Formal Modeling and Analysis of Leader Election in MANETs*

Si Liu[1], Peter Csaba Ölveczky[2], and José Meseguer[1]

[1] University of Illinois at Urbana-Champaign
[2] University of Oslo

Abstract. The modeling and analysis of mobile ad hoc networks (MANETs) pose non-trivial challenges to formal methods. Time, geometry, communication delays and failures, mobility, and uni- and bidirectionality can interact in unforeseen ways that are hard to model and analyze by automatic formal methods. In this work we use rewriting logic and Real-Time Maude to address this challenge. We propose a composable formal framework for MANET protocols and their mobility models that can take into account such complex interactions. We illustrate our framework by analyzing a well-studied leader election protocol for MANETs in the presence of both mobility and uni- and bidirectional links.

1 Introduction

The human factor is *everything*, particularly in scientific research. Thanks to the friendship, the creativity, the intellectual generosity, and the inspiration of Martin Wirsing and his pioneering work (with Kosiuczenko) on *timed rewriting logic* [29,14], two of us (Ölveczky and Meseguer) started working together on a line of research that has kept us busy for almost twenty years and is at the core of the present work. In 1995 Ölveczky visited Martin at LMU and was fired up by Martin's new ideas on formally modeling real-time systems with rewrite rules. By various circumstances he made his way to Menlo Park, fell in love with the place, and was allowed to work for his Bergen Ph.D. under Meseguer at SRI. At the time it was not clear how timed rewriting logic and standard rewriting logic, though by design very close to each other, were precisely related. This was clarified in [30] and in Ölveczky's dissertation [28], which proposed the alternative model of *real-time rewrite theories* as a *special class* of ordinary rewrite theories to specify real-time and hybrid systems. The associated Real-Time Maude tool [27] also started in [28], and has since then been applied to a wide range of real-time systems (see [26] for an overview).

It therefore seems fitting to honor Martin Wirsing on this festive occasion with a work in an area that he initiated and to which he has continued making important contributions up to this date. One of the key strengths of the

* Partially supported by NSF Grant CNS 13-19109 and AFOSR Grant FA8750-11-2-0084.

rewriting logic approach to the modeling and formal analysis of real-time systems is the flexibility and naturalness with which it can specify and analyze many distributed object-based real-time systems whose discrete states may have unbounded data structures and are therefore beyond the pale of timed automata models [2]. Furthermore, this is accomplished without losing completeness of the analysis and useful decidability properties [31].

The present work illustrates the use of the expressive power of real-time rewrite theories to model not only time and distributed objects, but also *geometry*, *mobility*, and *wireless communication*, which are needed for wireless applications such as wireless sensor networks and mobile ad hoc networks (MANETs). Real-Time Maude was first used on sensor network protocols in [33], and for MANETs in the work started in [16] and continued here. The work in [16], which is further developed in this paper, provides a general framework for modeling and analyzing MANETs in which a protocol can be seamlessly *composed* with various mobility models by exploiting the class inheritance features of Real-Time Maude in the style of [32]. In particular, our framework allows us to formally model different kinds of MANET protocols, and then formally analyze them together with reasonably precise models of both

– any of several commonly used models for node mobility, and
– spatially bounded wireless communication, which takes into account both link directionality (uni- or bidirectional) and the interplay between communication delay and mobility.

Simulation tools typically represent node locations explicitly and analyze MANETs using common node mobility models [34,6], whereas formal approaches to MANETs usually abstract from node locations and consider arbitrary topology changes (if any), as explained in Section 7. Our framework allows us to see Real-Time Maude as *both* a simulation tool and a formal analysis tool for MANETs.

In [16] we showed the power and flexibility of our framework by analyzing the Ad hoc On-Demand Distance Vector (AODV) routing protocol under various mobility models. In this paper we apply our framework on the well known leader election (LE) protocol for MANETS by Vasudevan, Kurose, and Towsley [41]. Modeling and analyzing the LE protocol pose a number of challenges not encountered in the analysis of AODV, including:

– LE assumes that the underlying framework detects new and lost links that appear/disappear due to nodes moving into, or out of, the transmission ranges of other nodes; however, no neighborhood discovery process is given.
– LE features both *one-hop* and *multi-hop* communication; however, LE does not present any transport protocol, but just assumes that the underlying communication framework provides certain guarantees, such as "a message must eventually be received if the receiver is connected to the sender forever."

LE is defined and verified only for bidirectional links [40], but its developers conjecture that LE also works correctly in the presence of unidirectional links.

Apart from providing a formal model of LE, our novel contributions in the LE case study include:

– Defining fairly abstract executable models of both multi-hop communication and of neighbor discovery in MANETs.
– Defining LE also for unidirectional links.
– Model checking LE in a number of different settings, including with unidirectional links, without finding flaws significant in LE, thereby strengthening the confidence that LE also works correctly with unidirectional links.

This paper is organized as follows: Section 2 gives a background to Real-Time Maude. Section 3 recapitulates our framework for modeling MANETs in Real-Time Maude. Section 4 gives an overview of the LE protocol. Section 5 presents our Real-Time Maude model of LE, and Section 6 describes its formal analysis. Finally, Section 7 discusses related work and gives some concluding remarks.

2 Real-Time Maude

Real-Time Maude [27] extends Maude [5] to support the formal specification and analysis of real-time systems. The specification formalism emphasizes *ease* and *generality* of specification, and is particularly suitable for modeling distributed real-time systems in an object-oriented style. Real-Time Maude specifications are executable, and the tool provides a variety of formal analysis methods, including simulation, reachability analysis, and LTL and timed CTL model checking.

Specification. A Real-Time Maude module specifies a *real-time rewrite theory* [27] $(\Sigma, E \cup A, \mathit{IR}, \mathit{TR})$, where:

– Σ is an algebraic *signature*; that is, declarations of *sorts*, *subsorts*, and *function symbols*, including a data type for time, which can be discrete or dense.
– $(\Sigma, E \cup A)$ is a *membership equational logic theory* [3], with E a set of (possibly conditional) equations, and A a set of equational axioms such as associativity, commutativity, and identity. $(\Sigma, E \cup A)$ specifies the system's state space as an algebraic data type.
– IR is a set of *labeled conditional rewrite rules* specifying the system's local transitions, each of which has the form[1] $[l] : t \longrightarrow t'$ **if** $\bigwedge_{j=1}^{m} \mathit{cond}_j$, where each cond_j is either an equality $u_j = v_j$ (u_j and v_j have the same normal form) or a rewrite $t_j \longrightarrow t'_j$ (t_j rewrites to t'_j in zero or more steps), and l is a *label*. Such a rule specifies an *instantaneous transition* from an instance of t to the corresponding instance of t', *provided* the condition holds.
– TR is a set of *tick rules* l : `{`t`}` $\longrightarrow$ `{`t'`}` **in time** τ **if** cond that advance time in the *entire* state t by τ time units.

