Suitability of Abstract State Machines for Discussing Mobile Ad-hoc Networks

Alessandro Bianchi*, Sebastiano Pizzutilo and Gennaro Vessio

Department of Informatics, University of Bari, 70125 – Bari, Italy

Abstract: Several formalisms have been applied for addressing development issues in Mobile Ad-hoc NETworks (MANETs), however they usually lack of understandability, expressiveness and executability features. Instead, the Abstract State Machine (ASM) formalism does not suffer these limitations and can provide a useful conceptual tool for reasoning about MANET behavior. This paper shows the practical suitability of ASMs in capturing the specific MANET issues: concurrency, communications, and so on. To this end, the Ad-hoc On-demand Distance Vector (AODV) routing protocol for MANETs is modeled, and some properties of interest are proved.

Keywords: Abstract State Machines, MANET, AODV.

1. INTRODUCTION

In recent years, wireless technology has attracted great interest and considerable popularity. Its uses span in a wide range of application domains: ambient intelligence, real-time systems, transportation, video surveillance, and so on.

In this context, a Mobile Ad-hoc NETwork (MANET for short) is a wireless network designed for communications among nomadic hosts [1]; it does not need any fixed infrastructure, and communication sessions between initiator and destination are established and maintained by the cooperation of hosts in the network. MANETs are useful, sometimes necessary, for allowing hosts to communicate when fixed physical infrastructures cannot be used, for example for supporting rescue teams operating where pre-existing infrastructures are not reliable [2]. Hosts are intended as autonomous agents and they can dispose without according to a predefined topology; moreover, during their lifetime, they can enter or leave the network at will and continuously change their relative position.

The twofold role played by hosts (end-point and intermediate router), as well as the continuous change of the network topology due to movement, poses several problems: study of performance, analysis of synchronization issues, concurrency and cooperation, and so on. Among them, we focus on the need to define specific routing protocols for properly managing the lack of a fixed infrastructure. Since each host can directly communicate only within the area established by its transmission range, these protocols need to ensure broadcasting and unicasting of packets and to take into account the contribution of intermediate hosts for realizing communications. The Ad-hoc On-demand Distance Vector (AODV) [3] is one of the most relevant and widely used routing protocols for MANETs.

Traditionally, literature discusses routing protocols with the support of tools simulating the MANET behavior; for example [4] and [5]. The simulation-based approach is very effective from the execution point of view, mainly for evaluating performance and comparing different solutions. However, it can take into account only a limited, predictable range of different scenarios and it is not able to formally prove properties of interest. In other words, these tools are just simulators: they implement the network at a low abstraction level, but they cannot support specifications and analysis at higher level. Moreover, according to [6] and [7], the obtained results are sometimes inaccurate and poorly reliable.

On the contrary, formal methods can be useful for reasoning about MANET behavior and can provide more reliable results. In fact, representing a system-under-study at high level of abstraction allows developers to focus on algorithmic aspects, rather than on specific realizations of solutions at lower levels. Moreover, the mathematical foundation of formal methods provides complete and more accurate investigations about the properties and the behavior the system-under-study is required to exhibit.

Several formalisms have been successfully applied in the MANET domain: process calculi, finite state machines, Petri nets, and so on (for example, see [8], [9], [10]). However, these approaches present some drawbacks concerning: understandability, since they

*Address correspondence to this author at the Department of Informatics, University of Bari, 70125 – Bari, Italy; Tel: +39 080 544 2283;
Fax: +39 080 544 2031; E-mail: alessandro.bianchi@uniba.it

are based on a syntax dissimilar to traditional programming languages; expressiveness, because they typically provide only a single level of abstraction; executability, since, especially process calculi, are not directly executable.

With respect to these features, Abstract State Machines (ASMs) [11] provide advantages under several viewpoints. When the model expressivity is considered, literature agrees that ASM-based models show versatility in capturing both sequential and parallel computations at any desired level of abstraction, e.g. [12], [13]. Secondly, considering methodological issues, the ASM formalism has been successfully applied in both academia and industry for the design and the analysis of complex systems in several domains, and a specific development method got prominence in the last years [14]. Thirdly, considering the implementation point of view, the capability of translating formal models into executable ones is provided by tools like AsmL [15], CoreASM [16], and ASMETA [17]. Finally, the ASM approach provides a way to describe algorithmic issues in a simple abstract *pseudo-code*, which can be translated into a high level programming language source code in a quite simple manner.

The aim of this paper is to show the practical suitability of the ASM formalism in capturing the specific MANET features: mobility, concurrency, communications, structures for representing and storing information, and so on, and for reasoning about them. To this end, in this paper the AODV routing protocol for MANETs is modeled and some properties of interest are proved.

The rest of this paper is structured as follows. Section 2 is about related work. Section 3 provides a background knowledge on both ASMs and AODV. Our contribution is mainly reported in Section 4, which deals with the ASM-based model of AODV and reasons about its correctness. Section 5 concludes the paper.

