Nodes Self-similarity to Test Wireless Ad Hoc Routing Protocols

Cyril Grepet and Stephane Maag

Institut National des Télécommunications
CNRS SAMOVAR
9 rue Charles Fourier
F-91011 Evry Cedex
{Cyril.Grepet, Stephane.Maag}@int-evry.fr

Abstract. In this paper we present a new approach to test the conformance of a wireless ad hoc routing protocol. This approach is based on a formal specification of the DSR protocol described by using the SDL language. Test scenarios are automatically generated by a tool developed in our laboratory. A method enabling to execute them on an implementation into a real network is illustrated. Indeed, an important issue is to execute some generated test scenarios on a dynamic network in which the links are routinely modified. Therefore, the concept of self-similarity is presented to reduce the number of nodes by collapsing them in a real network. This enables to execute the test scenarios in defining a relationship between the network and specification topologies.

1 Introduction

A wireless mobile ad hoc network (MANET) is a collection of mobile nodes which are able to communicate with each other without relying on predefined infrastructures. In these networks, there is no administrative node and each node participates in the provision of reliable operations in the network. The nodes may move continuously leading to a volatile network topology with interconnections between nodes that are often modified. As a consequence of this infrastructureless environment, each node communicates using their radio range with open transmission medium and some of them behave as routers to establish multi-hop connections. Due to these aspects and the limited resources of the mobile nodes, efficient routing in ad hoc networks is a crucial and challenging problem for the quality of the communication systems.

From these unique characteristics of ad hoc networks, many requirements for routing protocol design are raised. Protocols can be classified mainly into three categories: the *proactive*, *reactive* and *hybrid* protocols. Classes such as *hierarchical*, *geographical* or *multicasting* protocols also emerge.

The techniques used by the ad hoc network experts to design and ensure the quality of their protocols essentially rely on descriptions for simulations and/or emulations. These methods provide an idea of the real behavior of the

A. Petrenko et al. (Eds.): TestCom/FATES 2007, LNCS 4581, pp. 123–137, 2007.
© IFIP- International Federation for Information Processing 2007

implemented protocol in a node. However, the testing coverage is rather low and is only restricted to the simulation context.

Formal description techniques and their testing tools are rarely applied in such kind of networks. The main reason is the difficulty to take into account the MANET protocol characteristics and the mobility of nodes in the test sequences generation and their execution. Therefore, our work focuses on a new testing technique to check the conformance of these ad hoc routing protocols. We present in this paper a formal specification of the Dynamic Source Routing (DSR) protocol from which we may generate some test scenarios. Nevertheless, the execution of these scenarios is currently an issue. Indeed there is often a gap between the dynamic topology designed in a specification and the one of a real case study. Therefore, we illustrate in this paper the concept of Node self-similarity in order to execute generated test scenarios on a real wireless ad hoc routing protocol taking into account the network topologies.

In the Section 2, we present the related works dealing with formal methods enabling to test ad hoc routing protocols. In Section 3, our DSR formal model is described and the conformance testing approach is depicted. Then, in Section 4, the concept of self-similarity to combine real nodes is illustrated and in Section 5 an example is developed. Finally we conclude the paper.

2 Related Works

Conformance testing for ad hoc routing protocols is crucial to the reliability of those networks. Paradoxically, few works currently exist on the formal specifications to analyse routing protocol testing [1]. The majority of these works rely on non-formal models provided as input to simulators such as NS-2 [2] or OpNet [3]. However, as is often noted, the simulation and emulation techniques do not replace a real case study [4]. Indeed, normal or ideal behaviors obtained by simulation may be proved erroneous in the real case. This is why formal description techniques are required to test this kind of protocols.

In [5] and [6], two routing protocols are validated using a testbed and analysing the network performances. But many constraints are applied and no formal model is used. In [7], a formal validation model named RNS (Relay Node Set) is studied. It is well suited to the conformance testing but denotes two drawbacks. First, only experts of the proposed languages may establish the tests which does not facilitate the test of other protocols. Secondly, only metrics of the protocol may be tested (overhead control, power consumption, etc.) and no basic functionalities (reception of a RteReply, etc.). Moreover, interactions between nodes may not be tested with this method.