We briefly summarize the syntax of Real-Time Maude and refer to [5] for more details. Operators are declared `op` f `:` $s_1 \ldots s_n$ `->` s, and can have user-definable syntax, with underbars '`_`' marking the argument positions. Some operators can have equational *attributes*, such as `assoc`, `comm`, and `id`, stating, respectively,

[1] An equational condition $u_i = v_i$ can also be a *matching equation*, written u_i `:=` v_i, which instantiates the variables in u_i to the values that make $u_i = v_i$ hold, if any.

that the operator is associative and commutative and has a certain identity element, so that rewriting is performed *modulo* the declared axioms. Equations and rewrite rules are introduced with, respectively, keywords `eq`, or `ceq` for conditional equations, and `rl` and `crl`. The mathematical variables in such statements are declared with the keywords `var` and `vars`. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` (for "otherwise") attribute can be applied to a term $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied.

A class declaration `class` C `|` att_1 `:` s_1`,` ... `,` att_n `:` s_n declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C in a given state is represented as a term `<` $O : C \mid att_1 : val_1, ..., att_n : val_n$ `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. A *message* is a term of sort `Msg`.

The state of an object-oriented specification is a term of sort `Configuration`, and is a *multiset* of objects and messages. Multiset union is denoted by an associative and commutative juxtaposition operator, so that rewriting is *multiset rewriting*. For example, the rewrite rule

```
rl [l] :  m(O,w)
          < O : C | a1 : x, a2 : O', a3 : z, a4 : y >
         =>
          < O : C | a1 : x + w, a2 : O', a3 : z, a4 : y >
          dly(m'(O',x), y) .
```

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`, the attribute `a1` of object `O` is changed to `x + w`, and a new message `m'(O',x)` is generated; this message has a *message delay* `y`, and will become the "ripe" message `m'(O',x)` in `y` time units. Attributes whose values do not change and do not affect the next state of other attributes or messages, such as `a3`, need not be mentioned in a rule. Attributes that are unchanged, such as `a2`, can be omitted from right-hand sides of rules.

A *subclass* inherits all the attributes and rules of its superclasses.

Formal Analysis. Real-Time Maude's *timed rewrite* command simulates *one* of the many possible system behaviors from the initial state by rewriting the initial state up to a certain duration. The timed *search* command uses a breadth-first strategy to search for states matching a search pattern that are reachable from the initial state t within a certain time interval.

Real-Time Maude's *linear temporal logic model checker* analyzes whether *each* behavior satisfies a temporal logic formula. *State propositions* are operators of sort `Prop`, and their semantics is defined by equations of the form

`ceq` *statePattern* `|=` *prop* `=` *b* `if` *cond*

for b a term of sort `Bool`, which defines *prop* to hold in all states t where t `|=` *prop* evaluates to `true`. A temporal logic *formula* is constructed by state propositions and temporal logic operators such as `True`, `False`, `~` (negation), `/\`, `\/`, `->` (implication), `[]` ("always"), `<>` ("eventually"), and `U` ("until"). Real-Time Maude provides both *unbounded* and *time-bounded* LTL model checking.

If the reachable state space is finite, the unbounded model checking command `(mc` *t* `|=u` *formula* `.)` checks whether the temporal logic formula *formula* holds in all behaviors starting from the initial state *t*. If the reachable state space is infinite, the *time-bounded* model checking command

```
(mc t |=t formula in time <= timeLimit .)
```

in which each behavior is only analyzed up to time *timeLimit*, can be used to ensure termination of the analysis.

3 Modeling MANETs in Real-Time Maude

Analyzing MANET protocols under reasonably realistic conditions is challenging. Models of node movement are needed, and must be combined with wireless communication, where typically only nodes within a certain distance of the sender receive a message with sufficient signal strength. Since *both* the sender *and* a potential receiver may move during the "messaging delay," the potential receiver could move *into*, or *out of*, the transmission range of the sender *during* the messaging delay.

Combining node mobility with reasonable precise models of wireless communication is therefore nontrivial and is currently barely addressed by formal methods. In [16] we propose a framework for specifying and analyzing MANET protocols under different mobility models in Real-Time Maude by: (i) formally specifying several popular *mobility models*; (ii) studying the different constituents of wireless "messaging delay"; (iii) defining a model of wireless communication in the presence of node movement; and (iv) explaining how our model of mobility and communication is easily composable with a Real-Time Maude specification of a MANET protocol. This section briefly recapitulates our framework.

Mobility Models. A number of different *entity mobility patterns*, where a node's movement is independent of the movements of the other nodes, have been proposed to model node mobility in realistic scenarios. The following main entity mobility models [4] are illustrated in Fig. 1:

– *Random Walk:* The node moves in "rounds" of fixed durations. At the beginning of each round, the *new speed* and the *new direction* of a node are randomly chosen, and the node moves accordingly.
– *Random Waypoint:* In each round, the node first pauses for some time, and then randomly chooses a *new destination* and a *new speed*, and travels to that destination at the chosen speed.
– *Random Direction:* The node chooses a *random direction and speed*, and travels until it reaches the boundary of the area. The node then pauses for some time, before randomly selecting a new direction and speed, and so on.

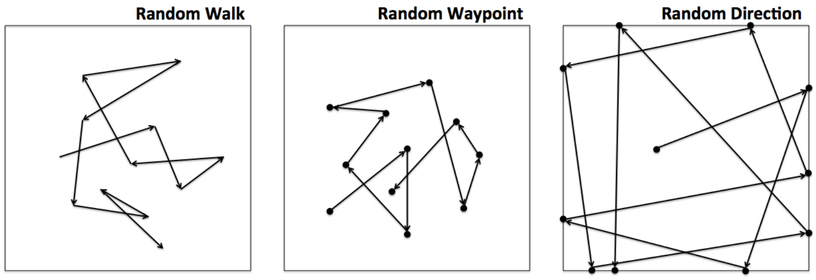

Fig. 1. Motion paths of a mobile node in three mobility models, where a bullet • depicts a pause in the movement.

Communication Delay. The per-hop communication delay can be seen as consisting of three parts [39]. The *sending delay* is the time from the moment a sender wants to send a message until the moment it is sent. This includes the time that the sender needs to buffer the message in its radio output buffer, the *media access delay* waiting for a clear channel to transmit, and the time needed to transmit the message. The *radio propagation delay* is the time it takes for a message to travel through the air from sender to receiver. The *receiving delay* denotes the time spent on the receiver side to pass the received message from device to application. Since the transmission range in MANETs usually ranges from 10 to 100 meters, while the radio propagation speed is approximately 3×10^8 meters per second, we abstract from the radio propagation delay.

