View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

A Formal Framework for Mobile Ad hoc Networks in Real-Time Maude

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

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

Abstract. Mobile ad hoc networks (MANETs) are increasingly popular and deployed in a wide range of environments. However, it is challenging to formally analyze a MANET, both because there are few reasonably accurate formal models of mobility, and because the large state space caused by the movements of the nodes renders straight-forward model checking hard. In particular, the combination of wireless communication and node movement is subtle and does not seem to have been adequately addressed in previous formal methods work. This paper presents a formal executable and parameterized modeling framework for MANETs in Real-Time Maude that integrates several mobility models and wireless communication. We illustrate the use of our modeling framework with the Ad hoc On-Demand Distance Vector (AODV) routing protocol, which allows us to analyze this protocol under different mobility models.

1 Introduction

A *mobile ad hoc network* (MANET) is a self-configuring network of mobile devices (laptops, smart phones, sensors, etc.) that communicate wirelessly and cooperate to provide the necessary network functionality. Since MANETs can form ad hoc networks without fixed infrastructure, they are supposed to have a wide applicability, for example for providing ad hoc networks for cooperating "smart" cars, for emergency responders during accidents, during natural disasters which may disable fixed infrastructure, in battlefield areas, and so on.

Although many such applications are safety-critical and need formal analysis to ensure their correctness, the formal modeling and analysis of MANETs present a number of challenges that include:

1. The need to model node movement realistically.
2. Modeling communication. There is a subtle interaction between wireless communication, which typically is restricted to distances of between 10 and 100 meters, and node mobility. For example, nodes may move into or out of the sender's transmission range *during* the communication delay; furthermore, the sender may itself move during the communication. Modeling communication in MANETs is therefore challenging for formal languages, which are usually based on fixed communication primitives.

3. Since the communication topology of the network depends on the *locations* of the nodes, such locations must be taken into account in the model. However, this leads to very large state spaces, which makes direct model checking analysis unfeasible: if there are m nodes and n locations, there are n^m different nodes/locations states. A 10×10 grid with four nodes would therefore lead to 100 million states just to capture all nodes and their locations.

As explained in Section 7, we are not aware of any formal model that provides a reasonably detailed model of both mobility and communication in MANETs. Given its expressiveness and flexibility to define models of communication, Real-Time Maude [23] seems to be a promising language for formally modeling MANETs. In this paper we provide, to the best of knowledge, the first reasonably precise formal modeling framework for MANETs. In particular, we formalize

– the most popular models for node mobility, and
– geographically bounded wireless communication, which takes into account the interplay between communication delay and mobility,

in Real-Time Maude. Furthermore, we use object-oriented techniques to make it easy to *compose* our framework with a model of a MANETs protocol.

Concerning Challenge 3 above, in this paper we do not develop abstraction techniques for node mobility. Instead, to be able to perform model checking analysis, our model is parametric in aspects such as the possible velocities and directions a node can choose. However, even if a node moves slowly, it may still cover the entire area (and hence contribute to an unmanageable state space) given enough time. Another key feature of Real-Time Maude that makes some meaningful model checking analysis of MANETs possible is therefore *time-bounded* model checking, which allows us to analyze scenarios only up to a certain duration (during which the nodes may not reach most locations). Abstracting the state space caused by node mobility and the need to keep track of node locations is the *sine qua non* for serious model checking of MANETs. The point is that this paper lays the necessary foundations for developing such abstractions by providing a first reasonably detailed formal model of location-aware MANETs.

One of the main tasks of a MANET is to maintain an (ad hoc) network, which means that the network must figure out how to route messages between nodes. In this paper we illustrate the use of our MANETs framework by modeling and analyzing the widely used *Ad hoc On-Demand Distance Vector* [25] (AODV) routing protocol for MANETs developed by the IETF MANET working group.

The rest of this paper is organized as follows. Section 2 gives a background to Real-Time Maude. Section 3 briefly introduces MANETs. Section 4 presents our Real-Time Maude modeling framework for MANETs. Section 5 shows how our framework can be used to model the AODV protocol, and Section 6 explains how that model of AODV can be model checked using Real-Time Maude. Finally, Section 7 discusses related work and Section 8 gives some concluding remarks.

2 Real-Time Maude

Real-Time Maude [23] is a language and tool that extends Maude [6] 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* [23] $(\Sigma, E \cup A, IR, TR)$, where:

- Σ is an algebraic *signature*; that is, a set of declarations of *sorts*, *subsorts*, and *function symbols.*
- $(\Sigma, E \cup A)$ is a *membership equational logic theory* [2], with E a set of possibly conditional equations, and A a set of equational axioms such as associativity, commutativity, and identity, so that equational deduction is performed *modulo* the axioms A. $(\Sigma, E \cup A)$ 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** $\bigwedge_{j=1}^{m} cond_j$, 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`}` $\xrightarrow{\tau}$ `{`t'`}` **if** $cond$ that advance time in the *entire* state t by τ time units.

We refer to [6] for the syntax of Real-Time Maude. We briefly summarize the syntax of Real-Time Maude and refer to [6] for more details. Operators are introduced with the `op` keyword: `op` f `:` $s_1 \dots s_n$ `->` s. They can have user-definable syntax, with underbars '`_`' marking the argument positions. Some operators can have equational *attributes*, such as `assoc`, `comm`, and `id`, stating, for example, that the operator is associative and commutative and has a certain identity element. Such attributes are used by the Maude engine to match terms *modulo* the declared axioms. An operator can also be declared to be a constructor (`ctor`) that defines the carrier of a sort. 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`, or can be introduced on the fly in a statement without being declared previously, in which case they have the form *var*`:`*sort*. An equation $f(t_1, \dots, t_n) = t$ with the `owise` (for "otherwise") attribute can be applied

[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.

