Chapter 5

A Java Application for Modelling and Simulating Mobile Ad-hoc NETworks

Emanuele Covino and Giovanni Pani

5.1. Introduction

Mobile Ad-hoc NETwork (MANET) is a technology used to model wireless communication among hosts in absence of physical infrastructure [1]. In a MANET, hosts are autonomous agents: they can enter or leave the network, and they can change their relative position. This implies that these networks lack a predefined topology. Each host can communicate with the others inside a predefined range only; communication outside this area is possible only by means of cooperation between intermediate hosts. They can act as initiator, intermediate and destination of a communication, following a predefined protocol.

This technology may be used in a number of sensitive applications, where malfunctioning or not adequate performance could result in severe damage to people, environment, or other systems; for instance, rescue operations in case of disasters, data tracking of environmental conditions, health-care, intelligent transportation, and environmental emergency management. This raises several problems about the analysis of performance, synchronization and concurrency of the network, and it is important to be able to verify qualities like responsiveness, robustness, correctness and performance, starting from the early stages of the development. This research area is receiving special attention in the last few years, in the context of smart mobile computing, cloud computing, Cyber Physical Systems and Internet of Things ([2] and [3]).

E. Covino (emanuele.covino@uniba.it), G. Pani (giovanni.pani@uniba.it)
Dipartimento di Informatica, Università di Bari, Italy

In general, the analysis and the evaluation of MANET's properties can be done by means of simulators (focusing on performance's metrics), or by means of formal models of the system (studying computational properties). For instance, [3-7] compare some routing protocols performances; [8-10] study congestion adaptive routing; [11, 12] discuss about managing synchronization among components involved in simulation; [13] evaluates a topology control approach. Nevertheless, some Authors show that the results obtained using simulators can be inaccurate or unreliable [9, 10, 14, 15]. Simulators are suitable to evaluate and compare performances, but they don't provide a formal model of the MANETs. The network is implemented at a lower level, while a higher abstraction level of specification is needed in order to study problems such as concurrency, synchronization, or deadlock. On the other hand, some examples of the application of formal methods to the analysis of MANETs have been proposed, such as process calculi [12], CMN (Calculus of Mobile Ad Hoc Networks) [15], and AWN (Algebra for Wireless Networks) [10]. They capture some essential characteristics of nodes, such as mobility or packet's broadcasting and unicasting.

Another class of formal methods used for studying MANETs is represented by state-based models, such as Finite State Machines [16] and Petri nets. The latter have been employed to study modeling and verification of routing protocols [17], evaluation of protocol performance [18], and application to vehicular networks [19]. With respect to process calculi, they provide a more suitable way of representing algorithms, and they are typically equipped with tools, such as CPN Tools [20], that allow to simulate the algorithms, directly. However, state-based models lack expressiveness: basically, they provide only a single level of abstraction, and cannot support refinements to executable code.

In this chapter, we introduce MOTION (MOdeling and simulaTIng mObile ad-hoc Networks), a Java application in which the behavior of MANETs is modeled by means of an Abstract State Machine representation [21], and then simulated with the simulation engine ASMETA [22]. This approach is similar to [17], in which Colored Petri Nets are used to model the AODV routing protocol (Ad-hoc On-demand Distance Vector), and CPN model is used to simulates the MANET behavior. As an improvement, our approach is more general purpose, meaning that the implementation of the routing protocol is only one of the several services that can be modeled in our layered framework and

implemented in the simulator. Thanks to the structured approach, services can be easily added, removed and replaced by changing some transitions and nested nets, as well as changing classes in software implementation. The ASM approach also provides a way to describe algorithms in a simple abstract pseudo-code, which can be translated into a high-level programming language source code [21]. Finally, from the implementation point of view, the capability of translating formal specifications into executable code, in order to carry out simulations of the models, is provided by tools like CoreASM [23] and ASMETA [22]. MOTION can be used to prove properties of the network, formally, as well as it can simulate its behavior. We provide a detailed description and a platform-independent version of the MOTION environment; the initial interface of the application and the dialogue with AsmetaS (i.e., ASMETA's Simulator) are coded entirely in Java, in order to ensure compatibility with the main Operating Systems.

5.2. Models for Routing Protocols

In this section, we recall the basic concepts related to the routing protocols used within MOTION, to the Abstract State Machines formalism, and to the ASMETA framework.