2. RELATED WORK

Several process calculi specifically tailored for MANETs have been proposed in literature, for example: the ω-calculus [8], CMN (Calculus of Mobile Ad Hoc Networks) [18], and AWN (Algebra for Wireless Network) [19]. These formalisms naturally capture essential characteristics of network nodes: from the host mobility to the packet broadcasting and unicasting. In fact, the expressiveness of process calculi is very suitable for representing the parallel activities of the nodes and for abstracting the underlying communication mechanisms. However, these process calculi are not directly executable, so, executing models for conducting simulations is not possible. Moreover, they are typically based on mathematic notions that developers could find unfamiliar for their purposes. Instead, ASMs overcome both these issues by providing several tools for developing executable models and by offering a syntax very similar to traditional high level programming languages.

Various general purpose state-based models have also been used in the MANET domain, for example, finite state machines [9] and Petri nets [10]. With respect to process calculi, state-based models provide a comfortable way for representing algorithmic issues, especially from the graphical point of view: in fact, they are very useful for representing the execution flow along computational states. Moreover, they are typically equipped with tools, such as CPN Tools [20], that allow to execute models and conduct validation activities. Nevertheless, state-based models lack of expressiveness: basically, they provide only a single level of abstraction and cannot support refinements to executable source code. Conversely, ASMs are Turing-equivalent [11], so, they can represent systems at any desired level of abstraction till to models whose features are very close to implementation details.

An example of application of formal methods for specifically addressing issues in the AODV protocol can be find in [21], where a variant which improves the route selection of the original protocol is specified by means of Coloured Petri nets. However, the formalism is not used for proving correctness. Another example is [22], where a rigorous analysis of AODV and two variants is conducted by means of the previously mentioned AWN. These variants concern the problem of non-optimal route selection and that of failure in discovering routes. In this case, the formalism helps in proving correctness, nevertheless the focus is only on the *loop-freedom* property, i.e. the absence of loop routes.

Along the years, ASMs have been successfully used for modeling several systems, often concurrent and distributed, and for investigating their properties. For example, [23] and [24] show the application of ASMs for verifying several safety and liveness properties of two popular case studies: the Railroad Crossing and the Production Cell, respectively. More

recently, an ASM specification to model concurrency in a Web browser and to prove some consistency properties has been proposed in [25]; ASMs have also been used to model and validate vision-based robot control applications in [26]; and they have been applied for studying some aspects of Grid systems, e.g. reachability and reversibility, in [27]. These works attested how ASMs are sufficiently general for modeling systems in a very broad range of domains.

To our best knowledge, concerning the specific MANET domain, the ASM-based approach has been used only for modeling specific issues. In [28], the authors use ASMs for specifying location services and position-based routings among known locations, however they focus on the communication among the various architecture layers and do not formally prove properties of interest. Instead, in [29], a variant of the AODV protocol, which improves the network topology awareness of each node, is specified and formally proved by means of an ASM-based model, but the work does not emphasize the capability of the ASM formalism to face the MANET-specific modeling challenges.

3. BACKGROUND

3.1. Abstract State Machines

Informally speaking, ASMs are finite sets of so-called *rules* of the form **if** *condition* **then** *updates* (possibly with the **else** clause in addition) which transform the *abstract* states of the machine [14]. The concept of abstract state extends the usual notion of *state* occurring in finite state machines: it is an arbitrary complex structure, i.e. a domain of objects with functions and relations defined on them. On the other hand, the concept of rule reflects the notion of *transition* occurring in traditional transition systems: *condition* is a first-order formula whose interpretation can be true or false; while *updates* is a finite set of assignments of the form $f(t_1, \ldots, t_n) := t$, whose execution consists in changing in parallel the value of the specified functions to the indicated value.

According to [14], pairs of function names together with values for their arguments are called *locations*: they abstract the notion of memory unit. The current configuration of locations together with their values determines the current state of the ASM. In each state, all conditions are checked, so that all updates in rules whose conditions evaluate to *true* are simultaneously executed, and the result is a transition of the machine from a state to another, i.e. from a configuration of values in locations to another. Moreover, for the unambiguous determination of the next state, updates must be *consistent*, i.e. no pair of updates must refer to the same location.

The formalism also supports the mechanism of procedure calls; this is achieved by the definition of ASM *submachines*, i.e. parameterized rules, which allow the declaration of *local* functions, so that each call of a submachine works with its own instantiation of its local functions.

A generalization of basic ASMs is represented by Distributed ASMs (DASMs) [14], capable to capture the formalization of multiple agents acting in a distributed environment. Essentially, a DASM is intended as an arbitrary but finite number of independent agents, each executing its own underlying ASM. In a DASM the keyword **self** is used for supporting the relation between local and global states and for denoting the specific agent which is executing a rule.