to a subterm $f(\ldots)$ 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 >
         =>
          < O : C | a1 : x + w, a2 : O', a3 : z >
          dly(m'(O',x), z) .
```

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 `dly(m'(O',x),z)` is generated; this message will become the "ripe" message `m'(O',x)` after `z` 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 fair 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

(`tsearch [`*n*`]` *t* `=>*` *pattern* `such that` *cond* `in time <=` *timeLimit* `.)`

uses a breadth-first strategy to search for (at most n) states that are reachable from the initial state t within a time *timeLimit*, match the *search pattern*, and satisfy the *search condition*.

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. The time-bounded model checking command

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

checks whether the temporal logic formula *formula* holds in all behaviors up to duration *timeLimit* starting from the initial state t.

3 Mobility and Communication Delay in MANETs

This section gives an overview of the main mobility models used by researchers on protocol evaluations, and of the per-hop delay in wireless communication.

3.1 Mobility Models

A number of different *mobility patterns* have been proposed to model node mobility in realistic scenarios. Such models of mobility include *entity mobility models*, where a node's movement is independent of the movements of the other nodes, and *group mobility models*, where the nodes' movements depend on each other. In this paper we focus on entity mobility models.

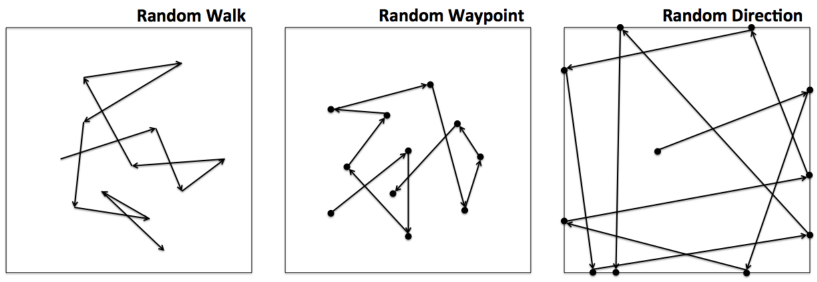

Fig. 1. Motion paths of a mobile node in three mobility models, where a bullet • depicts a pause in the movement.

The following main entity mobility models [3] are illustrated in Fig. 1:

- *Random Walk:* Each node moves in "rounds" of fixed durations. A node moves in the same direction and with the same speed throughout one "round". At the end of each round, the *new speed* and the *new direction* of a node are randomly chosen, and a new moving round starts.
- *Random Waypoint:* Each node initially pauses for a fixed duration. When a pause ends, a node randomly chooses a *new destination* and a *new speed*, and then travels to that destination at the chosen speed. After arriving, the node again pauses before a new moving round starts.
- *Random Direction:* Each mobile node chooses a *random direction*, along which it travels until reaching the border of the sensing area. When a node arrives at the border, the node pauses for a given time, and then randomly selects a new direction and starts to move in that direction.

3.2 Communication Delay

To understand how node movement affects wireless communication, it is necessary to understand the nature of messaging delays in wireless communication. In a typical wireless transmit/receive process, the per-hop delay, i.e., the communication delay from a transmitter to a receiver, consists of the following five parts [27]:

Delay Factor	**Description**
Sender Processing Delay	The duration elapsed on the sender side from the moment a message timestamp is taken to the point the message is buffered in the device.
Media Access Delay	The duration for a message to stay in the radio device buffer; e.g., in a CSMA system, this is the delay waiting for a clear channel to transmit.
Transmit Delay	The duration for a radio device to transmit a message over a radio link.
Radio Propagation Delay	The duration for a message to propagate through the air to a receiver.
Receiver Processing Delay	The duration spent on the receiver side to pass the received message from the device buffer to the application module.

We can typically abstract from the radio propagation delay, 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. The media access delay is an uncertainty, depending on the MAC overhead, such as collisions and waiting time.

4 Formalizing MANET Mobility and Communication in Real-Time Maude

This section presents a modeling framework for MANETs with nodes that communicate wirelessly. The combination of wireless communication and mobility is challenging, because both the sender and the potential receivers may be moving *during* the communication delay, and could, for example, enter or exit the area within the transmission range of the sender during the delay. Section 4.2 shows how mobile nodes can be specified in Real-Time Maude, Section 4.3 explains how the time behavior of MANETs can be defined in a way that allows us to easily compose our MANETs model with MANET protocols, and Section 4.4 shows how wireless communication for MANETs can be formalized.

4.1 Some Basic Data Types

We assume a sort `Location` for the set of locations, a sort `Speed` for the different velocities with which a node can move, a set `Direction` for the different directions that a node can choose, and sorts SpeedRange, DirRange, and DestRange denoting *sets* of, respectively, Speed, Direction, and Location elements.

Although our framework is parametric in these domains, in this paper we assume for simplicity that nodes move in a two-dimensional square with length areaSize. A location is therefore represented as a pair $x\,.\,y$ of rational numbers:[2]

```
op _._ : Rat Rat ~> Location [ctor] .

cmb X . Y : Location
    if 0 <= X and X <= areaSize /\ 0 <= Y and Y <= areaSize .
```

We do not further specify the different powersets, whose elements could be unions of dense intervals or of single points, or both. Since the nodes may need to nondeterministically select a new speed, a new next destination, and/or a new next direction, we assume for generality's sake that there is an operator choose that can select any value in the respective set nondeterministically:

```
op choose : SpeedRange -> Choice [ctor] .
op choose : DestinationRange -> Choice [ctor] .
op choose : DirRange -> Choice [ctor] .
op [_] : Speed -> Choice [ctor] .
op [_] : Location -> Choice [ctor] .
op [_] : Direction -> Choice [ctor] .
```

We assume that an element e can be chosen from a set S if and only if there is a rewrite (in zero or more steps) choose(S) => [e]. For example, if we have a discrete set of possible next directions d_1 ; d_2 ; ...; d_n, where the set union operator _;_ is declared to be associative and commutative, we can specify that any value from the set can be selected, by giving the following rewrite rule:

```
var D : Direction .   var DR : DirRange .
rl [chooseDir] :  choose(D ; DR) => [D] .
```

4.2 Modeling Mobile Nodes

We model a MANET in an object-oriented style, where a mobile node is modeled as an object instance of some subclass of the following base class Node:

```
class Node | currentLocation : Location .
```

The attribute currentLocation denotes the node's current location. Any node that is *not* mobile is an object of the subclass StationaryNode that does not add any attribute to the base class:

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

[2] We do not show most variable declarations, but follow the Maude convention that variables are written in capital letters.

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

```
class MobileNode | speed : Speed,  direction : Direction,  timer : TimeInf .
subclass MobileNode < Node .
```

where `speed` and `direction` denote, respectively, the node's current speed and its current movement direction. The `timer` attribute is used to ensure that a node changes its movement (or lack thereof) in a timely manner; that is, `timer` denotes the time remaining until some discrete event must take place.

We are now ready to define the different mobility models.

Random Walk. A node moving according to the random walk model is continuously moving, in time intervals of length `movingTime`. At the end of an interval, the node nondeterministically chooses a new speed and a new direction for its next interval. Such a node is modeled by an object of the `RWNode` subclass:

```
class RWNode | speedRange : SpeedRange, dirRange : DirRange .
subclass RWNode < MobileNode .
```

where `speedRange` and `dirRange` denote the set of possible next speeds and directions, respectively. The `timer` attribute inherited from its superclass denotes the time remaining of its current move interval. The instantaneous behavior of the mobility part of such a node can be modeled by the following rule. In this rule, the node is finishing one interval (the `timer` attribute is 0), and must select new speed and direction for its next round, and reset the timer:

```
crl [startNewMove] :
   < O : RWNode | timer : 0, speedRange : SR,  dirRange : DR >
 =>
   < O : RWNode | timer : movingTime, speed : S, direction : D > .
  if choose(SR) => [S]  /\ choose(DR) => [D] .
```

The actual movement of such a node is modeled in Section 4.3.

Random Waypoint. In the random waypoint mobility model, a node alternates between pausing and moving. When it starts moving, it selects a new speed and a new destination and starts moving towards the destination. Such a node should be modeled by an object instance of the `RWPNode` subclass:

```
class RWPNode | speedRange : SpeedRange,   destRange : DestRange,
                status : Status .
subclass RWPNode < MobileNode .
```

The `status` attribute is either `pausing` or `moving`, and `destRange` denotes the range of possible goal locations.

The instantaneous behavior of this mobility model is given by the following rewrite rules. First, if the node is `pausing` and the `timer` expires, the node must get moving by selecting a new speed and desired next location, and resetting the timer so that it expires when the goal location is reached:

```
var MOVE-TIME : Time .

crl [startMoving] :
    < O : RWPNode | currentLocation : CURR-LOC, status : pausing, timer : 0,
                    speedRange : SR,  destRange : DER >
   =>
    < O : RWPNode | status : moving, speed : S,
                    direction : D,  timer : MOVE-TIME >
  if choose(SR) => [S]
     /\ choose(DER) => [NEXT-LOC]
     /\ D := direction(L, NEXT-LOC)
     /\ MOVE-TIME := timeBetweenLocations(CURR-LOC, NEXT-LOC, S) .
```

where `direction` gives the direction from one location to another, and `time-BetweenLocations` denotes the time it takes to travel between two locations at a given speed. Notice that the selected speed cannot be zero, unless the selected next location is also the current location, because then the last matching equation would not hold, since the traveling time between the two locations would be the infinity value `INF`, which is not a `Time` value.

The following rule applies when the timer of a *moving* node expires; then it is time to take a rest for `pauseTime` time units:

```
rl [startPausing] :
   < O : RWPNode | status : moving, timer : 0 >
  =>
   < O : RWPNode | status : pausing, timer : pauseTime, speed : 0 > .
```

Random Direction. A node that moves according to the random direction model nondeterministically chooses a direction and a speed, and walks in the given direction until it reaches the boundary of the area. It then pauses for some time before starting a new walk. Nodes following this mobility pattern should be declared as instances of the `RDNode` subclass:

subclasses of the following class

```
class RDNode | speedRange : SpeedRange,   dirRange : DirRange,
               status : Status .
subclass RDNode < MobileNode .
```

Its instantaneous behaviors are formalized by two rewrite rules; the first one chooses a new direction and speed when the node has paused enough:

```
crl [newRDwalk] :
    < O : RDNode | currentLocation : CURR-LOC, speedRange : SR,
                   dirRange : DR, timer : 0, status : pausing >
   =>
    < O : RDNode | status : moving, speed : S,
                   direction : D, timer : MOVE-TIME >
  if choose(SR) => [S]  /\ choose(DR) => [D]
     /\ NEW-GOAL-LOC := borderLocation(CURR-LOC, D)
     /\ NEW-GOAL-LOC =/= CURR-LOC
     /\ MOVE-TIME := timeBetweenLocations(CURR-LOC, NEW-GOAL-LOC, S) .
```

The second-to-last conjunct in the condition ensures that the selected direction leads inwards towards the area of operation.

The second rule models the stage when a moving node starts pausing:

```
rl [startPausing] :
   < O : RDNode | status : moving, timer : 0 >
  =>
   < O : RDNode | status : pausing, timer : pauseTime, speed : 0 > .
```

4.3 Timed Behavior and Compositionality

Our model of mobile nodes must be easily *composable* with "application" protocols such as AODV to define a particular MANET system. The straight-forward way of composing our model of mobility with a MANET protocol is to let the nodes in the application protocol be modeled as objects of subclasses of the classes introduced above, since a subclass "inherits" all the attributes and rewrite rules of its superclasses; in particular, such application-specific subclasses would inherit the rewrite rules modeling the movements of their nodes.

However, we must allow the user to define the *timed behavior* of her system, and compose it with the timed behavior of mobile nodes. We therefore use the following extension of the "standard" tick rule for object-oriented specifications

```
var T : Time .    var C : Configuration .
crl [tick] : {C} => {timeEffect(timeEffectMob(C, T), T)} in time T
             if T <= min(mte(C), mteMob(C)) .
```

where `timeEffectMob` defines the effect of time elapse on the mobility-specific parts of the system, and `timeEffect` defines how the passage of a certain amount of time changes the state in the parts of the composed system that does not deal with node mobility. Likewise, `mteMob` denotes the maximum amount of time that may elapse from a given state until some mobility action must be taken, and `mte` defines the amount of time until the application protocol must perform a discrete action. These functions should distribute over the single objects and messages in the configuration as follows:

```
vars NECF1 NECF2 : NEConfiguration .

ops timeEffectMob timeEffect : Configuration Time -> Configuration [frozen (1)] .
ops mte mteMob : Configuration -> TimeInf [frozen (1)] .

eq timeEffectMob(none, T) = none .
eq timeEffect(none, T) = none .
eq timeEffectMob(NECF1 NECF2, T)
  = timeEffectMob(NECF1, T)  timeEffectMob(NECF2, T) .
eq timeEffect(NECF1 NECF2, T) = timeEffect(NECF1, T)  timeEffect(NECF2, T) .
eq mte(NECF1 NECF2) = min(mte(NECF1), mte(NECF2)) .
eq mteMob(NECF1 NECF2) = min(mteMob(NECF1), mteMob(NECF2)) .
```

That is, the "user" can specify his protocol without worrying about having to model the node's mobility, and should take care about defining `timeEffect` and `mte` for the "application-specific" timed features.

Since we set the speed to 0 when a node is pausing, we can easily define the timed behavior of both stationary and mobile nodes. First of all, time does not affect (the mobility-specific parts of) a stationary node:

```
eq timeEffectMob(< O : StationaryNode | >, T)  =  < O : StationaryNode | > .
```

Time affects a mobile node by moving the node and decreasing its timer value:

```
eq timeEffectMob(< O : MobileNode | currentLocation : L, speed : S,
                                    direction : D, timer : T1 >, T)
 = < O : MobileNode | currentLocation : move(L,S,D,T), timer : T1 monus T > .
```

where `move(`l`,`s`,`d`,`t`)` denotes the location resulting from moving a node in location l for t time units in direction d and with speed s. This function also makes sure that a node does not move beyond the area under consideration.

The mobility model does not restrict the time advance for stationary nodes, whereas for mobile nodes, time can advance until the `timer` becomes 0:

```
eq mteMob(< O : StationaryNode | >) = INF .
eq mteMob(< O : MobileNode | timer : T >) = T .
```

Finally, the following equations take care of messages and of objects that are not mobile or stationary nodes, in case such extra objects are introduced by the application:

```
eq mteMob(OBJECT) = INF [owise] .
eq timeEffectMob(OBJECT, T) = OBJECT [owise] .
eq mteMob(MSG) = INF .
eq timeEffectMob(MSG, T) = MSG .
```

4.4 Modeling Wireless Communication in Mobile Systems

Finally, we need to model wireless communication in mobile systems. Typically only nodes that are sufficiently close to the sender, i.e., within the sender's *transmission range*, receive the message with sufficient signal strength. However, both the sender and the potential receivers might move (possibly out of, or into, the sender's transmission range) *during* the entire communication delay.

As mentioned in Section 3, the total communication "delay" can be decomposed into five parts. However, 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 (including sender processing delay, media access delay and transmit delay) and the delay at the receiver side (including receiver processing delay). 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.

It is also worth mentioning that our model is still somewhat abstract and does not capture all network factors, most notably collisions.

In MANETs communication can be by broadcast, unicast, or groupcast, depending on which kind of message a transmitter intends to send, and who are the recipients. In our model we have three corresponding message constructors for broadcast, unicast, and groupcast, respectively:

```
msg broadcast_from_ : MsgCont Oid -> Msg .
msg unicast_from_to_ : MsgCont Oid Oid -> Msg .
msg gpcast_from_to_ : MsgCont Oid NeighborSet -> Msg .
```

where `Oid` is the identifier of a node; `NeighborSet` is a set of nodes that should be informed as a group; and `MsgCont` is the sort for message contents.

When a node *sender* wants to broadcast some message content *mc*, it generates a "message" `broadcast` *mc* `from` *sender*. The following equation adds the delay on the sending side, `sendDelay`, to this "broadcast message:"

```
eq broadcast MC from O = dly(transmit MC from O, sendDelay) .
```

The crucial moment is when the sending delay expires and the `transmit` message becomes "ripe." All the nodes that are within the transmission range of the sender *at that moment* should receive the message. This distribution is performed by the function `distrMsg`, where `distrMsg(`*snd*, *loc*, *mc*, *conf*`)` generates a *single* message, with content *mc*, to each node in *conf* that is currently within the transmission range of location *loc*; furthermore, this single message has delay `recDelay` modeling the delay at the receiving site:

```
eq {< O : Node | currentLocation : L >  (transmit MC from O) C}
 = {< O : Node | >   distrMsg(O, L, MC, C)} .

eq distrMsg(O, L, MC, < O' : Node | currentLocation : L' > C)
 = < O' : Node | currentLocation : L' > distrMsg(O, L, MC, C)
   (if L withinTransRangeOf L'
         then dly((MC from O to O'), recDelay)
    else none fi) .
```

Unicast and groupcast are modeled similarly.

5 Case Study: Route Discovery in AODV

This section first gives an overview of the Ad hoc On-Demand Distance Vector (AODV) routing protocol, and then presents our Real-Time Maude model of AODV, focusing on the route discovery process. The entire executable Real-Time Maude specification is available at `http://folk.uio.no/RealTimeMaude/MANET`.

5.1 Route Discovery in AODV

AODV [25] is a widely used algorithm for routing messages between mobile nodes which dynamically form an ad hoc network. AODV allows a source node not to maintain any routing information but instead to initiate a route discovery process based on an on-demand mechanism to establish a route to a destination node.

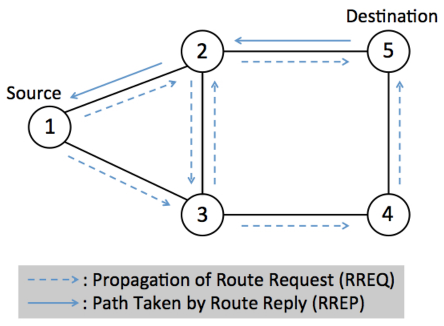
Destination
Source
1
2
3
4
5
- - - ->: Propagation of Route Request (RREQ)
——→: Path Taken by Route Reply (RREP)

Fig. 2. Route discovery process.

When necessary, a source node `S` initiates a route discovery process by broadcasting a route request (RREQ) message to its neighbors. An intermediate node can either unicast a route reply (RREP) message back to the source if a valid route to the destination `D` can be found in its local routing table, or re-broadcast the received RREQ to its own neighbors. As the RREQ travels from `S` to `D`, reverse paths from all nodes back to `S` are automatically set up. Eventually, when the RREQ reaches `D`, it sends a RREP back along the previously established reverse path. After this process, a route between `S` and `D` is set up, along which further packets can be delivered. An intermediate node that receives multiple RREQ messages drops the subsequent ones if the same RREQ was recorded previously. To ensure loop-free routing, AODV employs a sequence number to represent how fresh the received information is. The higher a sequence number is, the fresher a route will be. Therefore a requesting node is required to select the one with the greatest sequence number.

5.2 Modeling AODV Nodes and Messages

We model an AODV node as an object of a subclass `AODVNode` of class `Node`. The new attributes show the identification of a node's routing request, the sequence number of a node itself, the local routing table, and the buffered routing requests sent since the beginning of the current round, respectively. Since the routing protocol is aimed at finding a route between two nodes, we can check if this has been achieved by looking up a special routing table entry for the destination node in the local routing table after the current round.