5.2.1. MANET and Routing Protocols

We have already introduced the Mobile Ad-hoc NETworks; they are wireless communication systems in which each host is an autonomous agent that can rearrange its position with respect to the other hosts. This means that routes connecting the hosts can rapidly change. Several routing protocols have been proposed to handle this kind of networks; among them, the Ad-hoc On-demand Distance Vector (AODV, [24]) is one of the most popular, with many simulation studies dealing with it. For this reason, it is a reliable baseline when comparing its simulations' results to those obtained with MOTION. We add two variants of AODV: NACK-based Ad-hoc On-demand Distance Vector (N-AODV, [25]), that improves the awareness that each host has about the network topology, and Blackhole-free N-AODV (BN-AODV, [26]), that detects the presence of malicious hosts leading to a blackhole attack.

Ad-hoc On-demand Distance Vector (AODV). This protocol combines two mechanisms, the *route discovery* and the *route*

maintenance, in order to store into *routing tables* some knowledge about the routes. Each node maintains its routing table, that is a list of the routes towards other nodes that have been discovered and are still valid. In particular, an entry of the routing table of the node *i* concerning a node *j* includes: the *address* of *j*; the last known *sequence number* of *j*; the *hop count* field (expressing the distance between *i* and *j*); and the *next hop* field (identifying the next node in the route to reach *j*). The sequence number is an increasing number maintained by each node, that express the freshness of the information about the respective node. When an *initiator* wants to start a communication session towards a *destination*, it checks if a route is currently stored in its routing table. If so, the communication can start. Otherwise, the initiator broadcasts a control packet called *route request* (RREQ) to all its neighbors. An RREQ packet includes the initiator address and broadcast id, the destination address, the sequence number of the destination (i.e., the latest available information about destination), and the hop count, initially set to 0, and increased by each intermediate node. The pair *<initiator address; broadcast id>* identifies the packet; this implies that duplications of RREQs already handled by nodes can be ignored.

When an intermediate node *n* receives an RREQ, it creates the routing table entry for the initiator, or updates it in the fields related to the sequence number and to the next hop. Then, the process is iterated: *n* checks if it knows a route to destination with corresponding sequence number greater than (or equal to) the one contained into the RREQ (this means that its knowledge about the route is more recent). If so, *n* unicasts a second control packet (the *route reply* - RREP) back to the initiator. Otherwise, *n* updates the hop count field and broadcasts once more the RREQ to all its neighbors.

The process successfully ends when a route to the destination is found. While the RREP travels back to the initiator, routes are set up inside the routing tables of the traversed nodes, creating an entry for destination, when needed. Once the initiator receives back the RREP, the communication session can start. If the nodes' movements break a link (i.e., a logical link stored in a routing table is no more available), a route maintenance is executed in order to notify the error and to invalidate the corresponding routes: to this end the control packet *route error* (RERR) is used.

NACK-based AODV (N-AODV). With the AODV protocol, the nodes have a limited knowledge about the network topology. Each node *n* is

aware of the existence of a node *m* only when *n* receives an RREQ, either originated by, or directed to *m*. The NACK-based AODV routing protocol has been proposed and modeled by means of a Distributed ASM in [25], in order to improve this awareness.

This protocol adds a *Not ACKnowledgment* (NACK) control packet in the route discovery phase. Whenever an RREQ originated by *n* and directed to *m* is received by the node *p* that doesn't have any information about *m*, *p* itself unicasts the NACK to *n*. In this way, *n* and all the nodes in the path to *p* receive fresh information about the existence and the position of *p*, and they add an entry in their respective routing tables, or they update the pre-existing entry. N-AODV has been experimentally validated through simulations, showing its efficiency: the nodes in the network improve their knowledge about the other nodes and, in the long run, the number of RREQ decreases, with respect to the AODV protocol.

Black hole-free N-AODV (BN-AODV). In general, routing protocols assume the trustworthiness of each node; this implies that MANETS are very prone to the *black hole attack* [27]. In AODV and N-AODV a black hole node produces fakes RREPs, in which the sequence number is as great as possible, so that the initiator sends the message packets to the malicious node, and the latter can misuse or discard them. The black hole can be supported by one or more *colluders*, that confirm the trustworthiness of the fake RREP. The Black hole-free N-AODV protocol [26] allows the honest nodes to intercept the black holes and the colluders, thanks to two control packets: each intermediate node *n* receiving an RREP must verify the trustworthiness of the nodes in the path followed by the RREP; to do this, *n* produces a *challenge packet* (CHL) for the destination node, and only the latter can produce the correct *response packet* (RES). If *n* receives RES, it sends the RREP, otherwise the next node towards the destination is considered as a possible black hole.