Formal Model of Mobility and Wireless Communication. We summarize the Real-Time Maude model of mobility and wireless communication first presented in [16] and improved in our current work (see [18] for more details).

Mobility Models. We assume that nodes move in a two-dimensional square with length `areaSize`. A *location* is then represented as a pair x . y of rational numbers. We model a MANET node as an object of (a subclass of) the class `Node`, whose attributes denote the node's current location and its transmission range:

```
class Node  |  currentLocation : Location,  transRange : Nat .
```

Since different nodes can have different transmission ranges, we may have *unidirectional* connections.

A *stationary* node is an object instance of the subclass `StationaryNode`:

```
class StationaryNode .          subclass StationaryNode < Node .
```

A mobile node is modeled as an object of a subclass of the class `MobileNode`:

```
class MobileNode | speedVector : SpeedVector,   timer : TimeInf .
subclass MobileNode < Node .
```

`speedVector` is a term `<` *xSpeed* `,` *ySpeed* `>` denoting the node's speed *and* direction, with *xSpeed* (resp., *ySpeed*) denoting the distance traveled along the x-axis (resp., y-axis) during one time unit. The `timer` attribute is used to ensure that a node changes its movement (or lack thereof) in a timely manner.

A node moving according to the *random walk* (resp., *random waypoint*) model is modeled by an object of the subclass `RWNode` or `RWPNode`, respectively:

```
class RWNode  | speedVectorRange : SpeedVectorRange,
                boundaryTimer : TimeInf .
class RWPNode | speedRange : SpeedRange,  destRange : DestRange,
                status : Status .
subclass RWNode  RWPNode < MobileNode .
```

`speedVectorRange`, `speedRange`, and `destRange` denote the set of possible speed vectors, speeds, and destinations, respectively. The `status` attribute is either `pausing` or `moving`, and `boundaryTimer` denotes the time until the node hits the area boundary. The rewrite rules modeling node movement are given in [18].

Modeling Wireless Communication in Mobile Systems. As mentioned above, if we abstract from the radio propagation delay, the per-hop delay can be seen to consist of two parts: the delay at the sender side and the delay at the receiver side. The point is that exactly those nodes that are within the transmission range of the sender *when the sending delay ends* should receive a message. Our communication model assumes that the one-hop sending delay is a *constant* `sendDly`; it therefore abstracts from issues such as buffering of multiple messages at the sender or dynamic delays caused by network congestion, etc.

One-hop broadcast, *one-hop unicast*, and *one-hop multicast* are modeled using the following "message wrappers:"

```
msg broadcast_from_ : MsgContent Oid -> Msg .
msg unicast_from_to_ : MsgContent Oid Oid -> Msg .
msg multicast_from_to_ : MsgContent Oid OidSet -> Msg .
```

where `Oid` is the identifier of a node; `OidSet` denotes sets of node identifiers; and `MsgContent` is the sort for message contents. For example, when a node *sender* wants to broadcast some message content *mc* in one hop, it sends a "message" `broadcast` *mc* `from` *sender*.

Each node (in the set of intended receivers) that is within the transmission range of the sender exactly when the sending delay expires should receive the message, as a *single* message of the form *mc* `from` *sender* `to` *receiver*.[2]

Compositionality. A MANET protocol P can be specified, without having to take mobility and communication into account, by letting a node in the protocol description be specified as a subclass of `Node`:

[2] Since there is also a delay on the receiver side, this message can only be read/consumed when also the receiving delay has elapsed.

```
class PNode | protocol-specific attributes .
subclass PNode < Node .
```

A specification involving nodes of class *P*`Node` can then be analyzed under different mobility models by defining the nodes in the initial state to be object instances of a subclass of *both* *P*`Node` and a mobility class, such as `RWPNode`:

```
class RWPPNode .
subclass RWPPNode < RWPNode PNode .
```

4 The LE Leader Election Algorithm for MANETs

Leader election is a fundamental problem in distributed systems, and has a variety of applications in wireless networks, such as key distribution, routing coordination, and general control. One of the most well known leader election algorithms targeting MANETs is the algorithm, which we call LE, of Vasudevan, Kurose, and Towsley in [41]. The LE algorithm aims at electing the *best-valued* node (according to some measure, such as the amount of remaining battery life) in each connected component as the leader of that connected component.

In a *static* topology, LE works as follows. When an election is triggered at a node, the node broadcasts an *election* message to its immediate neighbors. A node that receives an *election* message for the *first* time, records the sender of the message as its *parent* in the spanning tree under construction, and multicasts an *election* message to its other immediate neighbors. When a node receives an *election* message from a node that is not its parent, it immediately responds with an *ack* message. When a node has received *ack* messages from all of its children, it sends an *ack* message to its parent. Each such *ack* message to a parent includes the identity and value of the best-valued node in the subtree (of the spanning tree defined by the "parent" relation) rooted at the sender. Therefore, when the source node has received an *ack* message from all of its children, it can determine the best-valued node in the entire spanning tree. The source node then broadcasts a *leader* message announcing the identity of this new leader. Figure 2 shows a run of LE under a static topology of five nodes, with node `1` being the source and node `5` the best-valued node (the higher the node number, the better value it has).

Multiple nodes can concurrently initiate *multiple* elections; in this case, only one election should "survive." This is done by associating to each election a *priority*, so that a node already in an election ignores incoming elections with lower priority, but participates in an election with *higher* priority.

To deal with *dynamic* settings, with node mobility and the resulting new and lost links, the algorithm is extended as follows:

1. When a parent-child pair becomes disconnected during the election process, the parent removes the child from its waiting list of acknowledgments and continues its election. The child terminates the current election by announcing as the leader its maximal downstream node.

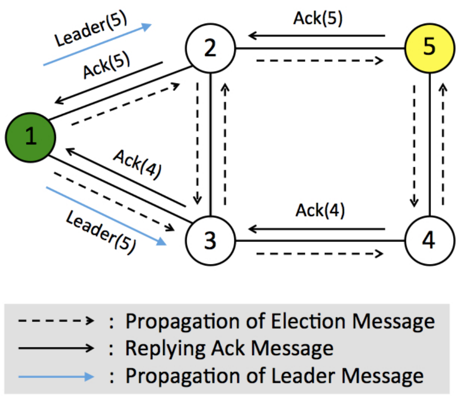
Leader(5)
Ack(5)
Ack(5)
Ack(4)
Ack(4)
Leader(5)
1
2
3
4
5
- - - - → : Propagation of Election Message
——→ : Replying Ack Message
——→ : Propagation of Leader Message

Fig. 2. An LE run in a static topology.

2. When a new link forms between two nodes that have already finished their elections, the new neighbors exchange leader information. The node with the lower-valued leader adopts the higher-valued leader as its new leader, and propagates this new leader to the other nodes in its connected component. If one or both of the nodes in the new link are still in the process of electing a leader, they continue their separate election processes. When they have both terminated their local leader elections, they exchange leader information with each other and proceed as in the above case.