```
class AODVNode | rreqID : Nat, sequenceNumber : Sqn,
                 routingTable : RouteTable, requestBuffer : RreqBuffer .
subclass AODVNode < Node .
```

A routing table of sort `RouteTable` is modeled using the predefined data type `MAP` in Maude, consisting of routing table entries of the form `Oid |-> Tuple3`, where `Oid` refers to a destination node; and `Tuple3` refers to a tuple of three elements in a routing table entry: the next hop towards the destination, the distance to the destination, and the local destination sequence number. A route request buffer of the sort `RreqBuffer` is specified as a set of requests of the sort `RreqID` that is of the form `Oid ~ Sqn`, uniquely identifying a route request by the identifier of a node and its sequence number.

In the AODV route discovery process there are mainly two kinds of messages, i.e., RREQ by broadcast and RREP by unicast. They are specified in our model as `rreq(...)` and `rrep(...)` respectively. The message content will be illustrated below.

AODV is demand-driven and does not impose additional timing constraints:

```
eq timeEffect(< O : AODVNode | >, T)  =  < O : AODVNode | > .
eq mte(< O : AODVNode | >) = INF .
```

5.3 Modeling Route Discovery in Real-Time Maude

In our model a route discovery process of AODV consists of three parts: initiating route discovery, route request handling, and route reply handling.

Initiating Route Discovery. At the start of the route discovery process, an originator is bootstrapped to look up its local routing table for a special entry towards the destination. If that exists, the originator is in fact ready to deliver the data; otherwise, it initiates the route discovery process by broadcasting a `rreq(O,SN + 1,RREQID,DIP,0,0,O)` message. Since in that case the originator does not know the sequence number of the destination or the distance from it, we set both elements to `0`. Before broadcasting, the originator needs to increase the local routing request ID, as well as its own sequence number, and add the outgoing RREQ to the request buffer.

```
 rl [init-route-discovery] :
    (bootstrap O)
    < O : AODVNode | rreqID : RREQID, sequenceNumber : SN,
                     routingTable : RT, requestBuffer : RB >
  =>
    if inRT(RT,DIP)
      then < O : AODVNode | > (msg PKT from O to DIP)
      else < O : AODVNode | rreqID : RREQID + 1, sequenceNumber : SN + 1,
                            requestBuffer : (O ~ RREQID, RB) >
           (broadcast rreq(O,SN + 1,RREQID,DIP,0,0,O) from O) fi .
```

Route Request Handling. The RREQ-handling rules specify all events that may happen when a route request is received. The receiving node first checks whether a received `(OIP ~ RREQID)` has already been stored locally in the request buffer. If so, the route request can be ignored and the local routing table is updated by adding a routing table entry towards the sender; otherwise, the receiving node adds the new route request identifier to the request buffer, and takes further actions according to the following policy.

On the one hand, if the receiving node `O` is actually the intended destination (`DIP == O`), a route reply message `rrep(OIP,DIP,SN',0,O)` should be generated. `SN'` is actually the maximum of the current sequence number and the destination sequence number in the RREQ according to [25]. The hop count is obviously `0`. The destination should also update the routing table entry for the sender, as well as the source node, in its local routing table `RT` respectively, `RT' := update(SIP,SIP,1,0,RT)` and `RT'' := update(OIP,SIP,HOPS + 1,OSN,RT')`. Then the RREP is unicast to the next hop along the route back to the source node `nexthop(RT''[OIP])` by looking up the newly updated routing table `RT''`.

On the other hand, if the receiving node `O` is not the destination `DIP` but an intermediate node, it either: (a) generates a route reply to the sender, or (b) re-broadcasts the received RREQ to its neighbors. Action (a), as the following rewrite rule shows, happens only when `O`'s local information is fresher than that in the RREQ; that is, its local sequence number of the destination is greater than or equal to the received sequence number (`DSN <= localdsn(RT[DIP])`). In this case, `O` unicasts the route reply with the fresher destination sequence number and its distance in hops from the destination along the route back to the source node.

Action (b) happens if local information is not fresh enough, or no route table entry for `DIP` can be found. `O` should then re-broadcast the received RREQ with the hops increment and the maximal destination sequence number in the message content.

```
crl [on-receiving-rreq-3] :
    (rreq(OIP,OSN,RREQID,DIP,DSN,HOPS,SIP) from SIP to O)
     < O : AODVNode | routingTable : RT, requestBuffer : RB >
   =>
     < O : AODVNode | routingTable : RT'',
                      requestBuffer : (OIP ~ RREQID, RB) >
    (msg rrep(OIP,DIP,localdsn(RT''[DIP]),hops(RT''[DIP]),O)
       from O to nexthop(RT''[OIP]))
    if RT' := update(SIP,SIP,1,0,RT)
       /\ RT'' := update(OIP,SIP,HOPS + 1,OSN,RT')
       /\ not (OIP ~ RREQID) in RB /\ inRT(RT,DIP)
       /\ DIP =/= O /\ DSN <= localdsn(RT[DIP]) .
```

Route Reply Handling. The RREP-handling rules describe all events that may happen when a route reply is received. If a receiver is actually the originator, the RREP can be resolved and a route table entry for the destination is created or updated. We can consider these cases as either: (a) the destination sequence number in the originator's existing routing table is smaller then the one in the received RREP; or (b) the two destination sequence numbers are the same but the increased hop count in the received RREP is smaller than the one in the local routing table `hops(RT[DIP]) > HOPS + 1`; or (c) no route table entry for the destination can be found locally. In all other cases, the received RREP can be silently ignored. The following rewrite rule shows the handling of case (b) as an example:

```
crl [on-receiving-rrep-4] :
    (rrep(OIP,DIP,DSN,HOPS,SIP) from SIP to O)
    < O : AODVNode | routingTable : RT >
   =>
    < O : AODVNode | >
    if OIP == O /\ inRT(RT,DIP) /\ DSN == localdsn(RT[DIP])
       /\ hops(RT[DIP]) <= HOPS + 1 .
```

When a receiver is an intermediate node, the RREP can be forwarded if any of the above cases (a), (b) or (c) can be satisfied, or can be silently ignored otherwise. The following rewrite rule shows case (c) as an example:

```
crl [on-receiving-rrep-6] :
    (rrep(OIP,DIP,DSN,HOPS,SIP) from SIP to O)
    < O : AODVNode | routingTable : RT >
   =>
    < O : AODVNode | routingTable : RT' >
    (msg rrep(OIP,DIP,DSN,HOPS + 1,O) from O to nexthop(RT'[OIP]))
    if OIP =/= O /\ not inRT(RT,DIP)
       /\ RT' := update(DIP,SIP,HOPS + 1,DSN,RT) .
```

6 Formal Analysis of AODV

In this section we explain how specifications based on our mobility framework can be combined to analyze the AODV route discovery process under various mobility models. We construct several scenarios to investigate how different mobility models influence the performance of our target routing protocol.

6.1 Motivation

In protocol design for MANETs, the choice of appropriate mobility models has always been considered as a critical factor for the successful evaluation of protocols. In terms of using formal approaches to model and analyze MANET protocols, mobility is also given an essential role. However, protocols in general, and AODV in particular, have so far not been formally analyzed under realistic mobility models, but in static topologies where some random link failures were consideredd, or under an arbitrary mobility setting. Thus, very little is known by way of formal analysis about how AODV behaves under realistic mobility. Besides, few studies have taken into account communication delay and mobility together. However, this issue is important, since both may happen concurrently, thus affecting the transmission between nodes. Using our framework, we can naturally combine the independent AODV model with intended mobility models, as well as communication delay, for the above analysis purpose.

6.2 Preliminaries

Based on the previous specification for nodes, we combine an AODV node with each mobility-specific node by defining a subclass that inherits from both the AODV and the mobility model's classes. For example, an AODV node moving according to the random waypoint model is defined as:

```
class RWPANode .
subclasses RWPANode < RWPNode AODVNode .
```

In our experiments we consider the most crucial property of any routing protocol such as AODV, i.e., once a route discovery process starts, eventually there will be a route established between the source node and the destination node.

To check such a property, we use RTM's temporal logic model checker, and define an atomic proposition `route-found` to hold if we can find, in the routing table of the source node (`OIP`), a routing table entry towards the destination node (`DIP`):

```
op route-found : Oid Oid -> Prop [ctor] .
eq {CONFIG} |= route-found(OIP,DIP) = routeFound(OIP,DIP,CONFIG) .
```

where `routeFound` is defined by checking whether such a routing table entry exists in the source node's routing table:

```
op routeFound : Oid Oid Configuration -> Bool [frozen (3)] .
eq routeFound(OIP,DIP,< OIP : AODVNode |
              routingTable : (RT,DIP |-> TP) > C) = true .
eq routeFound(OIP,DIP,C) = false [owise] .
```

Thus, the property is defined using temporal logic as `<> route-found(...)`. Given an initial state `initConfig`, the following command returns `true`, if the property holds up to a test round `roundTime`; Otherwise, a trace illustrating the counterexample is shown.

```
(mc {initConfig} |=t <> route-found(oip,dip) in time <= roundTime .)
```

Moreover, if the property holds (or does not hold), we use the following timed search command to search for states reachable from a given initial state, and matched by a pattern satisfying the condition `routeFound`, which is defined by looking up a routing table entry towards the destination node in the source node's routing table.

```
(tsearch {initConfig} =>* {C:Configuration} such that
      routeFound(oip,dip,C:Configuration) in time <= roundTime .)
```

From the returned solutions, we can check, in the source node's routing table, the routing table entry towards the destination for the next hop, and further track its next hop towards the destination, so on and so forth, until we find the complete route.

6.3 Scenarios and Analysis

We make the following assumptions and setting for our experiments:

- The transmission range is `10m`, and the test area is `100m` × `100m`;
- The test round is `100s`. Both delays at a sender side and a receiver side are fixed after initialized, and independent of the distance between nodes. We set them to `10s` and `5s` respectively for all scenarios except for scenarios (iii) and (iii');
- The speed range is initialized as a singleton `(1)`. Nodes can move right, up, left or down. Thus, the direction range is correspondingly defined as a subset of `(0,90,180,270)`, and the destination range is a subset of four locations in the corresponding four directions based on a node's current location.

Scenario (i). This scenario, as shown in Fig. 2, considers the route discovery process with up to five stationary nodes, where the source node `1` located at `(45 . 45)` intends to build a route to the destination node `5` located at `(60 . 50)`, and nodes `2`, `3` and `4` are initially at `(50 . 50)`, `(50 . 40)` and `(60 . 40)` respectively. The model checking result shows that the property holds. By using the `tsearch` command, we can find a routing table entry (`5 |-> tuple3(2,2,1)`) in node `1`'s routing table, indicating that a 2-hop route with the next hop `2` is built towards node `5`. By further checking node `2`'s routing table, we can obtain the route (`1`→`2`→`5`).

Scenario (i'). This scenario, as shown in the left graph of Fig. 3 (in this paper, a solid circle refers to the initial location of a node, while a dash circle refers to some point along the motion path of a node), enjoys the same topology and setting with the above scenario, except that node `2` is a Random Waypoint node that can only move upwards. We set its pause time to: (a) `10s`, (b) `30s`, or (c) `60s`. Still the source node `1` intends to establish a route to the destination node `5`. As an example, the initial state of this scenario is specified as:

```
eq oip = 1 .
eq initConfig = < 1 : SANode | currentLocation : 45 . 45 , rreqID : 10,
                                sequenceNumber : 1, routingTable : empty,
                                requestBuffer : empty >
              < 2 : RWPANode | currentLocation : 50 . 50, speed : 0,
                                direction : 0, timer : pauseTime,
                                speedRange : (1), destRange : (50 . 60),
                                status : pausing, rreqID : 20,
                                sequenceNumber : 1, routingTable : empty,
                                requestBuffer : empty >
                 < 3 : SANode | currentLocation : 50 . 40, rreqID : 30,
                                sequenceNumber : 1, routingTable : empty,
                                requestBuffer : empty >
                 < 4 : SANode | currentLocation : 60 . 40, rreqID : 40,
                                sequenceNumber : 1, routingTable : empty,
                                requestBuffer : empty >
                 < 5 : SANode | currentLocation : 60 . 50, rreqID : 50,
                                sequenceNumber : 1, routingTable : empty,
                                requestBuffer : empty > (bootstrap oip) .
```

The experimental results show that:

- For Case (a), the property holds, and by searching we can find a routing table entry (`5 |-> tuple3(3,3,1)`) in node `1`'s routing table, indicating that a 3-hop route with the next hop `3` is built towards node `5`. By further tracking, we can obtain the route (`1`→`3`→`4`→`5`);
- For Case (b), the property does not hold, and the `tsearch` command returns no solution, meaning that there is no possibility that a route can be built between nodes `1` and `5`;
- For Case (c), the property holds, and we can obtain the same route with Scenario (i).

Analysis: For Case (a), in the search results, we also can find a routing table entry (`1 |-> tuple3(10,1,2)`) in node `2`'s routing table, meaning that node `2` has received the RREQ message from node `1`. This is obvious because the pause time (`10s`) equals to the sending delay (`10s`). However, when the receiving delay (`5s`) expires, node `2` has moved to `(50 . 55)`, as shown by the dash circle, that is beyond the transmission range of node `5`. Thus, we cannot obtain the same route with Scenario (i), but the route (`1`→`3`→`4`→`5`), because other nodes are stationary.

For Case (b), the pause time (`30s`) allows node `2` to forward the RREQ message to node `5`. However, node `2` cannot receive the RREP message from node `5` due to its movement (the dash circle in this case is at `(50 . 60)`). Meanwhile, since node `5` has already recorded node `1`'s RREQ from node `2`, it silently ingores the one from node `4`. Thus, in this case, no route can be established between nodes `1` and `5`.

For Case (c), the large pause time (`60s`) allows the same scenario with Scenario (i) before node `2` starts to move. Thus, a route (`1`→`2`→`5`) can be built.

Thus, our experimental results indicate that:

- Based on Scenario (i) and Case (a), AODV demonstrates robustness in route discovery in the face of some dynamic topologies;
- Further, based on Case (b), AODV's robustness in route discovery is limited;

- Based on Scenario (i) and Case (c), with large pause time, the network topology hardly changes, and thus provides a more stable environment to improve the success rate for the AODV route discovery;
- Based on the cases (a), (b) and (c), the performance of the AODV route discovery varies with different settings, even under the same mobility model.

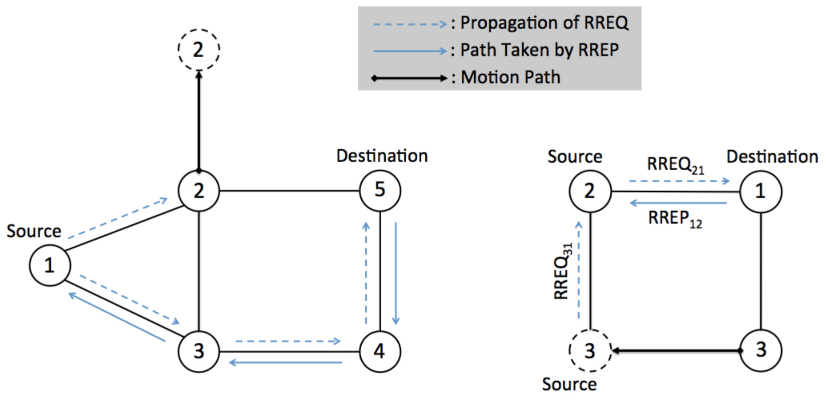

Fig. 3. Topologies of Scenarios (i') and (ii)

Scenario (ii). This scenario, as shown in the right graph of Fig. 3 considers three nodes with both nodes `2` located at `(40 . 50)` and `3` (a Random Waypoint node initialized with `timer : 0` and `status : pausing`) located at `(50 . 40)` intending to build a route to the destination node `1` located at `(50 . 50)`. Before sending out the RREQ message, node `3` moves left to a new location `(40 . 40)` within the transmission range of node `2`. Thus, to establish the route to node `1`, node `3`'s RREQ message needs to be forwarded by node `2`. However, the experimental results show that route discovery for node `3` fails, i.e., no route can be found between nodes `3` and `1`, though obviously node `2` succeeds in building a route to node `1`.

Analysis: This problem arises due to the discarding of the RREP message. As stated in [25], an intermediate node forwards a RREP message only if the RREP message serves to update its routing table entry towards the destination. However, in this case, node `2` has already secured an optimal route to node `1` before receiving the RREQ message from node `3`. [8] also pointed out this problem, but in a static linear topology with three nodes and two links. However, our scenario, besides uncovering that case, shows in a more realistic setting that node mobility may cause failure in the AODV route discovery.

Scenario (iii). This scenario, as shown in Fig. 4, considers four nodes with the source node `1` located at `(40 . 50)`, the destination node `4` located at `(70 .`

50), the intermediate stationary node 3 located at (60 . 50), and the intermediate Random Walk node 2 located at (50 . 40), which can move up or down for each step. Also, we set the delays at the sender side and the receiver side to 10s and 0s respectively, and node 2's moving time to 10s. Thus, it can receive the RREQ message from node 1, once it reaches up to the intermediate point (50 . 50) between nodes 1 and 3. However, the experimental results show that route discovery fails.

Scenario (iii'). This scenario enjoys the same topology and setting as Scenario (iii), except that node 2 is a Random Waypoint node with pause time 10s (initialized with `timer : 0` and `status : pausing`, so that it can receive the RREQ message from node 1, once it moves to the intermediate point (50 . 50) between nodes 1 and 3). The model checking results show that the property does not hold. However, the searching results illustrate that a route between nodes 1 and 4 can sometimes be successfully established.

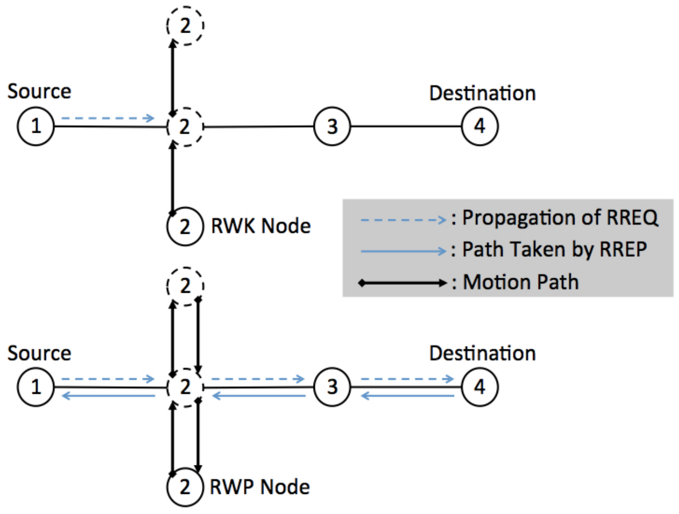

Fig. 4. Topologies of Scenarios (iii) and (iii')

Analysis: For Scenario (iii), when node 2 is ready to send out the RREQ message after both delays expire, it has moved to (50 . 60), indicated by the uppermost dash circle, which is beyond the transmission range of node 3, and therefore no RREQ message will be delivered to node 3, not to mention the destination node 4.

For Scenario (iii'), since node 2 has, for each moving step, two nondeterministic choices (up or down) for the intending destination, the property cannot be guaranteed for all possible motion paths, e.g., a path (up,down,down,down).

However, thanks to the Random Waypoint mobility model, AODV could be lucky to find a route. Specifically, node `2` pauses at the intermediate point `(50 . 50)` for a time interval that equals to the sum of both delays at its side, thus generating the RREQ message for node `3` before it moves away. Likewise, when node `2` returns to the intermediate point, it happens to receive the RREP message from node `3`. Despite of delays, it takes advantage of the pause time to forward the RREP message back to the source node `1`.

Thus, our experimental results indicate that, with mobility, AODV cannot guarantee the desired property of route discovery, but a route may be found in some cases. Also, performance of the AODV route discovery varies with mobility models in general, and Random Walk and Random Waypoint mobility models in particular, i.e., Random Waypoint nodes in some cases increase the chance of route discovery.

Table 1. Property Check Results (For `mc`, if the property holds, we mark with (✓), otherwise with (×). For `tsearch`, if a route can be found, we mark with (✓), otherwise with (×).)