5.2.2. Abstract State Machines and ASMETA

An Abstract State Machine (ASM [21]) M is a tuple (Σ, S, R, P_M). Σ is a *signature*, that is a finite collection of names of total functions; each function has -arity *n*, and the special value *undef* belongs to the range. Relations are functions that always evaluate to *true*, *false* or *undef*.

S is a finite set of *abstract states*. The concept of abstract state extends the usual notion of state occurring in finite state machines: it is an algebra over the signature Σ, i.e. a non-empty set of objects of arbitrary complexity, together with interpretations of the functions in Σ.

R is a finite set of *rules* of the form "*if condition then updates*", which transform the states of the machine. 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; *updates* is a finite set of assignments of the form $f(t_1; t_2; \ldots t_n) := t$, whose execution changes in parallel the value of the specified functions to the indicated value.

P_M is the *main rule* of the machine M, of -arity 0, which is the starting point of the computation.

Pairs of function names together with values for their arguments are called *locations*: they are the abstraction of the notion of memory unit. Since a state can be viewed as a function that maps locations to their values, the current configuration of locations, together with their values, determines the current state of the ASM.

In order to clarify the semantics of the states with respect to the computational behavior of the system, we underline that each ASM state can be characterized by one or more predicates over the states. More precisely, a predicate *H* over an ASM state *s* is a first-order formula defined over the locations in *s*, such that $s \mid = H$. Each predicate allows us to focus on the subsets of locations that turn out to be interesting for verification purposes.

The execution of an ASM is made of computational steps. Given a state *s*, a *computational step* in *s* consists in executing all the rules whose condition is true in that state. Since different updates could affect the same location, it is necessary to impose a consistency requirement: a set of updates is said to be *consistent* if it contains no pairs of updates referring to the same location. Therefore, if the updates are consistent, the result of a computational step is the transition of the machine from the current state to another. Otherwise, the computation doesn't produce a next state. A *run* is a (possibly infinite) sequence of steps: the computational step is iterated until no more rules are applicable.

The previous notions refer to the so-called *basic* ASMs. However, there exist some generalizations, namely the *parallel* ASMs and *distributed* ASMs (DASMs) [28]. Parallel ASMs are basic ASMs enriched with the *forall* construct, to express the simultaneous execution of the same ASM (i.e., of rules satisfying a given condition) over many independent agents. A distributed ASM is intended as a finite number of independent agents, each one executing its own underlying ASM: this model formalizes the behaviour of multiple agents acting in a distributed environment. A run of a DASM is a partially ordered set of the runs of its ASMs: the underlying synchronization scheme reflects causal dependencies; determining which agent's move comes before is a single computational step of an individual agent, and is only restricted by the consistency condition, which is mandatory. Roughly speaking, a global state corresponds to the union of the signatures of each ASM, together with the interpretations of their functions.

The ASM-based method consists in development phases, from requirements' specification to implementation, supporting developers in designing complex systems. Some environments support this method, and among them we use the ASMETA (ASM mETAmodeling) framework [5, 29]. This framework is characterized by logical components that capture the requirements by constructing the so-called *ground models*, i.e. representations at high level of abstraction that can be graphically depicted. Starting from ground models, hierarchies of intermediate models can be built, leading to executable code: each refinement describes the same system at a finer granularity. The framework supports verification, through formal proof, and validation, through simulation.

5.3. Defining MOTION

MOTION (MOdeling and simulaTIng mObile ad-hoc Networks) is a Java application by which the simulation parameters of a network are specified, the network is executed, and the simulation's output data are collected. The related web pages can be found at https://sourceforge.net/projects/motion-project/. MOTION is developed within the ASMETA framework, using the abstract syntax defined in the Abstract State Machines Metamodel (AsmM). This is the description of a language for ASMs, representing domains, functions, axioms, rules; the syntactic constructs occurring in the ASM's states; the syntactic elements enabling the transition rules, and so on. The MANET is

modelled using the ASMETA Language (AsmetaL), and it is executed by the ASMETA Simulator (AsmetaS). Since the latter simulates instances of the model expressed by means of the AsmetaL, the information concerning each instance, such as the number of agents and their features, must be recorded into the AsmetaL file.