Finally, the *ASM Method* defined in [14] encloses development phases from requirements capture to implementation in a unique ASM-based framework. Requirements can be captured by constructing so-called *ground models*, i.e. representations at high level of abstraction that can be graphically depicted; then, starting from ground models, a hierarchy of intermediate models is constructed by *stepwise refinements*, leading to executable code: each refinement describes the same system at a finer granularity. The method then allows both verification, through formal proof, and validation, through simulation.

3.2. Ad-hoc On-Demand Distance Vector Protocol

AODV is a *reactive* protocol that discovers and maintains routes on-demand, i.e. routes are built only as desired by initiator nodes using a route request/route reply cycle, which allows updating routing tables stored in each node [3]. When an initiator needs to start a communication session to a destination, and it does not know a proper route, it broadcasts a route request (RREQ) packet to all its neighbors. An RREQ packet, among the others, includes: *initiator address* and *broadcast id* (this pair uniquely identifies the packet); *destination address*; *destination sequence number*, which expresses the freshness of the information about destination; and *hop count*, initially set to 0, and increased by each intermediate node, for

expressing the distance. Because of broadcast transmissions, each intermediate node can receive several instances of a given RREQ from different neighbors: possible duplications of RREQs are discarded.

Knowledge of routes is stored into routing tables, recorded into a cache memory of each node. A routing table in a node lists all other nodes in the network, and the best (known) route to reach each of them. To this end, each entry of the routing table includes the address of the node, its sequence number, the hop count to reach it, and the *next hop* field identifying the next node in the route to reach it.

When a node receives an RREQ, it checks if one of the following holds: it is the destination, or destination is one of its neighbors, or it knows a route to destination with corresponding sequence number greater than or equal to the one contained in the RREQ (this means that the knowledge about the route is recent). If so, it unicasts a route reply (RREP) packet back to initiator; otherwise, it updates the hop count field, and rebroadcasts the RREQ. The process is so reiterated until a route to destination is found, or until a previously set timeout expires. An RREP packet contains: *initiator* and *destination address*, *destination sequence number*, and *hop count*. While RREP travels towards initiator, routes are set up inside the routing tables of the traversed hosts. Once initiator receives the RREP, communication starts.

The protocol also includes mechanisms for recording the up-to-date information about hosts. But, for the sake of simplicity, we do not consider this feature.

4. ASM-BASED MODEL OF AODV

A MANET that adopts the AODV routing protocol can be modeled by a DASM including a homogeneous set of *agents* = $\{a_1, \ldots, a_n\}$, where each agent models the behavior of a node executing the protocol (we can think that each a_i is univocally identified by its IP address).

Note that the intrinsic parallel execution of multiple ASMs, each of them executing sequential processes that, in turn, may be run in parallel, is a natural way for modeling the parallel composition of hosts in a MANET and their own computation.

Since all agents implement the same protocol, each agent behaves according to the same ASM, so only one ASM is discussed in the following. Each ASM can be in one of different computational states, which are characterized by the following predicates:

- *idle*: the agent is inactive. Its configuration is given by: *wishToInitiate*(**self**, *dest*) = *false*; *receivedRREQ*(**self**, *dest*) = *false*, with *dest* $\in$ *agents*, *isEmpty*(*replies*(**self**)) = *true*;
- *router*: the agent has received an RREQ. It is characterized by *receivedRREQ*(**self**, *dest*) = *true*;
- *initiator*: the agent has to start a new communication session. It is characterized by *wishToInitiate*(**self**, *dest*) = *true*;
- *forwarding*: the agent is forwarding an RREP to another destination. It is characterized by *isEmpty*(*replies*(**self**)) = *false*.

The functions dealing with the *idle*, *router* and *initiator* computational states are:

- *wishToInitiate*: *agents* × *agents* → *boolean*, which indicates whether a new communication session to *dest* is required by the environment;
- *receivedRREQ*: *agents* × *agents* → *boolean*, which acts as a flag indicating whether an RREQ packet has been received.

For modeling broadcasting and unicasting, each agent is associated with two types of queues of messages: *requests* and *replies*, which include RREQ and RREP packets, respectively. This allows us to model sending/receiving of packets by means of enqueuing/dequeuing abstract messages into the corresponding queue. These queues are managed by the function *isEmpty*, which states if a queue is empty or not, and by some specific ASM framework constructs: *enqueue*, *dequeue*, *empty*, and *top*, which adds an element to a queue, removes an element from a queue, removes all elements from a queue, and returns the top element of a queue, respectively. Each RREQ, RREP, or *record* (i.e. an entry of a routing table) is built by concatenation of the information described in Section 3.2 (in the pseudo-code the dot notation helps in identifying the value of a specific field of a packet).