The report [40] gives a detailed pseudo-code specification of LE.

In [41,40] the authors prove the following main correctness theorem: The system will reach the following situation: Each node i has a leader, which, furthermore, is the best-valued node reachable from i. The authors of [41] state that they assume that the links are bidirectional, but add that “the algorithm should work correctly even in the case of unidirectional links, provided that there is symmetric connectivity between nodes.”

Communication. In LE, *election* messages are sent to “immediate neighbors,” which should amount to one-hop broadcast/multicast. On the other hand, *ack* messages use source-to-destination (i.e., “multi-hop”) unicast, and therefore rely on the network infrastructure being able to transport messages to a given node. This can be achieved by composing LE with some routing protocol, such as AODV, and some transport protocol, such as UDP or TCP, that uses the obtained routing information to transport messages from source to destination. However, the description of LE does not specify any routing or message transport. Instead, [40] assumes that (i) links between two neighbors are bidirectional and FIFO, and (ii) that “a message sent by a node is eventually received by the intended receiver, provided that the two nodes remain connected forever starting from the instant the message is sent.”

Neighbor Discovery. LE assumes that each node knows its neighbors, and that new links formed by node mobility are detected somehow. However, LE does not specify any neighbor discovery algorithm, nor does it make explicit the assumptions/requirements on the discovery of new links. The exception is that an explicit "probe" protocol is used to discover the loss of connection to a node from which a node awaits an *ack* message.

5 Modeling LE in Real-Time Maude

This section presents our Real-Time Maude model of the LE protocol as specified in detail in [40]. We show 8 of the 23 rewrite rules in our specification. The entire executable Real-Time Maude specification is available at `https://sites.google.com/site/siliunobi/leader-election`.

5.1 Nodes and Messages

We model an LE node as an object of a subclass `LENode` of the class `Node`. The new attributes are the identifier of the leader (`leader`), the parent (`parent`), the current best-valued node (`max`), the node's computation number (`number`), its computation index (`src`), the set of neighbors from which the node has yet to receive an *ack* message (`acks`), a flag indicating whether the node is currently in an election (`eflag`), a flag indicating whether the node has sent an acknowledgement to its parent (`pflag`), the node's (immediate) neighbors (`neighbors`), the new neighbors found by the neighbor discovery process (`newNbs`), and the relevant nodes which can no longer reach the node (`lostConxs`):

```
class LENode | leader : Oid,       parent : Oid,     max : Oid,
               number : Nat,       src : CompIndex,  acks : OidSet,
               eflag : Bool,       pflag : Bool,     neighbors : OidSet,
               newNbs : OidSet,    lostConxs : OidSet .
subclass LENode < Node .
```

A computation index is a pair o ~ k, with o a node identifier and k a computation number. As in [41], we assume that a node's identifier determines its value.

The three message types in LE are represented as message contents of the forms `election(...)`, `ack(...)`, and `leader(...)`.

5.2 Modeling Communication

In LE, nodes broadcast/multicast *election* messages to *immediate neighbors*, and unicast *ack* messages to their parent (and other) nodes. Sending to immediate neighbors may be seen as *one-hop* broadcast/multicast, which we model as explained in Section 3: the sender sends a "broadcast message;" after time `sendDly` this broadcast message is distributed to all nodes within transmission range of the sender *at that moment*, and will arrive `rcvDly` time units later.

Unicasting *ack* messages, however, may involve multiple hops, since a node may have moved away from its parent by the time the *ack* message should be sent. As mentioned in Section 4, LE does not specify a transport protocol to transmit such messages, but only requires (i) that communication between neighbors is FIFO and (ii) that the destination node must get the message if it is connected to the sender forever from the time when the message is sent.

In this paper, we abstract from details about how messages are routed, and model multi-hop message transmission as follows:

– the sender sends a `multiHopUnicast` message to the destination node;
– if there *exists* a communication path from source to destination exactly `multiHopSendDelay` time units later, the message will be received by the destination node `multiHopSendDly` + `rcvDly` time units after it was sent.

We model such communication as follows:[3]

```
op multiHopUnicast_from_to_ : MsgContent Oid Oid -> Msg .
op mhTransfer : MsgContent Oid Oid -> Configuration .

eq multiHopUnicast MC from O1 to O2 = dly(mhTransfer(MC,O1,O2), multiHopSendDly).
eq {mhTransfer(MC, O1, O2) CONF}
 = if O2 in reachable(O1, CONF) then {dly(MC from O1 to O2, rcvDly) CONF}
   else {CONF} fi .

op reachable : OidSet Configuration -> OidSet .

ceq reachable(O1 ; OS,         --- add O2 to nodes reachable from (O1 ; OS)
              < O1 : Node | currentLocation : L1, transRange : R >
              < O2 : Node | currentLocation : L2 >  CONF)
  = reachable(O1 ; O2 ; OS,   < O1 : Node | >  < O2 : Node | >  CONF)
    if not (O2 in (O1 ; OS)) /\ L2 withinTransRange R of L1 .

eq reachable(OS, CONF) = OS [owise] .    --- fixed point reached
```

Since this model abstracts from the actual route by which a message is transported, a message that happens to be transferred in one hop has the same delay as one that uses 10 hops. Our model satisfies the requirement that messages are delivered if there is a path from source to destination forever. However, our model does not guarantee FIFO transmission between neighbors for two reasons:

1. Two one-hop messages sent "at the same time" results in two messages with the same delay, since our model abstracts from details about the buffering of outgoing messages.
2. Since we abstract from routing details, a "multi-hop" message has sending delay `multiHopSendDly` even if it happens to need only one hop, and could be overtaken by a one-hop broadcast message sent later along the same link.

[3] We do not show the declarations of mathematical variables; they follow the Maude convention that such variables are written with capital letters.

5.3 Neighbor and Connectivity Discovery

LE assumes that new (one-hop) links caused by node movement are detected. We model such neighbor discovery abstractly by periodically updating the `newNbs` attribute of each node with those nodes that are within transmission range but are not included in the node's `neighbors` attribute, and by removing from `neighbors` those nodes that are no longer within transmission range.

In LE, the leader of a component "periodically sends out a heartbeat message to other nodes," which can then discover whether they are disconnected from the leader. Each node n also periodically sends a *probe* message to each node n' from which it awaits an *ack* message. If n does not receive a *reply* message from n' within certain time, it assumes that the connection to n' is lost. Finally, LE assumes that a node knows when it becomes disconnected from its parent. We abstract from heartbeat and probe/reply protocols, and instead periodically check whether a connection is lost to nodes in `acks`, the leader, or the parent.