The executions of MOTION and ASMETA are interleaved. MOTION provides the user interface and accepts the parameters of the simulation; then, it includes these data into the AsmetaL file, and it runs AsmetaS. AsmetaS executes an ASM move, simulating the behavior of the network protocol, then it records the values of the locations in a log file, for each state. The control is returned to MOTION, that gets the information about the results of the move (such as, the relative position of the hosts, the sent/received packets, and the values of waiting time) and records them into the AsmetaL file. Then, MOTION calls AsmetaS for the next move. At the end of the simulation session, MOTION stores the contents of the log file into a csv file.

5.3.1. The Mobility Model

In a realistic scenario, the hosts of a MANET follow the rules of a routing protocol, and they play two different roles. On one hand, they are communication agents, acting as initiators, destinations, or as intermediate hosts of a communication. At the same time, they move into the MANET space, breaking and creating new links. Because of the wireless nature of MANET, each host is associated with a radio range, which specifies the maximum distance the signal sent by a host can be received by another host. Amplitude of the radio range and movement of the hosts determine the topology of the network.

A realistic simulation should consider all these features, but the simulation of all aspects of a MANET can be cumbersome, and sometimes impossible; according to [30], the model of the systems to be simulated must be tailored depending on the goals of the simulation project itself. Therefore, the movement issues and the amplitude of the radio range are abstractly defined within the mobility model. In this sense, we assume that the whole network topology is expressed by the connections among nodes, implicitly, and for each node we consider only its current neighborhood. More precisely, in MOTION the network topology is expressed by a *connectivity matrix C*, such that $c_{ij} = 1$ if i and j are neighbors; 0 otherwise, for each pair of nodes i and j. Within the ASM model, C is expressed by the predicate $isLinked(a_1;a_2)$, which

evaluates to *true* when a_1 is linked to a_2; to *false* otherwise. Changes of *isLinked* represent the transitions of each node from one set of neighbors to another.

The mobility model is implemented into a Java class that, before executing any ASM move, updates the connectivity matrix. In order to do this, each c_{ij} is set to 0 or 1 randomly, according to a parameter defined by the user. The new values of the connectivity matrix are then stored into the AsmetaL file, so that the ASM move can be executed, accordingly.

5.3.2. Models Based on Abstract State Machine

The AODV routing protocol has been formally modelled through ASMs in [31]. It is defined as a collection of agents, each one representing a node. The high-level machine in MOTION is:

```
MAIN RULE =
        forall a ∈ Agents do AODVSPEC(a)
where
AODVSPEC(a) =
        forall dest ∈ Agents with dest ≠ self do
                if WaitingForRouteTo(self, dest) then
                        if Timeout(self, dest) > 0 then
                                Timeout(self, dest): = Timeout(self, dest)-1
                        else
                                WaitingForRouteTo(self, dest): = false
        if WishToInitiate(self) then PREPARECOMM
        if not Empty(Message) then ROUTER
```

If a node has to start a communication, the function *WishToInitiate* evaluates to true, and the PREPARECOMM submachine is called. The function *WaitingForRouteTo* evaluates to *true* if the discovery process previously started is still running; in this case, if the waiting time for RREP is not expired (*Timeout*( ) > 0), the time-counter is decreased. Finally, if the node has received a message (either RREQ, RREP, or RERR), the ROUTER submachine is called:

```
ROUTER =
```

ProcessRouteReq
ProcessRouteRep
ProcessRouteErr

where each submachine expresses the behavior of the node, that depends on the type of the message received.

The ASM model for N-AODV is similar: the main difference concerns ROUTER, that includes a submachine PROCESS-NACK, in order to unicast the NACK packet, if needed. The BN-AODV model is more structured, because it describes the behavior of three different types of agents: honest, black holes, and colluders. Thus, the main rule has the form:

MAIN RULE =
forall a ∈ Blackhole **do** BLACKHOLESPEC(a)
forall a ∈ Colluder **do** COLLUDERSPEC(a)
forall a ∈ Honest **do** HONESTSPEC(a)

HONESTSPEC submachine describes the behavior of the honest nodes, and it's analogous to AODVSPEC. BLACKHOLESPEC and COLLUDERSPEC are the specifications for the non-honest nodes and the colluders, respectively. Moreover, the ROUTER submachine for the honest nodes includes a submachine that verifies the trustworthiness of the RREP's.