When the MANET starts operating, each agent is idle, i.e. for each agent both *wishToInitiate*(**self**, *dest*) and *receivedRREQ*(**self**, *dest*) evaluate to *false* for

each *dest*, and *isEmpty*(*replies*(**self**)) evaluates to *true*. During the normal execution of an agent, for example, it can be in the *router* computational state with respect to a destination, but at the same time it can be in different computational states for other destinations: the value of the parameter *dest* is used for distinguishing these cases. Moreover, note that a *destination* computational state is not necessary: a host knows to be the destination when, in *router*, it receives an RREQ directed to it.

In addition, each ASM includes the following functions:

- *neighb*: *agents* → PowerSet(*agents*), which specifies the nodes in the neighborhood of each agent. Note that only the environment can set it;
- *routingTable*: *agents* → PowerSet(*records*), which represents the information about the nodes recorded into the agent's routing table.

It is worth noting that in general a host is characterized by more features, in particular, the amplitude of the area in which it is able to transmit, and the direction and the speed of its movement. In fact, due to their mobility, nodes can move in and out of a transmission range, and, as extreme case, they can leave the MANET space and become unreachable from any other node. However, for representing physical connection links, these features can be abstracted away: for each agent we can simply take into account its neighborhood.

On the other hand, the capability of ASMs to manipulate objects of arbitrary complexity, provides a useful way for expressing data structures which store entries of routing tables. Note that the *routingTable* function indirectly depends on the value of the *requests* and *replies* functions. In fact, whenever a node receives an RREQ or an RREP, it updates its routing table according to the content of the received packet. Moreover, in order to check if information about a host is stored into the agent's routing table, the function *hostInRT*: PowerSet(*records*) → PowerSet(*agents*) is defined: it returns the set of the agents stored in a given routing table.

The values of the *neighb* and *routingTable* functions, as well as the set *agents*, depend on the particular scenario: they are dynamically set according to the MANET evolution, with respect to both the host mobility and the computational history.

The ASM pseudo-code of the *i*-th agent is shown below:

```
AgentProgram(a_i) =
    if ¬(isEmpty(requests(self))) then {
        RREQ = top(requests(self))
        nextHop = sender of top(requests(self))
        update routingTable(self)
        receivedRREQ(self, dest) := true
        Router(RREQ, nextHop)
    }
    if wishToInitiate(self, dest) = true then
        Initiator(dest)
    if ¬(isEmpty(replies(self))) {
        RREP = top(replies(self))
        nextHop = select c.nextHop ∈
    hostInRT(routingTable(self)) with RREP.init = c.dest
        update routingTable(self)
        UnicastRREP(RREP, nextHop)
        dequeue RREQ from replies(self)
    }
UnicastRREP(RREP, a) =
    enqueue RREP into replies(a)
```

In the pseudo-code above, the instruction "update *routingTable*(**self**)" is not specified: it simply indicates that the agent's routing table is updated according to the received packet, i.e. an RREQ or an RREP.

Informally speaking, each agent is inactive until a new communication session is required by the environment; or until its computation is solicited by the receipt of an RREQ; or until it has to forward a unicast packet to the next hop in the route to reach the receiver of that packet. So, activation of an agent unfolds different computational branches: two of them lead to the execution of the *Router* or *Initiator* submachine, respectively; in the third case, forwarding computation of RREPs is executed. It is worth noting that all these activities evolve concurrently.

Finally, the *UnicastRREP* rule simply enqueues the RREP packet into the *replies* queue.

4.1. Router

Figure **1** shows the ground model of the ASM representing the behavior of router. According to the traditional notation of ASMs, circles represent computational states, diamonds represent conditions, and boxes represent updates. Moreover, note that this notation forces the usual syntax of flow charts because of the intrinsic parallel execution of ASM models: for example, ASM notation allows using diamonds with only one outgoing arc.

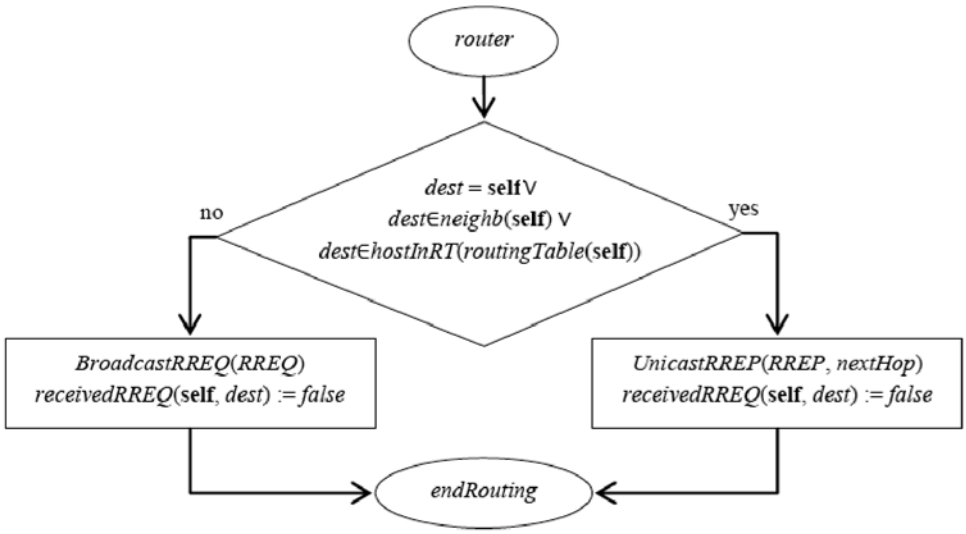

Figure 1: Ground model of router.