Scenario / Command	(i)	(i')-(a)	(i')-(b)	(i')-(c)	(ii)	(iii)	(iii')
`mc`	✓	✓	×	✓	×	×	×
`tsearch`	✓	✓	×	✓	×	×	✓

6.4 Discussion

The above experiments demonstrate the feasibility and flexibility of our mobility framework, on the one hand, to formally analyze the AODV protocol under different mobility models with communication delays, and on the other hand, to compare the strength of such mobility models for a given protocol. Moreover, our framework is a step towards bridging the gap between protocol development and formal analysis, because it involves the same realistic mobility models with protocol simulation. Since our framework involves mobility, and does not handle a large number of nodes (due to state space limitations of normal model checking), the property in some scenario cannot be guaranteed. We simply compare with other approaches on this point:

- Protocol simulation allows a large number of nodes, so the property is more likely to be guaranteed. However, it cannot exhaustively check all possible scenarios, thus neglecting possible property violations;
- Other works using formal approaches did not consider realistic mobility, but static topologies, arbitrary node movement, or simple dynamic topologies with random link break, thus preventing a thorough reasoning about protocol properties.

In the future, we plan to improve the scalability of our framework by using probabilistic rewrite rules and exploiting statistical model checking methods and tools.

7 Related Work

There are a number of formal specification and analysis efforts of MANETs in general, and AODV in particular.

Bhargavan et al. [1] use the SPIN model checker to analyze AODV. They only consider a 3-node topology with one link break, but without node movement, and communication delay is not considered. Chiyangwa et al. [5] apply the real-time model checker Uppaal to analyze AODV. They only consider a static linear network topology. Although they take communication delay into account, the effect of mobility on communication delay is not considered, since the topology is fixed. Fehnker et al. [8] also use Uppaal to analyze AODV. They also only considered static topologies, or simple dynamic topologies by adding or removing a link, and those topologies are based on the connectivity graph without concrete locations for nodes. Furthermore, no timing issues are considered. Höfner et al. [13] apply statistical model checking to AODV. However, mobility is simply considered by arbitrary instantaneous node jumping between zones that split the whole test grid. Although they take into account the communication delay, the combination of mobility and communication delay is not considered. None of these studies has built a generic framework for MANETs. Our modeling framework aims at the combination of wireless communication and mobility, and allows formal modeling and analysis of protocols under realistic mobility models.

On the process algebra side, [20,22,11,18,26,10,12,19,7,17,15,16] have 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. Furthermore, [20] only considers fixed network topologies, whereas the others (except [12]) deal with arbitrary changes in topology. Godskesen et al. [12] consider realistic mobility, and propose concrete mobility models. However, no protocol application or automated analysis is given, and communication delay is not taken into account. Merro et al. [19] propose a timed calculus with time-consuming communications, and equip it with a formal semantics to analyze communication collisions.

Generally, these studies have proposed a framework for MANETs, but they lack of either mobility modeling or timing issues handling.

There are also a number of well known "ambient" calculi for mobility, such as the ambient calculus [4], the π-calculus [21], and the join-calculus [9]. However, these are very abstract models that do not take locations and geographically bounded communication into account, and are therefore not suitable to model MANETs at the level of abstraction considered in this paper.

Finally, Maude and Real-Time Maude have been applied to analyze wireless sensor networks, but the work in [24,14] do not consider node mobility (even though [14] mentions that mobility is addressed in a technical report in preparation; however, we cannot find that technical report).

8 Concluding Remarks

We have defined in Real-Time Maude what we believe is the first formal model of MANETs that provides a reasonably faithful model of popular node movement patterns and wireless communication. We have used our compositional model to specify and formally analyze the AODV routing protocol, and have shown that such Real-Time Maude analysis could easily find the known flaw in AODV.

We have abstracted from message collision, which should also be considered in our model. The price to pay for having a much more realistic model of MANETs than other formal approaches is that the state space quickly becomes too large for model checking. We should therefore develop statistical model checking techniques for MANETs. Most importantly, we should develop abstraction techniques for MANETs. The formalization presented in this paper has provided the necessary foundations for such efforts.

References

1. Bhargavan, K., Obradovic, D., Gunter, C.: Formal verification of standards for distance vector routing protocols. Journal of the ACM 49(4), 538–576 (2002)
2. Bouhoula, A., Jouannaud, J.P., Meseguer, J.: Specification and proof in membership equational logic. Theoretical Computer Science 236(1-2), 35–132 (2000)
3. 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)
4. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Proc. POPL'98. ACM (1998)
5. Chiyangwa, S., Kwiatkowska, M.Z.: A timing analysis of AODV. In: Proc. FMOODS'05. LNCS, vol. 3535. Springer (2005)
6. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude, LNCS, vol. 4350. Springer (2007)
7. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. In: Tech. Rep. 5513. NICTA (2012)
8. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: TACAS. pp. 173–187 (2012)
9. Fournet, C., Gonthier, G.: The reflexive cham and the join-calculus. In: Proc. POPL'96. ACM (1996)
10. Ghassemi, F., Fokkink, W., Movaghar, A.: Restricted broadcast process theory. In: Proc. SEFM '08. IEEE (2008)
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. Höfner, P., Kamali, M.: Quantitative analysis of AODV and its variants on dynamic topologies using statistical model checking. In: Proc. FORMATS'13. LNCS, vol. 8053. Springer (2013)
14. 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)
15. 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)
16. 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)
17. 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)
18. Merro, M.: An observational theory for mobile ad hoc networks (full version). Inf. Comput. 207(2), 194–208 (2009)
19. Merro, M., Ballardin, F., Sibilio, E.: A timed calculus for wireless systems. Theor. Comput. Sci. 412(47), 6585–6611 (2011)
20. Mezzetti, N., Sangiorgi, D.: Towards a calculus for wireless systems. Electron. Notes Theor. Comput. Sci. 158, 331–353 (2006)
21. Milner, R.: Communicating and mobile systems – the Pi-calculus. Cambridge University Press (1999)
22. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theor. Comput. Sci. 367(1), 203–227 (2006)
23. Ölveczky, P., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-order and Symbolic Computation 20(1-2), 161–196 (2007)
24. 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)
25. Perking, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (experimental) (2003), `http://www.ietf.org/rfc/rfc3561`
26. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A process calculus for mobile ad hoc networks. Sci. Comput. Program. 75(6), 440–469 (2010)
27. Su, P.: Delay measurement time synchronization for wireless sensor networks. Intel Research Berkeley Lab (2003)