5.3.3. Specific Behavior of MOTION

A simulation in MOTION is performed in sessions, whose number is established by the user. Hosts included in each session depend on the specific evolution of the network (due to movements, some of them can be disconnected, meaning that they cannot be reached by the other hosts). Moreover, during each session, each host is the initiator for some attempts to establish a communication towards a destination different from the initiator itself: the user expresses the probability that each host will act as an initiator by setting the value of the parameter *Initiator Probability* (in Fig. 5.1, the value is 10%). For each communication attempt (in what follows, CA), both initiator and destination are randomly defined. Thanks to the intrinsic parallelism in the execution of the ASM's rules, more attempts can be simultaneously executed. A CA is considered successful if the initiator receives an RREP packet within the waiting time expressed by the parameter *RREP Timeout*; otherwise,

the attempt is considered failed. The elapsed time is measured as the number of times the main rule of the ASM has been executed.

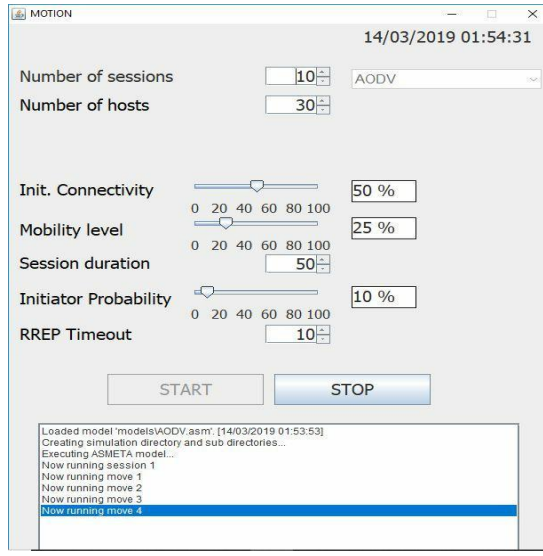

Fig. 5.1. MOTION user interface.

The hosts mobility is defined by the user by means of two parameters, the *Initial Connectivity* and the *Mobility level*. The former defines the initial topology of the MANET: it is the probability that each host is directly linked to any other host. During the simulation, for each pair of hosts $<a_i;a_j>$, and for each move of the ASM, the hosts mobility is expressed by changing the value of $isLinked(a_i;a_j)$ with a probability expressed by *Mobility level*.

When the BN-AODV protocol is simulated, the user interface includes the definition of the number of black holes and colluders, together with two parameters establishing the increment of the fake sequence number produced by the black hole.

In Fig. 5.1, the current state of the simulation can be found in the window under the two buttons START and STOP. From the ASM perspective, there are two different machines, both called by the ASMETA's main rule. First, OBSERVERPROGRAM is used to manage the execution. It initializes the locations and data structures for all the hosts, manages the

mobility (setting the initial topology and resetting the connectivity matrix at each move), and updates the counter for the time expiration. The second machine, called by the main rule, is the model of the hosts' behavior. MOTION allows the users to study AODV, N-AODV, and BN-AODV, specified according to the ASMs presented in [31, 25, 26], respectively. Note that, for all of them, the MANET is modeled by a Distributed ASM. In both AODV and N-AODV all the nodes behave in the same way, described by the respective DASM, so the machine specifying the protocol is called; at each move the machine randomly decides if the current node will initiate new communication attempts by invoking the R-PREPARECOMM submachine, then it acts as a router by processing the proper control packets (R-ROUTER submachine).

5.4. Experiments with MOTION

In this section, we show the results of some simulations made with MOTION, in order to evaluate the performances of the AODV and the N-AODV protocols, as well as to test the usability of the tool. The results have been compared to those already discussed in literature, with the exception of studies about BN-AODV, that are not available. The first analysis compares performances measured by MOTION to those obtained with other simulators. The second one deepens into the relationships existing among some simulation parameters.