A formal model using Distributed Abstract Machines allows to specify the LTLS protocol [8]. Nevertheless, the specifications are not executable and no functional analysis of the protocol is realized. In addition, the authors do not consider the testing process of an implementation from these models. Even if the syntax and the semantic are interesting, this formal description is still unusable for the conformance testing.

The game theory is also used in order to specify and analyse ad hoc routing protocols [9]. But two main inconveniences appear. First, non determinism is not allowed in this model and random behavior of nodes is not specified. Secondly the inherent constraints of this kind of networks are not considered. Indeed, a very strong assumption in this work is that every node needs to have a global knowledge of the network topology which is unusual in real case studies.

In our work we propose a new approach relying on well-known formal methods in order to enable conformance testing of ad hoc routing protocols.

3 Conformance Testing of an Ad Hoc Routing Protocol

Testing techniques can be divided in two categories: active testing which relies on stimulation and observation of an implementation, and passive testing which only observes the system without interactions [10], [11], [12]. Our research focuses on active testing of ad hoc routing protocols.

Conformance testing usually relies on the comparison between the behavior of an implementation and the formal specification of a given protocol i.e a conformed implementation has to behave as its specification.
The conformance testing procedure follows these steps :

- Step 1. Define a *testing architecture* with respect to the characteristics of the system under test and its possible implementations. This step could impact on each following step and has to be defined according to the context.
- Step 2. Make some *assumptions* that are sometimes required to enable the test.
- Step 3. Construct a precise *formal specification* of the system to be tested. This specification takes into account the system functionalities as well as the data specific to the test environment (test architecture, test interface, etc.).
- Step 4. *Select* the appropriate tests. This step corresponds to the definition of the test purposes. A test purpose can be a specific property of the system such as tasks or assignments with regard to values of variables, or the behavior of a specific component of the system taking into account the current values of the variables.
- Step 5. *Generate* the test scenarios. The test purposes are used as a guide by an algorithm based on simulation to produce the test scenarios. As a result, our algorithm computes a test scenario that can be applied to the implementation under test to verify the test purpose. A scenario is a sequence of interactions (between the system and the environment) that includes the interactions that represent a test purpose.
- Step 6. *Format* the test scenarios i.e to produce test scenarios in some accepted formalism as *Message Sequence Charts* (MSC), a formalism widely used in industry to describe message exchanges, or in *Testing and Test Control Notation* (TTCN), the ITU-TS standard language used for test specification.

Problematic. The main goal tackled in this paper is to provide a reliable method to test a routing protocol in a network in which we do not control

neither the number of nodes nor the mobility scenario. Therefore, three main and relevant problems may be defined.

Mobility representation: The mobility in ad hoc networks implies that a specification has to represent more than one node communicating with each other. Thus the specification has to allow the creation or the suppression of a link between a pair of nodes in order to represent their mobility.

Test sequences generation: Another objective is to try to maximize the automation of the test sequences generation from classical tools avoiding the well-known state space explosion problem.

Test execution: The mobility of nodes and the hazard of radio communications can lead to many inconclusive verdicts or even to prevent the test. Moreover, the testing architecture to be used is also subject to the same problems. This problematic has to be study in order to provide a reliable verdict to the implementation conformity.

In the remaining of the paper we present our solutions to these problems with respect to the six steps aforementioned.

3.1 Testing Architecture

Some testing architectures are proposed by the ISO standard [13]. The coordinated, remote and distributed test architectures need reliable communications between the implementation under test (IUT) and the other components of the test. Due to the nodes mobility and in order to provide a general approach, we can not ensure that both sites can always communicate with each other. Due to the inherent constraints of ad hoc networks, a local testing architecture is chosen. We describe the different components of our architecture as follows. First, in order to observe packets, we need some Points of Observation (PO), whereas to observe and control these packets (if the IUT allows it (white/grey/black box testing)) we need Points of Control and Observation (PCO). These points are connected with the upper and lower testers (UT and LT) which are controlled by the test coordination procedure (TCP). The PO/PCO connected to UT aim to control the packets between the IUT and the upper layer (IP for example) whereas those connected to LT aim to control communications between the IUT and the lower layer (the link layer for instance). Each time one of the testers observes a packet, the TCP checks if it is the one expected regarding the specification. The testing architecture is depicted in Figure 1.

This architecture enables to observe and to control the message exchanges between the implementation and the strict upper and lower layers. In our case study (see section 5) the IUT is a DSR implementation. The UT controls the packet between DSR implementation and IP layer whereas the LT controls the communication between DSR implementation and the link layer.