We include in the state a timer object `< 100 : GlobalND | timer :` t `>` that triggers both the neighbor discovery process and the lost connectivity process, periodically, each time the timer expires:

```
rl [computeNewNbsAndLostConnections] :
   {< O : GlobalND | timer : 0 >   CONF}  =>
   {< O : GlobalND | timer : period >  updateNbsAndAck(CONF)} .
```

where, for each node object o in `CONF`, `updateNbsAndAck`:

1. sets o's `newNbs` attribute to o's current immediate neighbors minus the nodes already in o's `neighbors` attribute;
2. removes all nodes which are no longer o's neighbors from o's `neighbors` attribute;
3. sets o's `lostConxs` attributes to those relevant nodes that cannot reach o (in multiple hops).

We refer to the online specification for the definition of this function.

5.4 Modeling the Behavior of LE

LE consists of five parts: initiating leader election, handling an *election* message, handling an *ack* message, handling a *leader* message, and dealing with new neighbors and lost connections.

Starting Leader Election. A "message" `electLeader(`o`)` kicks off a run of LE with node o as initiator. Node o multicasts a message `election(`o ~ n`)` to all its immediate neighbors, where n is the latest computation number.[4] The source will then wait for the *ack* messages from those neighbors by setting `acks` to `OS`. Moreover, it sets `eflag` to `true`, indicating that it is currently in an election:

[4] In case there are multiple concurrent runs of the protocol, this index helps deciding which run should continue.

```
rl [init-leader-election] :
   electLeader(O)
   < O : LENode | eflag : false,  neighbors : OS,  number : N,  leader : O2 >
 =>
   < O : LENode | acks : OS,     src : O ~ N,    number : N + 1,
                  eflag : true,  pflag : false,  parent : O,   max : O >
   (multicast election(O ~ N, O2) from O to OS) .
```

Receiving an Election Message. When a node that is not involved in an election (`eflag` is `false`) receives an `election` message from `SND`, the node sets `SND` as its `parent`, and sets its `src`, `eflag`, and `pflag` attributes accordingly. The node multicasts an `election` message to all its neighbors except the parent:[5]

```
crl [join-1] :
    (election(I, LID) from SND to O)
    < O : LENode | eflag : false, leader : LID, neighbors : OS1 >
  =>
    < O : LENode | src : I, acks : OS2, eflag : true, pflag : false,
                   parent : SND, max : O >
    (multicast election(I, LID) from O to OS2)
  if OS2 := delete(SND, OS1) .
```

There are five more rules for handling `election` messages; see [17] for details.

Receiving Ack Messages. When a node receives an `ack` message, for the current computation `I`, from a node `SND`, it deletes `SND` from the set `acks`. If the reported best node `M'` is better than the node's current best-valued node `M`, then the `max` attribute is also updated accordingly:

```
rl [update-acks] :
   (ack(I, FL, M') from SND to O)
   < O : LENode | pflag : false, src : I, acks : OS, max : M >
 =>
   < O : LENode | acks : delete(SND, OS),
                  max : (if FL and M' > M then M' else M fi) > .
```

All `acks` Received. When a node is no longer waiting for any `ack` message (`acks` is `empty`), and it has not yet sent an `ack` to its parent (`pflag` is `false`), it sends an `ack` message to its parent, with its best-valued node `M`. However, if the node initiated this round of the protocol (and therefore is the root node) it starts propagating the leader `M` to its immediate neighbors:

```
rl [all-acks-received-1] :
   < O : LENode | acks : empty, src : (O' ~ N), pflag : false,
                  parent : P,   max : M,        neighbors : OS >
 =>
   if O =/= O'   --- not root node
```

[5] Multicast to the empty set generates no messages in our communication model [18].

```
   then < O : LENode | pflag : true >
        (multiHopUnicast ack(O' ~ N, true, M) from O to P)
   else < O : LENode | eflag : false, leader : M >
        (multicast leader(O' ~ N, M) from O to OS)  fi .
```

Leader Message Handling. If a node already in an election receives a `leader` message for the first time, it just updates the local `leader`, clears the `eflag` (its election is over), and propagates the received message:

```
crl [adopt-new-leader-1] :
    (leader(I, LID) from SND to O)
    < O : LENode | pflag : true, eflag : true, max : M, neighbors : OS >
  =>
    < O : LENode | leader : LID, eflag : false, src : I >
    (multicast leader(I,LID) from O to OS)   if M <= LID .
```

New Links. If one or more new neighbors have been found from a node `O` that has already finished its election, then the node multicasts the leader message to the new immediate neighbors:

```
rl [new-links-found] :
   < O : LENode | newNbs : O' ; OS, eflag : B, src : I, leader : LID >
 =>
   < O : LENode | newNbs : empty >
   (if not B and LID =/= O then
      (multicast leader(I,LID) from O to (O' ; OS))  else none fi) .
```

Lost Connections. If a node gets disconnected from its parent while still in an election, it terminates the diffusing computation by announcing its maximal downstream node as the leader:

```
rl [disconnected-from-parent] :
   < O : LENode | lostConxs : OS ; P, pflag : true, eflag : true,
                  parent : P, max : M, src : I, neighbors : OS2 >
  =>
   < O : LENode | lostConxs : OS,  eflag : false, leader : M >
   (multicast leader(I, M) from O to OS2) .
```

Timed Behavior. Due to lack of space, we refer to [18,17] for the definition of the timed behaviors. The main point is that time cannot advance when an instantaneous rule is enabled, so that all actions are performed in a timely manner.

6 Formal Analysis of the LE Protocol

This section shows how our modeling framework for MANETs can be composed with our model of the LE protocol to analyze LE under realistic mobility and communication models. As already mentioned, the LE developers prove the correctness of LE only for bidirectional links, although they "strongly believe that

[LE] would still work correctly if links were unidirectional, as long as all nodes have a path to each other." We analyze LE with both bidirectional links and unidirectional links; the latter are a consequence, e.g., of nodes sending with different signaling power. We consider both static and dynamic topologies, and also analyze a system with two connected components that repeatedly merge and partition because of node movement.

Although many papers (e.g., [41,7,36,10,35,15,37,38,23,8]) have studied LE, little is known by way of formal analysis about how it behaves with unidirectional connections or under realistic mobility scenarios. We are also not aware of any study taking into account the joint effects of communication delay and mobility.

