D. Parker. Quantitative analysis with the probabilistic model checker PRISM. *ENTCS*, 153(2):5–31, 2005.
15. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In *Proc. of the Second Joint Intl. Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV 2002)*, 2002.
16. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7), 1978.
17. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Intl. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
18. N. Li, J. C. Hou, and L. Sha. Design and Analysis of an MST-Based Topology Control Algorithm. In *INFOCOM 2003*, 2003.
19. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
20. J. Meseguer and R. Sharykin. Specification and Analysis of Distributed Object-Based Stochastic Hybrid Systems. In *Hybrid Systems: Computation and Control (HSCC 2006)*, volume 3927 of *LNCS*, pages 460–475. Springer, 2006.
21. P. C. Ölveczky and J. Meseguer. Specification of Real-Time and Hybrid Systems in Rewriting Logic. *Theoretical Computer Science*, 285:359–405, 2002.
22. P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher Order and Symbolic Computation*, 20(1-2), 2007.
23. P. C. Ölveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude. In *20th Intl. Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.
24. P. C. Ölveczky and S. Thorvaldsen. Formal Modeling and Analysis of the OGDC Wireless Sensor Network Algorithm in Real-Time Maude. In *Proc. of the 9th IFIP Intl. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2007)*, 2007.
25. E. Parzen. *Modern Probability Theory and Its Applications*. John Wiley & Sons, Inc., 1960.
26. K. Sen, M. Viswanathan, and G. Agha. VESTA: A Statistical Model Checker and Analyzer for Probabilistic Systems. In *2nd Intl. Conf. on the Quantitative Evaluation of Systems*, 2005.