Each simulation is performed on a specific number of hosts in the MANET: 10, 20, and 30 hosts, respectively. For each population, three different values of the Mobility level parameter are taken into consideration: 25, 50, and 75 %, respectively. This leads to nine different simulation, and all the remaining parameters are left unchanged. Each simulation includes ten sessions, each of which lasting 50 ASM moves; the initial connectivity value is 50 %; each host is an initiator of a CA with a probability of 10 %; a CA is successful if the packet RREP is received by the initiator within 10 ASM moves. The following metrics have been defined and collected, for each simulation:

M1 the *rate of success*, that is the ratio between successful and overall number of CA's;

M2 the *control overhead*, that is the total number of control packets produced for each CA (i.e., RREQs, RREPs, and REERs for both protocols);

M3 the *RERR amount*, that is the total number of RERR packets produced as a result of a link breakage;

M4 the *RREQ percentage*, that is the percentage of RREQs w.r.t. the overall number of control packets.

The results of the previously mentioned simulations can be found in Figs. 5.2-5.5. Each data point represents an average of 10 simulation sessions with identical parameter setting, but with different initialization of the connectivity matrix. The figures show the protocol's rate of success (Fig. 5.2), the control overhead (Fig. 5.3), the number of RERRs (Fig. 5.4), and the percentage of RREQs (Fig. 5.5), for each population and for each mobility level.

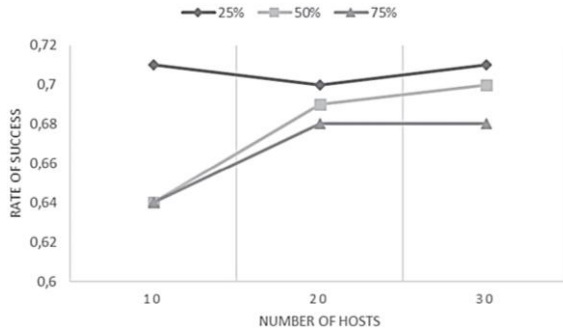

Fig. 5.2. Rate of success.

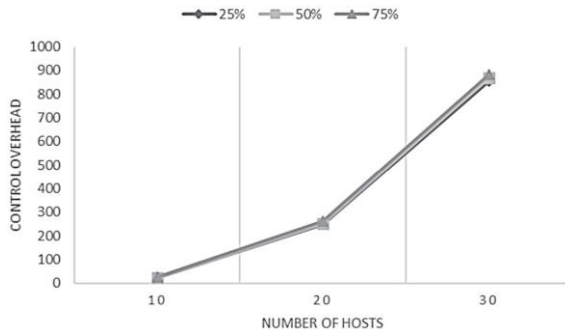

Fig. 5.3. Control overhead.

The Kruskal-Wallis test has been performed in order to check the null hypothesis, i.e., to check if the median of control overhead and route errors is equal for the MANET populations under consideration. The null hypothesis has been tested (1) for groups with different mobility levels and fixed network size, or (2) groups with different network sizes and fixed mobility level. We used this test because we have more than two independent groups to be compared, and the normality assumption is violated. The same approach has not been adopted for the rate of success and for the RREQ percentage, because they are only expressed as percentages. There isn't any statistically significant difference (at the significance level 0,01) between the control overhead induced by networks with the same population (10, 20 or 30 hosts), varying the mobility level (25, 50, and 75 %). Conversely, there is always a statistically significant difference (p-value < 0,0001) between the control overhead induced by networks with different populations and fixed mobility level. This suggests that the increasing of control overhead depends on the increasing of the network size, mainly.

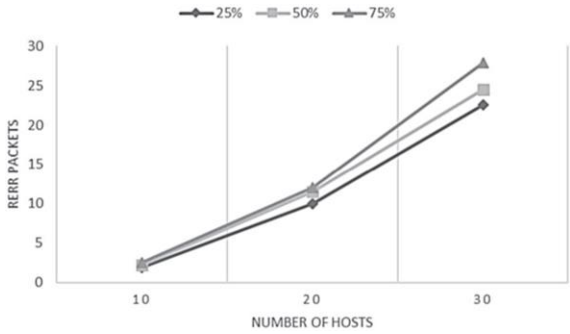

Fig. 5.4. Number of RERRs.