The corresponding pseudo-code is shown below:

```
Router(RREQ, nextHop) =
      dest = RREQ.dest
    if dest = self ∨ dest ∈ neighb(self ) ∨
    dest ∈ hostInRT(routingTable(self)) then {
        if ∃c ∈ routingTable(self) with c.dest = dest ∧
        c.destSeqNum ≥ RREQ.destSeqNum then {
            create RREP
            UnicastRREP(RREP, nextHop)
        }
        dequeue RREQ from requests(self)
        receivedRREQ(self, dest) := false
    }
    else {
        BroadcastRREQ(RREQ)
        dequeue RREQ from requests(self)
        receivedRREQ(self, dest) := false
    }
BroadcastRREQ(RREQ) =
    forall n ∈ neighb(self) do {
        forall r ∈ requests(n) do {
            if RREQ.dest = r.dest ∧ RREQ.id = r.id then
                discard RREQ
        }
        increase RREQ.hopCount
        enqueue RREQ into requests(n)
    }
```

Note that *UnicastRREP* behave as well as in the main program.

The *Router* submachine includes the *endRouting* computational state, which specifies that the execution of the routing activities due to the route discovery process is completed. This computational state is characterized by the value *false* for the *receivedRREQ*(**self**, *dest*) function. If router is the destination of the received RREQ (*dest* evaluates to **self**) or if it knows a fresh route to *dest* (*dest* ∈ *neighb*(**self**) ∨ *dest* ∈ *hostInRT*(*routingTable*(**self**))), then it unicasts an RREP packet back to initiator. Otherwise, it rebroadcasts the RREQ to all its neighbors. In both cases, the computation evolves to the *endRouting* computational state for that value of *dest*.

4.2. Initiator

Figure **2** shows the ground model of initiator. The corresponding pseudo-code is shown below:

```
Initiator(dest) =
    if dest ∈ neighb(self) ∨ dest ∈
hostInRT(routingTable(self)) then {
        StartCommunicationSession(dest)
        wishToInitiate(self, dest) := false
        sentRREQ(self) := false
    }
    else {
      create RREQ
        BroadcastRREQ(RREQ)
        sentRREQ(self) := true
    }
    if sentRREQ(self) ∧ ¬(isEmpty(replies(self))) then {
        select r ∈ replies(self) with maximum
    destSeqNum
        update routingTable(self)
        StartCommunicationSession(dest)
        empty replies(self)
        wishToInitiate(self, dest) := false
        sentRREQ(self) := false
    }
    if sentRREQ(self) ∧ isEmpty(replies(self)) ∧
    ¬(timeout(self) = 0) then
        timeout(self) := timeout(self) – 1
    if sentRREQ(self) ∧ isEmpty(replies(self)) ∧
    timeout(self) = 0 then {
        wishToInitiate(self, dest) := false
        sentRREQ(self) := false
    }
```