6.1 Nodes

As mentioned in Section 3, we can combine our protocol specification with a node mobility model by having nodes as object instances of a subclass of both `LENode` and a mobility class, such as `RWPNode` for random waypoint mobility and `StationaryNode` for stationary nodes:

```
class RWPLENode .         subclass RWPLENode < RWPNode LENode .
class SLENode .           subclass SLENode < StationaryNode LENode .
```

6.2 Modeling Checking the Correctness Property

We use model checking to analyze the main correctness property of LE, as described in [41]:

> *"[E]very connected component will eventually select a unique leader, which is the most-valued-node from among the nodes in that component."*

The following atomic proposition `unique-leaders` holds if and only if all nodes in a connected component have the same leader, which is, furthermore, the highest-valued node in that connected component:[6]

```
op unique-leaders : -> Prop [ctor] .

eq {< O : LENode | leader : O >  REST} |= unique-leaders = false .
--- no leader ('O') selected by some node

ceq {< O : LENode | leader : O' >  REST}
    |= unique-leaders = false if O' < O .  --- O better than its leader

ceq {< O1 : LENode | leader : O' >  < O2 : LENode | >  REST}
    |= unique-leaders = false
   if O' < O2    --- wrong leader selected by O1
      /\ O2 in reachable(O1, < O1 : LENode | >  < O2 : LENode | >  REST) .

eq {SYSTEM} |= unique-leaders = true [owise] .
```

[6] Remember that the value of a node is given by its identifier.

The main correctness property can then be formalized as the LTL formula `<> unique-leaders`. Given an initial state `initConf`, the following commands return `true` if the property holds (possibly up to the duration of the test round, `roundTime`); otherwise, a trace illustrating the counterexample is shown.

```
(mc {initConf} |=u <> unique-leaders .)
(mc {initConf} |=t <> unique-leaders in time <= roundTime .)
```

We can use unbounded model checking for static topologies. In dynamic topologies, the locations of the moving nodes contribute to an infinite reachable state space, and model checking must be time-bounded in order to terminate.

6.3 Scenarios and Analysis

Our model enables us to experiment with LE under many different scenarios by changing the values of system parameters such as node locations and movement patterns, transmission ranges, one-hop and multi-hop send delays, node velocities, the frequency of the neighbor/connectivity detection process, the number of concurrent runs of the protocol, and so on.

We use the following setting for our experiments, with additional scenario-specific settings presented separately:

- The transmission range of a node is $10\,m$, and the test area is $100\,m \times 100\,m$.
- The one-hop delays at the sender side and the receiver side are 1 time unit and 0, respectively. The multi-hop "send" delay is 2 time units.
- `roundTime` (i.e., the time bound in the model checking) is 20.

Scenario I. Scenario I corresponds to the topology in Fig. 2, and consists of five stationary nodes with bidirectional connections. The nodes `1`, `2`, `3`, `4`, and `5` are located at `(45 . 45)`, `(50 . 50)`, `(50 . 40)`, `(60 . 40)`, and `(60 . 50)`, respectively. We consider two sub-scenarios: (a) only node `1` initiates a round of the leader election protocol; and (b) all five nodes initiate a run of the protocol. Time-bounded model checking shows that the property holds: all five nodes elect the best-valued node `5` as their leader within 20 time units; the execution times of the analyses are 150 milliseconds (ms) and 4500 ms, respectively.

Scenario II. This scenario, shown in Fig. 3 (where a solid line denotes a bidirectional link, an arrow denotes a unidirectional link, and a dashed circle shows a node's transmission area), considers a topology with three stationary nodes, where the links between nodes `2` and `3` and between `3` and `1` are unidirectional. This scenario defines a single connected component in the sense that there is a directed path from each node to any other node. To form such a *unidirectional* but *connected* component, we set the transmission ranges of the source `1` and other two nodes `2` and `3` to $10\,m$, $30\,m$, and $20\,m$, respectively.

Real-Time Maude model checking shows (in 100 ms CPU time) that the desired property is satisfied in this topology with the above system parameters.

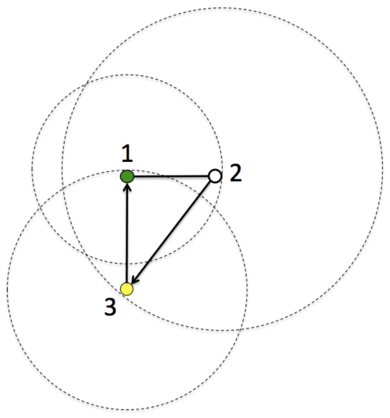

Fig. 3. Topology in Scenario II.

Scenario III. Scenario III, shown in Fig. 4 (where a dashed arrow denotes a node's motion path), considers a bidirectional *dynamic* topology with three nodes, where the source node `1` is located at `(50 . 50)`, and nodes `2` and `3` are initially at `(60 . 50)` and `(50 . 55)`, respectively. Node `3` is a *random waypoint* node that moves back and forth along the dashed arrow with end points `(50 . 55)` and `(60 . 55)` (denoted by `3'`). Note that the topology remains a connected component despite node `3`'s movement. We set the pause time of the moving node to 0 and the period of the neighbor/connectivity discovery process to 2.

We experiment with three sub-scenarios: (a) the speed of the moving node is 10; i.e., the `speedRange` attribute is the singleton `10`; (b) the speed is 5; and (c) the speed is again 10, but now the pause time is 1 time unit. The initial state of Scenario III-a is given by the term (with parts of the term replaced by '...'):

```
eq period = 2 .    eq pauseTime = 0 .
eq initConfig
 = electLeader(1)
   < 100 : GlobalND | timer : period >
   < 1 : SLENode | currentLocation : 50 . 50 ,  transRange : 10,  leader : 0,  max : 0,
                   neighbors : (2 ; 3),  parent : 0,  number : 100,  src : 0 ~ 0,
                   acks : empty,  eflag : false,  pflag : false,  newNbs : empty,
                   lostConxs : empty >
   < 2 : SLENode | currentLocation : 60 . 50 ,  transRange : 10,  leader : 0,  max : 0,
                   neighbors : 1,  parent : 0,   ...  >
   < 3 : RWPLENode | currentLocation : 50 . 55 ,  transRange : 10,  speed : 0,
                     speedVector : < 0 , 0 >,  speedRange : 10 ,
                     destRange : (60 . 55) ; (50 . 55) ,  timer : pauseTime,
                     status : pausing,  leader : 0,  neighbors : 1,   ...  > .
```

Real-Time Maude model checking of Scenario III-a shows (in 240 ms CPU time) that the desired property is *not* satisfied: Node `3` moves away from Node `1` *during* Node `1`'s multicast to "immediate neighbors," and is not within Node `1`'s

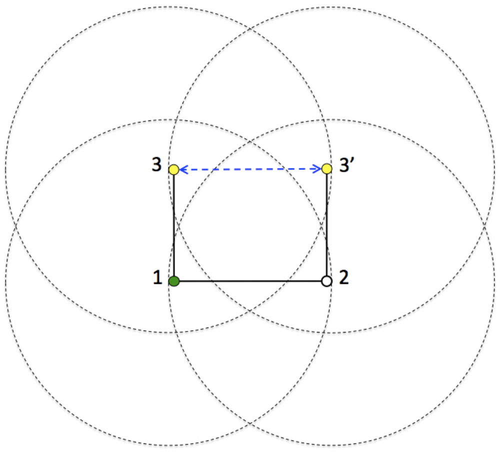

Fig. 4. Topology in Scenario III.

transmission range when the sending delay of the one-hop multicast of `election` messages to "immediate neighbors" *expires*. Therefore, Node `3` does not get this message. Furthermore, the neighbor discovery process takes place every 2 time units, which exactly coincides with the moments when Node `3` is close to Node `1`! The neighbor discovery process therefore never discovers that Node `3` is *not* an immediate neighbor of Node `1`, and will hence never discover that Node `3` is a *new* neighbor. Node `3` will therefore be left out of the election process forever.