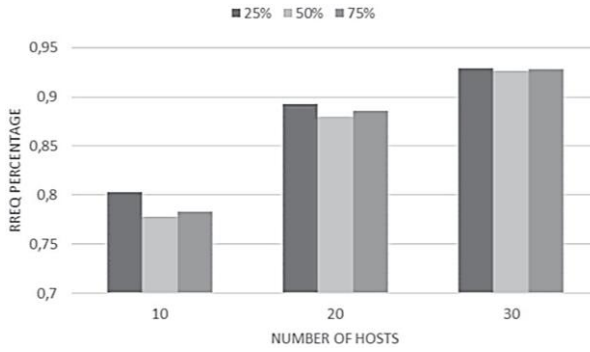

Fig. 5.5. Percentage of RREQs w.r.t. the overall number of control packets.

As for the spread of route errors along the network, it has been found that there is no statistical difference (at the significance level 0,01) between networks with 10 or 20 hosts, with a variable mobility level; this difference is statistically significant in the case of 30 hosts. Since rejecting the null hypothesis doesn't indicate which of the groups differ, the analysis has been refined by performing a pairwise comparison, using the Mann-Whitney test. As a result, there exists a statistical difference at the significance level 0,01 only between 25 % and 75 % of mobility level (p-value = 0,0002). Instead, there is always a statistically significant difference (p-value < 0,0001) between the route errors injected into the networks with different populations and fixed mobility level. These results suggest that the increasing of RERRs largely depends on the network size.

5.5. Conclusions and Future Work

In this chapter, we have introduced MOTION, a Java environment for modeling MANETs and for simulating their behavior. This tool has been used to analyze the performances of three routing protocols, and to compare the results to those that can be found in the literature. A sensible prosecution of this work could be the attempt to modeling a larger set of MANET behavior, in order to establish the usefulness of the tool, and to improve the user interface of our system, showing how the network evolves, during the computations.

References

[1]. D. P. Agrawal, Q.-A. Zeng, Introduction to Wireless and Mobile Systems, Fourth Edition, *Cengage Learning*, Boston, 2016.

[2]. A. P. Pandian, J. I.-Z. Chen, Z. A. Baig, Sustainable mobile networks and its applications, *Mobile Networks and Application*, Vol. 24, Issue 2, 2019, pp. 295-297.

[3]. A. Garcia-Santiago, J. Castaneda-Camacho, J. F. Guerrero-Castellanos, G. Mino-Aguilar, V. Y. Ponce-Hinestroza, Simulation platform for a VANET using the true time toolbox: Further result toward cyber-physical vehicle systems, in *Proceedings of the IEEE 88th Vehicular Technology Conference (VTC-Fall'18)*, 2018, pp. 1-5.

[4]. S. Basagni, M. Mastrogiovanni, A. Panconesi, C. Petrioli, Localized protocols for ad-hoc clustering and backbone formation: A performance comparison, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 17, Issue 4, 2006, pp. 292-306.

[5]. P. Arcaini, A. Gargantini, E. Riccobene, P. Scandurra, A model-driven process for engineering a toolset for a formal method, *Software: Practice and Experience*, Vol. 41, Issue 2, 2011, pp. 155-166.

[6]. J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, J. G. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, *MobiCom*, Vol. 98, 1998, pp. 85-97.

[7]. S. R. Das, R. Castaneda, J. Yan, R. Sengupt, Comparative performance evaluation of routing protocols for mobile, ad-hoc networks, in *Proceedings of the 7th International Conference on Computer Communications and Networks (ICCCN'98)*, 1998, pp. 153-161.

[8]. D. A. Tran, H. Raghavendra, Congestion adaptive routing in mobile ad-hoc networks, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 17, Issue 11, 2006, pp. 1294-1305.

[9]. S. Kurkowski, T. Camp, M. Colagrosso, MANET simulation studies: The incredibles, *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 9, Issue 4, 2005, pp. 50-61.

[10]. A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, W. L. Tan, A process algebra for wireless mesh networks, in *Proceedings of the European Symposium on Programming (ESOP'12)*, 2012, pp. 295-315.

[11]. L. Bononi, G. D'Angelo, L. Donatiello, HLA-based adaptive distributed simulation of wireless mobile system, in *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation (PADS'03)*, 2003, p. 40.

[12]. A. Singh, C. Ramakrishnan, S. A. Smolka, A process calculus for mobile ad-hoc networks, *Science of Computer Programming*, Vol. 75, Issue 6, 2010, pp. 440-469.