3.2 Testing Assumptions

In order to execute test sequences in a real network and to provide reliable verdicts, some testing assumptions are required. First, we make the classical

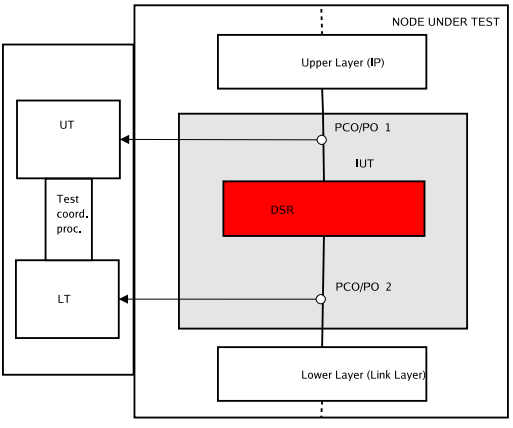

Fig. 1. Local test architecture applied on IUT

assumption that the implementation could be tested, i.e we can install some Points of Control (PCO) or Points of Observation (PO) (see section 3.1) into the IUT. The specific constraints of ad hoc networks imply that we have to make some assumptions to take into account the mobility of the nodes. Six hypothesis are defined:

1. *Existing destination nodes:* Each destination node (D) of the packets used by our test scenarios exists and is or will be connected by the network to the IUT.
2. *Connectivity of node D:* We assume that in a reasonable time, one or more paths will enable to execute the test scenarios between IUT and D.
3. *Stability of routes:* The routes which allow IUT and D to communicate with each other will remain stable during the execution of the test scenarios. This assumption is necessary to realize conformance testing and relies on the fact that an ad hoc network is created in order to allow communication for a community. If the communications are reliable enough for this purpose, we can suppose that the routes will allow to execute the test scenarios.
4. *Replay:* Despite the connectivity and stability assumptions, the test could sometimes fail or be inconclusive due to the radio or topological hazards. Thus, to avoid wrong decisions, we have to replay test scenarios before giving the verdict.
5. *Fail:* If the test is too many times "inconclusive" then we may consider that the test has failed. The testing replay number has to be decided according to the implementation and the test context.
6. *Implementation choices:* We assume that the implementation under test has the same options as the context. Capability testing techniques can be applied to check that aspect [14]. This assumption is necessary in conformance testing as well as in interoperability testing to prevent wrong decisions.

These assumptions allow to ensure a reliable testing environment by reducing wrong final verdicts.

3.3 Formal Specification

The third step is to formally describe the system to be tested. It is necessary to choose a formal model to reach this objective. We select the Extended Finite State Machines (EFSMs) [14] that are well adapted to describe communication protocols.

Definition 1. *An EFSM M is defined as :* $M = (I, O, S, \overrightarrow{x}, T)$ *with I, O, S,* $\overrightarrow{x}$ *and T respectively a set of input symbols, a set of output symbols, a set of states, a vector of variables and a set of transitions. Each transition* $t \in T$ *is a 6-tuple defined as :* $t=(s_t, q_t, i_t, o_t, P_t, A_t)$ *where*

- s_t *is the current state,*
- q_t *is the next state,*
- i_t *is an input symbol,*
- o_t *is an output symbol,*
- $P_t(\overrightarrow{x})$ *a predicate on the values of the variables,*
- $A_t(\overrightarrow{x})$ *an action on the variables.*

The language selected to provide the specification is the Specification and Description Language (SDL) standardized by ITU-T [15]. This is a widely used language to specify communicating systems and protocols, based on the semantic model of EFSM. Its goal is to specify the behavior of a system from the representation of its functional aspects. It allows to describe the architecture of the system i.e the connection and organization of the elements (blocks, processes, etc.) with the environment and between them. The behaviors of the entities in terms of their interactions with the environment and among themselves may also be designed. These interactions are described by tasks, transitions between states, and are based on the EFSMs.

All along our work, we assume that the conformance testing of a routing protocol could be performed in an unknown network topology. SDL allows to describe network topologies by using node instances, although it is impossible to guarantee that the topology of the real network will match the one of the specification. In that way, this work aims to reduce the number of nodes required to generate the test sequences and also to take into account their eventual mobility, according to the test objectives. The minimization of the specification helps to avoid the state space explosion problem. Nevertheless it is necessary to map it with the implementation in a real network.