We cannot claim that this behavior *invalidates* the LE protocol, since LE may be based on other assumptions, such as "continuous neighbor discovery" and/or multi-hop communication even to nodes that are immediate neighbors when a sending event begins. However, our "counterexample" shows the need to make explicit subtle requirements of the underlying neighbor discovery process, and to make more precise the meaning of sending to "immediate neighbors" when an immediate neighbor may cease to be one *during* the sending process.

Real-Time Maude model checking of Scenarios III-b and III-c show that the desired property holds in these scenarios. The only difference between Scenarios III-a and III-c is that `pauseTime` is 1 in Scenario III-c. This implies that Node `3` takes three time units to move from location `3` to location `3'`, and back. Since the neighbor discovery process takes place every two time units, it will sooner or later take place when Node `3` is in location `3'` in Fig 4, and hence no longer is an immediate neighbor of Node `1`. Some time later, the neighbor discovery will take place when Node `3` is again close to Node `1`, and will discover the "new" link between Nodes `1` and `3`, and will then involve Node `3` in the election.

In Scenario III-b, it takes Node `3` two time units to move between the locations `3` and `3'` in Fig. 4, and the neighbor discovery process will therefore update the neighbor information every time Node `3` reaches one of these end-points.

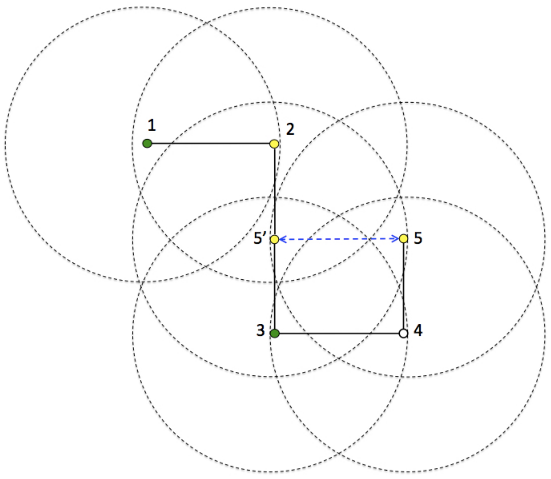

Fig. 5. Topology in Scenario IV.

Scenario IV. Finally, to analyze merge and partition of connected components we consider the system with two connected components (consisting of Nodes `1` and `2`, and of Nodes `3`, `4`, and `5`, respectively) in Fig. 5. Since Node `5` moves back and forth between position `5` and position `5'` in Fig. 5, the two connected components will repeatedly merge (when Node `5` is close to position `5'`) and partition (when Node `5` is close to position `5`).

Our model checking analysis shows that the property holds when both Node `1` and Node `3` initiate elections at the same time and when `pauseTime` is 8.

7 Related Work and Conclusions

LE has been subjected to a number of formal analysis efforts in recent years. Gelastou *et al.* [7] specify and verify LE using both I/O automata and process algebra. They only consider static bidirectional topologies with non-lossy channels, and communication delay is not taken into consideration. Singh *et al.* [36] present the ω-calculus for formally modeling and reasoning about MANETs, and illustrate their techniques by developing and analyzing a formal model of LE. They only consider dynamic bidirectional topologies where a node is free to move as long as the network remains connected, without taking into account unidirectional scenarios, communication delay, and message loss. Ghassemi *et al.* [10] provide a framework for modeling and analyzing both qualitative and quantitative aspects of MANET protocols, where communication delay and dynamic topologies (modeled by probabilistic message loss) are considered. They focus on the performance of LE under various parameters without giving any qualitative results. Sibilio *et al.* [35,23] propose a calculus for trustworthy MANETs with which they analyze a secure version of LE (neighbors trust each other at some security level) with three stationary nodes connected by bidirectional links. Kouzapas *et al.* [15] propose a calculus of dynamic networks whose semantics contain rules mimicking the behavior of a neighbor discovery protocol. They analyze LE with an arbitrary derivative of the initial state based on bisimilarity under the assumption of no message loss. Song *et al.* [37,38] introduce a stochastic broadcast calculus for MANETs with mobility modeled stochastically. They analyze a simplified model of LE with four nodes, where the mobility of (only) one node affects the transmission probability. In [8], Ghassemi *et al.* introduce both constrained labeled transition systems to represent mobility and a branching-time temporal logic to model check MANET protocols. They specify the correctness property of LE, but do not verify it in detail. Finally, the developers of LE present in their accompanying technical report [40] a "formal" specification of LE and use temporal logic to prove the correctness of the protocol, assuming bidirectional connections and no message loss.

Generally, [24,25,11,22,9,12,21,19,20] have also been proposed as process algebraic modeling languages for MANETs. These languages feature a form of local broadcast, in which a message sent by a node could be received by other nodes "within transmission range." However, the connectivity is only considered abstractly and logically, without taking into account concrete locations and transmission range for nodes. Also, these studies lack of either mobility modeling or timing issues handling.

The work presented in this paper distinguishes itself by modeling node locations, transmissions ranges, message loss, communication delay, well known mobility models, neighbor discovery, and uni/bidirectional connectivity, as well as their interrelations. From a modeling perspective:

- Related work does not model node locations explicitly, but represent the topologies abstractly as "neighborhood graphs."
- Related work therefore does not consider *realistic mobility models*, but only static topologies or simple dynamic topologies with arbitrary link breaks.
- Related work does not consider *unidirectional connectivity.*
- Our work is the only one that models a *neighbor discovery* process.
- Only [10,37,38] consider *communication delay.*
- No related work considers the *the interplay of all the above ingredients.*

Maude and Real-Time Maude have been applied to analyze wireless systems. Our previous work [16] builds the modeling framework that serves as the basis for this paper, and analyzes the Ad hoc On-Demand Distance Vector (AODV) routing protocol under different mobility models. The papers [13,33] model wireless sensor networks in (Real-Time) Maude, but do not consider node mobility.