[13]. J. Wu, F. Dai, Mobility-sensitive topology control in mobile ad-hoc networks, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 17, Issue 6, 2006, pp. 522-535.

[14]. D. Cavin, Y. Sasson, A. Schiper, On the accuracy of MANET simulators, in *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02)*, 2002, pp. 38-43.

[15]. M. Merro, An observational theory for mobile ad hoc networks, *Information and Computation*, Vol. 207, Issue 2, 2009, pp. 194-208.

[16]. G. Delzanno, A. Sangnier, G. Zavattaro, Parameterized verification of ad-hoc networks, in *Proceedings of the International Conference on Concurrency Theory (CONCUR'10)*, 2010, pp. 313-327.

[17]. C. Xiong, T. Murata, J. Leigh, An approach for verifying routing protocols in mobile ad-hoc networks using Petri nets, in *Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, 2004, Vol. 2, pp. 537-540.

[18]. F. Erbas, K. Kyamakya, K. Jobmann, Modelling and performance analysis of a novel position-based reliable unicast and multicast routing method using coloured Petri nets, *in Proceedings of the IEEE 58th Vehicular Technology Conference (VTC-Fall'03)*, Vol. 5, 2003, pp. 3099-3104.

[19]. M. H. Jahanian, F. Amin, A. H. Jahangir, Analysis of Tesla protocol in vehicular ad-hoc networks using timed colored Petri nets, *in Proceedings of the 6th International Conference on Information and Communication Systems (ICICS'15)*, 2015, pp. 222-227.

[20]. K. Jensen, L. M. Kristensen, L. Wells, Coloured Petri nets and CPN tools for modelling and validation of concurrent systems, *International Journal on Software Tools for Technology Transfer*, Vol. 9, Issues 3-4, 2007, pp. 213-254.

[21]. E. Börger, R. Stärk, Abstract State Machines: A Method for High-Level System Design and Analysis, *Springer Verlag*, Berlin, 2003.

[22]. A. Gargantini, E. Riccobene, P. Scandurra, Model-driven language engineering: The ASMETA case study, in *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA'08)*, October 26-31, 2008, Sliema, Malta, pp. 373-378.

[23]. R. Farahbod, V. Gervasi, U. Glässer, COREASM: An extensible ASM execution engine, *Fundam. Inform.*, Vol. 77, Issues 1-2, 2007, pp. 71-103.

[24]. C. E. Perkins, E. M. Belding-Royer, S. R. Das, Ad hoc on-demand distance vector (AODV) routing, RFC 3561, *Internet Engineering Task Force*, 2003, pp. 1-37.

[25]. A. Bianchi, S. Pizzutilo, G. Vessio, Preliminary description of Nack-based ad-hoc on-demand distance vector routing protocol for MANETS, in *Proceedings of the 9th International Conference on Software Engineering and Applications (ICSOFT-EA'14)*, 2014, pp. 500-505.

[26]. A. Bianchi, S. Pizzutilo, G. Vessio, Intercepting blackhole attacks in MANETS: An ASM-based model, in *Proceedings of the International Conference on Software Engineering and Formal Methods (SEFM'17)*, 2017, pp. 137-125.

[27]. F.-H. Tseng, L.-D. Chou, H.-C. Chao, A survey of black hole attacks in wireless mobile ad-hoc networks, *Human-Centric Computing and Information Sciences*, Vol. 1, 2011, 4.

[28]. U. Glässer, Y. Gurevich, M. Veanes, Abstract communication model for distributed systems, *IEEE Trans. Software Eng*., Vol. 30, Issue 7, 2004, pp. 458-472.
[29]. A. Gargantini, E. Riccobene, P. Scandurra, A metamodel-based language and a simulation engine for abstract state machines, *J.UCS*, Vol. 14, Issue 12, 2008, pp. 1949-1983.
[30]. A. Boukerche, L. Bononi, Simulation and modelling of wireless, mobile and ad-hoc networks, in Mobile Ad Hoc Networking (S. Basagni, M. Conti, S. Giordano, I. Stojmenovic, Eds.), *IEEE Press Wiley*, New York, 2004, pp. 373-410.
[31]. E. Börger, A. Raschke, Modeling Companion for Software Practitioners, *Springer*, 2018.