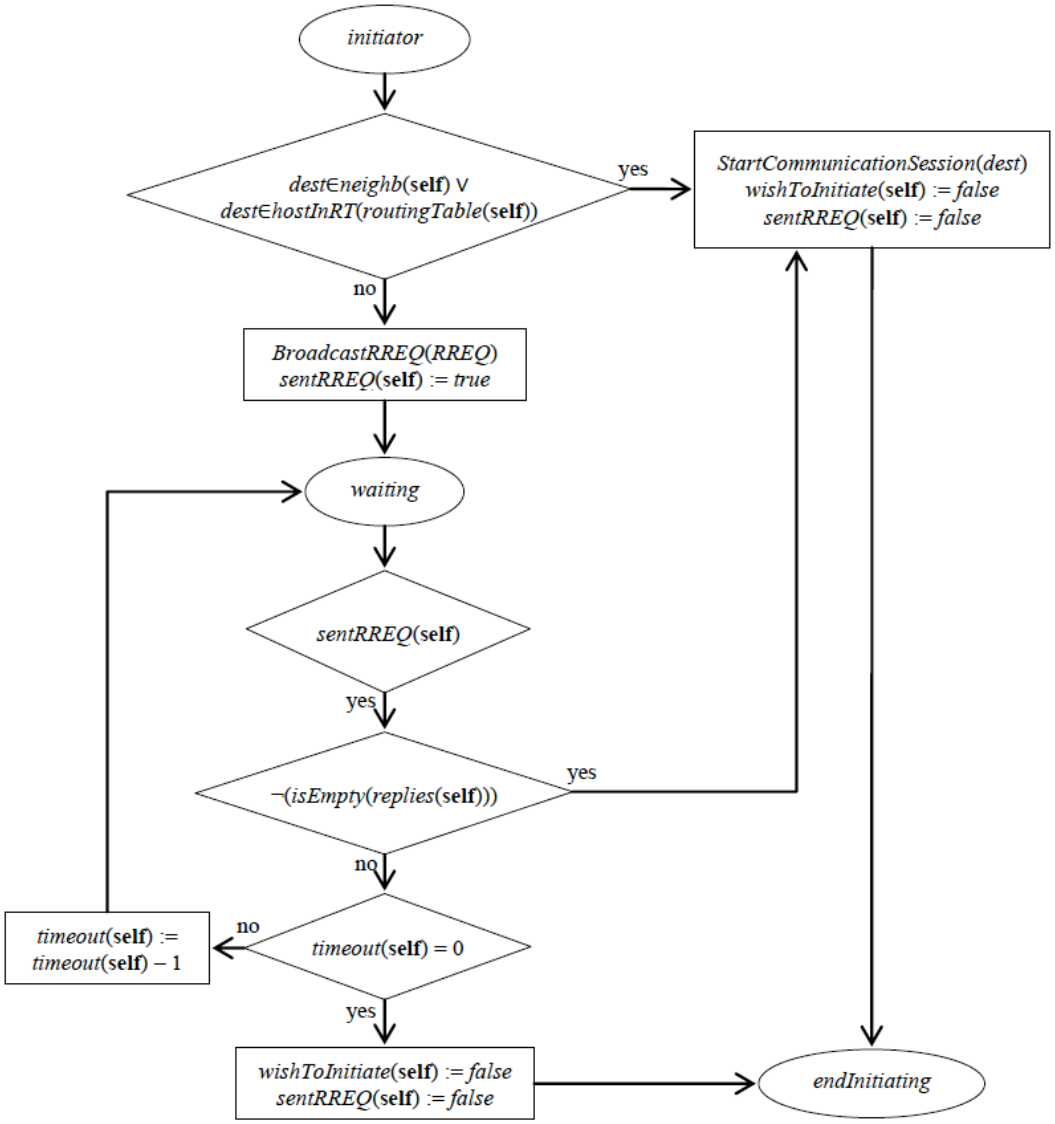

Figure 2: Ground model of initiator.

In the pseudo-code above: *BroadcastRREQ* behaves analogously to the *Router* submachine; instead, *StartCommunicationSession* is not described because it is not strictly part of the routing protocol.

The *Initiator* submachine is characterized by three local functions: *sentRREQ*: *agents* → *boolean*, which indicates whether an RREQ has been sent; *timeout*: *agents* → *integer*, which models the maximum waiting time for RREPs; and the aforementioned *replies*. This means that a new queue of *replies* is instantiated for each specific communication session. This submachine includes additional computational states, characterized by the following predicates:

- *waiting*: it indicates that the agent is waiting for responses concerning that *dest* from the other agents. Its configuration is given by: *wishToInitiate*(**self**, *dest*) = *true*; *sentRREQ*(**self**) = *true*; *isEmpty*(*replies*(**self**)) = *true*; *timeout*(**self**) > 0;
- *endInitiating*: it indicates that the computational activities executed by initiator, concerning the route discovery for that *dest*, are completed. Its configuration is: *wishToInitiate*(**self**, *dest*) = *false*; *sentRREQ*(**self**) = *false*.

If a route to *dest* is known, then the communication session simply starts; otherwise, *BroadcastRREQ* is executed. Its result consists in inserting a new RREQ into the *requests* queue of all the agent's neighbors and in evolving the current computational state to *waiting*. When an RREP is received (i.e.

isEmpty(*replies*(**self**)) evaluates to *false*), then the computation continues: the communication session starts and the *replies* queue is emptied; otherwise, when the timeout expires, the route discovery process ends.

Note that the model does not include the notification to the user about the timeout expiration. In order to acknowledge the impossibility to start the communication, the ASM can easily be refined, but, for the sake of brevity, we do not consider this feature.

4.3. Correctness

The DASM described so far behaves correctly, in the sense that it never starves and the expected packet is always received back by initiator. The proof of correctness is quite simple, so it is only sketched.