In this work we have used rewriting logic and Real-Time Maude to address what we see as a real gap between the current application of formal methods to MANETs and actual practice. Specifically, there is a need to formally analyze MANET protocols with realistic models of node mobility and wireless communication. Our solution has taken the form of a composable formal framework in rewriting logic for MANET protocols and mobility models that can take into account time, space, directionality, and transmission failures and delays. We have illustrated the usefulness of this approach by showing how it can discover a potential problem with LE that is caused by the subtle interplay of neighbor discovery, communication with "immediate neighbors," and node movement *during* a communication event.

Much work remains ahead. We should apply our framework to the analysis of other MANET protocols and composition of protocols under various modes of use. Finally, by using probabilistic rewrite theories and statistical model checking in the style of [1], our framework could also be used for formal quantitative analysis of MANET protocols to evaluate their performance and reliability.

References

1. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electronic Notes in Theoretical Computer Science 153(2), 213–239 (2006)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Bouhoula, A., Jouannaud, J.P., Meseguer, J.: Specification and proof in membership equational logic. Theoretical Computer Science 236(1-2), 35–132 (2000)
4. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications and Mobile Computing 2(5), 483–502 (2002)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude, LNCS, vol. 4350. Springer (2007)
6. Fall, K., Varadhan, K.: The *ns* Manual (2011), `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`
7. Gelastou, M., Georgiou, C., Philippou, A.: On the application of formal methods for specifying and verifying distributed protocols. In: Proc. NCA'08. IEEE (2008)
8. Ghassemi, F., Ahmadi, S., Fokkink, W., Movaghar, A.: Model checking MANETs with arbitrary mobility. In: Proc. FSEN'13. LNCS, vol. 8161. Springer (2013)
9. Ghassemi, F., Fokkink, W., Movaghar, A.: Restricted broadcast process theory. In: Proc. SEFM '08. IEEE (2008)
10. Ghassemi, F., Talebi, M., Movaghar, A., Fokkink, W.: Stochastic restricted broadcast process theory. In: Proc. EPEW'11. LNCS, vol. 6977. Springer (2011)
11. Godskesen, J.C.: A calculus for mobile ad hoc networks. In: Proc. Coordination'07. LNCS, vol. 4467. Springer (2007)
12. Godskesen, J.C., Nanz, S.: Mobility models and behavioural equivalence for wireless networks. In: Proc. Coordination'09. LNCS, vol. 5521. Springer (2009)
13. Katelman, M., Meseguer, J., Hou, J.C.: Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In: Proc. FMOODS'08. LNCS, vol. 5051. Springer (2008)
14. Kosiuczenko, P., Wirsing, M.: Timed rewriting logic with an application to object-based specification. Science of Computer Programming 28(2-3), 225–246 (1997)
15. Kouzapas, D., Philippou, A.: A process calculus for dynamic networks. In: Proc. FMOODS/FORTE'11. LNCS, vol. 6722. Springer (2011)
16. Liu, S., Ölveczky, P., Meseguer, J.: A framework for mobile ad hoc networks in Real-Time Maude. In: Proc. WRLA'14. LNCS, vol. 8663. Springer (2014)
17. Liu, S., Ölveczky, P.C., Meseguer, J.: Formal analysis of leader election in MANETs using Real-Time Maude (2014), `http://www.ifi.uio.no/RealTimeMaude/leader-election-report.pdf`

18. Liu, S., Ölveczky, P.C., Meseguer, J.: Modeling and analyzing mobile ad hoc networks in Real-Time Maude (2015), submitted for publication, `http://www.ifi.uio.no/RealTimeMaude/manets-report.pdf`
19. Liu, S., Wu, X., Li, Q., Zhu, H., Wang, Q.: Formal approaches to wireless sensor networks. In: Fifth International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, - Companion Volume. pp. 11–18. IEEE Computer Society (2011)
20. Liu, S., Zhao, Y., Zhu, H., Li, Q.: A calculus for mobile ad hoc networks from a group probabilistic perspective. In: 13th IEEE International Symposium on High-Assurance Systems Engineering, HASE 2011. pp. 157–162. IEEE Computer Society (2011)
21. Liu, S., Zhao, Y., Zhu, H., Li, Q.: Towards a probabilistic calculus for mobile ad hoc networks. In: 5th IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2011. pp. 195–198. IEEE Computer Society (2011)
22. Merro, M.: An observational theory for mobile ad hoc networks (full version). Inf. Comput. 207(2), 194–208 (2009)
23. Merro, M., Sibilio, E.: A calculus of trustworthy ad hoc networks. Formal Aspects of Computing 25(5), 801–832 (2013)
24. Mezzetti, N., Sangiorgi, D.: Towards a calculus for wireless systems. Electron. Notes Theor. Comput. Sci. 158, 331–353 (2006)
25. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theor. Comput. Sci. 367(1), 203–227 (2006)
26. Ölveczky, P.C.: Real-Time Maude and its applications. In: Proc. WRLA'14. LNCS, vol. 8663. Springer (2014)
27. Ölveczky, P., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-order and Symbolic Computation 20(1-2), 161–196 (2007)
28. Ölveczky, P.C.: Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic. Ph.D. thesis, University of Bergen, Norway (2000), `http://maude.csl.sri.com/papers`
29. Ölveczky, P.C., Kosiuczenko, P., Wirsing, M.: An object-oriented algebraic steam-boiler control specification. In: Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, LNCS, vol. 1165. Springer (1996)
30. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science 285, 359–405 (2002)
31. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. Electronic Notes in Theoretical Computer Science 176(4), 5–27 (2007)
32. Ölveczky, P.C., Meseguer, J., Talcott, C.L.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods in System Design 29(3), 253–293 (2006)
33. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theoretical Computer Science 410(2-3), 254–280 (2009)
34. OMNeT++. `http://www.omnetpp.org/`, accessed November 24, 2014
35. Sibilio, E.: Formal methods for wireless systems. Ph.D. Thesis, University of Verona (2011)
36. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A process calculus for mobile ad hoc networks. Science of Computer Programming 75(6), 440–469 (2010)
37. Song, L.: Probabilistic models and process calculi for mobile ad hoc networks. Ph.D. Thesis, IT University of Copenhagen (2012)

38. Song, L., Godskesen, J.C.: Broadcast abstraction in a stochastic calculus for mobile networks. In: Proc. TCS'12. LNCS, vol. 7604. Springer (2012)
39. Su, P.: Delay measurement time synchronization for wireless sensor networks. Tech. Rep. IRB-TR-03-013, Intel Research Berkeley Lab (2003)
40. Vasudevan, S., Kurose, J.F., Towsley, D.F.: Design and analysis of a leader election algorithm for mobile ad hoc networks. Tech. Rep. UMass CMPSCI 03-20, University of Massachusetts (2003)
41. Vasudevan, S., Kurose, J.F., Towsley, D.F.: Design and analysis of a leader election algorithm for mobile ad hoc networks. In: Proc. ICNP'04. IEEE (2004)