4.3.1. Starvation-Freedom

First of all, it is worth noting that the execution of each ASM does not starve. In fact, both the program of each agent, and the *Router* and *Initiator* submachines allow their execution always evolve: even when they must wait for external events, the waiting time is finite. For what concerns the overall machine, the branch dealing with the unicast forwarding of packets restores the *idle* computational state, so, the computation normally continues after this activity. The *Router* submachine returns back an RREP or rebroadcasts the previously received RREQ, depending on the values of the guard conditions, and then it evolves to the *endRouting* computational state. The *Initiator* submachine starts the communication session or enables nodes in neighborhood to execute the protocol. If the latter happens, then, if initiator receives back at least one RREP, it appropriately continues the execution by starting the communication session; otherwise, it stops waiting for the timeout expiration. In fact, if the computation loops within the *waiting* computational state, the value of the locations changes because of the update decreasing the timeout, which, under the assumption that it is greater than 0, will surely converge to 0. In this way, it is guaranteed that the state of the ASM can change, and the computation can evolve.

One more comment concerns the permanence in the *idle* computational state: it is not a case of starvation, but it simply indicates that the agent does not need to execute any activity. In ASM terms: the system evolution from *idle* only depends on the *wishToInitiate*(**self**, *dest*) and *isEmpty*(*requests*(**self**)) functions. If no one of them changes its value means that the agent is not proactive (it does not need to start a new communication) neither reactive (it has not received any RREQ).

4.3.2. Correctness of Received Packets

The ASM model shows that the correct packet is received back by initiator. In particular:

a. initiator receives back one or more RREPs if and only if a route to destination exists;

b. initiator does not receive back any packet if and only if no route to destination exists, or if it is isolated.

In order to prove issue (a) above, note that receiving RREPs means that the *replies* queue in the *Initiator* submachine is not empty, and this occurs if and only if the *UnicastRREP* rule has been executed: only this rule enqueues an RREP into *replies*. In turn, according to the guard conditions, *UnicastRREP* is executed if and only if that router or one of its neighbors is the destination, or a route to destination is recorded into its routing table. An analogous proof for issue (b).

Finally, it is easy to show that the execution of the protocol goes only through the described computational state: the rules execution transforms the values of the locations only into the desired way, so, no unexpected behavior can occur.

5. CONCLUSION

In this paper, the Ad-hoc On-demand Distance Vector routing protocol for Mobile Ad-hoc NETworks is modeled by means of an Abstract State Machine-based model. This case study allowed to show that the ASM approach is very useful for capturing the specific MANET features and for reasoning about them.

A MANET is characterized by a parallel composition of network nodes in which different sequential activities are, in turn, executed in parallel. The notion of run in a Distributed ASM, in which several ASMs are simultaneously executed, and the intrinsic parallel computation of ASM rules allow to express the nodes' execution in a very natural manner. Secondly, ASM functions, that can be defined over universes of objects of arbitrary complexity, can be used for representing nodes in neighborhood, so abstracting from physical features, and for recording the information about routes stored in routing tables. Thirdly, broadcasting and

unicasting of packets can be simply modeled by manipulating abstract messages and by inserting or deleting them into abstract queues. Finally, the similarity between the ASM pseudo-code and the syntax of traditional programming languages provides a way for describing algorithmic issues that developers can find familiar for modeling the system at high abstraction level and for investigating some properties of interest.

Furthermore, the ASM framework is equipped with several tools that allow to develop executable models and then conduct simulations. This issue is outside our purposes, so we do not consider it in this paper. However it is worth noting that the translation of the model we provided in Section 4 into the input required by an ASM tool, e.g. CoreASM, is only an exercise.

The advantages provided by the ASM formalism overcome the various drawbacks which affect other formalisms applied for addressing the specific MANET issues, such as process calculi, finite state machines and Petri nets. Typically, these formalisms lack of understandability and expressiveness features or of frameworks for executing models. On the contrary, ASMs do not suffer these limitations.

REFERENCES

[1] Agrawal DP, Zeng QA. Introduction to Wireless and Mobile Systems; Thomson Brooks/Cole 2003.

[2] Lien YN, Jang HC, Tsai TC. A MANET Based Emergency Communication and Information System for Catastrophic Natural Disasters. In: 29th International Conference on Distributed Computing Systems Workshops 2009; 412-417.

[3] Perkins CE, Belding-Royer EM, Das SR. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 2003. http://tools.ietf.org/html/rfc3561.

[4] Jayakumar G, Gopinath G. Performance Comparison of Two On-demand Routing Protocols for Ad-hoc Networks Based on Random Way Point Mobility Model. American Journal of Applied Sciences 2008; 5(6): 659-664. http://dx.doi.org/10.3844/ajassp.2008.659.664

[5] Goyal P. Simulation Study of Comparative Performance of AODV, OLSR, FSR and LAR Routing Protocols in MANET in Large Scale Scenarios. In: World Congress of Information and Communication Technologies 2012; 283-286.

[6] Cavin D, Sasson Y, Schiper A. On the Accuracy of MANET Simulators. In: ACM Workshop on Princples of Mobile Computing 2002; 38-43.

[7] Kurkowski S, Camp T, Colagrosso M. MANET Simulation Studies: The Incredibles. ACM SIGMOBILE Mobile Computing and Communications Review 2005; 9(4): 50-61. http://dx.doi.org/10.1145/1096166.1096174

[8] Singh A, Ramakrishnan C, Smolka S. A Process Calculus for Mobile Ad Hoc Networks. In: 10th International Conference on Coordination Models and Languages 2008; 296-314. http://dx.doi.org/10.1007/978-3-540-68265-3_19

[9] Delzanno G, Sangnier A, Zavattaro G. Parameterized Verification of Ad Hoc Networks. In: 21th Int. Conf. of Concurrency Theory 2010; 313-327.

[10] Bianchi A, Pizzutilo S. Studying MANET through a Petri Net-Based Model. In: 2th International Conference of Evolving Internet 2010; 220-225. http://dx.doi.org/10.1109/INTERNET.2010.44

[11] Gurevich Y. Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic 2000; 1(1): 77-111. http://dx.doi.org/10.1145/343369.343384

[12] Börger E, Riccobene E. Logic+ Control Revisited: An Abstract Interpreter for Gödel Programs. In: Levi G. (ed.), Advances in Logic Programming Theory 1994; 154-231.

[13] Németh Z. Denition of a Parallel Execution Model with Abstract State Machines. Acta Cybernetica 2002; 15(3): 417-455.

[14] Börger E, Stärk R. Abstract State Machines: A Method for High-Level System Design and Analysis 2003; Springer-Verlag. http://dx.doi.org/10.1007/978-3-642-18216-7

[15] Gurevich Y, Rossman B, Schulte W. Semantic Essence of AsmL. Theoretical Computer Science 2005; 342(3): 370-412. http://dx.doi.org/10.1016/j.tcs.2005.06.017

[16] Farahbod R, Gervasi V, Glässer U. CoreASM: An Extensible ASM Execution Engine. Fundamenta Informaticae 2007; 77(1-2): 71-103.

[17] Gargantini A, Riccobene E, Scandurra P. Model-Driven Language Engineering: The ASMETA Case Study. In: 3rd International Conference on Software Engineering Advances 2008; 373-378.

[18] Merro M. An Observational Theory for Mobile Ad Hoc Networks. Information and Computation 2009; 207(2): 194-208. http://dx.doi.org/10.1016/j.ic.2007.11.010

[19] Fehnker A, Glabbeek RV, Höfner P, McIver A, Portmann M, Tan WL. A Process Algebra for Wireless Mesh Networks. In: 21st European Symposium on Programming, 2012; 295-315.

[20] Jensen K, Kristensen LM, Wells L. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer 2007; 9(3-4): 213-254. http://dx.doi.org/10.1007/s10009-007-0038-x

[21] Nakhaee A, Harounabadi A, Mirabedini J. A Novel Communication Model to Improve AODV Protocol Routing Reliability. In: 5th International Conference on Application of Information and Communication Technologies 2011; 1-7.

[22] Höfner P, van Glabbeek RJ, Tan WL, Portmann M, McIver A, Fehnker A. A Rigorous Analysis of AODV and its Variants. In: 15th ACM Int. Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems 2012; 203-212.

[23] Gurevich Y, Huggins J. The Railroad Crossing Problem: An Experiment with Instantaneous Actions and Immediate Reactions. In: 9th International Workshop on Computer Science Logic 1996; 266-290. http://dx.doi.org/10.1007/3-540-61377-3_43

[24] Börger E, Mearelli L. Integrating ASMs into the Software Development Life Cycle. Journal of Universal Computer Science 1997; 3(5): 603-665.

[25] Gervasi V. An ASM Model of Concurrency in a Web Browser. In: 3rd International Conference on Abstract State Machines. Alloy. B VDM and Z 2012; 79-93.

[26] Luzzana A, Rossetti M, Righettini P, Scandurra P. Modeling Synchronization/Communication Patterns in Vision-Based Robot Control Applications Using ASMs. In: 3rd International Conference on Abstract State Machines, Alloy, B, VDM, and Z 2012; 331-335.

[27] Bianchi A, Manelli L, Pizzutilo S. An ASM-based Model for Grid Job Management. Informatica (Slovenia) 2013; 37(3): 295-306.

[28] Benczur A, Glässer U, Lukovskzi T. Formal Description of a Distributed Location Service for Mobile Ad-hoc Networks. In: Börger, E., Gargantini, A., Riccobene, E. (eds), Abstract State Machines 2003 – Advances in Theory and Applications 2003; 2589: 204-217.

[29] Bianchi A, Pizzutilo S, Vessio G. Preliminary Description of NACK-based Ad-hoc On-demand Distance Vector Routing Protocol for MANETs. In: 9th Int. Conference on Software Engineering and Applications 2014; 500-505. http://dx.doi.org/10.5220/0005105305000505

Received on 20-10-2014 Accepted on 10-11-2014 Published on 28-11-2014