This is the subject of our next section.

4 Self-similarity of Nodes

In this section we present the self-similarity of nodes used to reduce the number of nodes in the specification. Furthermore it allows the resulting test sequences to be executed on large class of real topologies. This approach takes into account the strong constraint of mobility in MANETs adapting the known self-similarity to this specific context as it is presented hereafter.

4.1 Definition of Self-similarity

Node self-similarity is inspired by [16] where fixed nodes in a wired network are merged in one single node with the main assumption that the communications are reliable between the combined nodes. In our work, we deal with wireless ad hoc nodes and unreliable links. We take into account these inherent constraints in the remaining of this work. First we have to define the combination of EFSMs.

Definition 2. *Let* $\{N_i\}_{i\in E}$ *where* $E \in [1..n]$ *and* $n \in \mathbb{N}$ *be a collection of model that can be described as EFSMs. We note* $N_1 \circ ... \circ N_n$ *the combination of all* N_i *defined as :*
$O(N) = \bigcup_{i\in E} O(N_i)$
$I(N) = \bigcup_{i\in E} I(N_i) - \bigcup_{i\in E} O(N_i)$
$S(N) = \Pi_{i\in E} S(N_i)$
$\overrightarrow{x}(N) = \Pi_{i\in E} \overrightarrow{x}(N_i)$
$T(N) = (s, s', e, o, P(\overrightarrow{x}), A(\overrightarrow{x}))$ *if* $(s_i, s'_i, e, o, P_i(\overrightarrow{x}), A_i(\overrightarrow{x})) \in T(N_i)$
where $P_i(\overrightarrow{x}) \equiv P_i(\overrightarrow{x_i})$, $A_i(\overrightarrow{x}) \equiv A_i(\overrightarrow{x_i})$, $e \in I(N_i)$ *and* $o \in O(N_i)$
Let $\Phi \subset O(N)$*, we define* $ActHide_\Phi(N)$ *as the obtained EFSM from* N *where each action of* Φ *becomes an internal one. This application transforms the communications between the different components of* N *into non-observable actions.*

Thus we can define the self-similarity of two nodes as :

Definition 3. *Let two possible actions for a node be* $send(Message, n, m)$ *and* $receive(Message, n', m')$ *where* n *(respectively* m'*) the observed node,* m *(respectively* n'*) the destination of the packet (respectively sender), and* $Message$ *the whole possible contents of a packet. Let* N *be a node specification. We note* $Tr(N)$ *the set of observable traces, a trace being an input/output sequence. Beside,* $Tr(N)$ *is a finite set, indeed the variable domains of the EFSM are discrete and finite (as most of the communication protocols).*

Some $N_{i\in I}$ *are self-similar if :*
$Tr(ActHide_\Phi(N_1 \circ N_2)) \subseteq Tr(N)$,
where $\Phi = \{send(Message, N_1, N_2), send(Message, N_2, N_1),$
$receive(Message, N_1, N_2), receive(Message, N_2, N_1)\}$

The self-similarity approach is easily applied in a wired network but due to mobility and the unreliable communications in a MANET, it can not be performed directly. The combination of mobile nodes is impossible, indeed the trace property could not exist if for example two consecutive nodes on a route can not communicate each other anymore.

Therefore, we use the self-similarity with three restrictions:

1. The self-similarity is applied from the point of view of a single node which is the IUT.
2. The self-similarity is applied each time a packet of the test sequences is received or sent in order to simplify the possible topologies known by the IUT.

3. The self-similarity is applied only for a specific communication on a defined route (not considering all communications in the network) between the IUT and another node.

The main idea, by reducing the number of nodes in the specification, is to identify all the different possible node behaviours according to the test purposes to define a minimal topology required and sufficient to generate test sequences for each test purpose. In order to illustrate our approach we choose the *Dynamic Source Routing* (DSR) protocol as a real case study.

5 A Case Study: DSR

5.1 Dynamic Source Routing Protocol

Dynamic Source Routing (DSR) is a reactive protocol that discovers and maintains routes between nodes on demand [17]. It relies on two main mechanisms, *Route Discovery* and *Route Maintenance*. In order to discover a route between two nodes, DSR floods the network with a *Route Request* packet. This packet is forwarded only once by each node after concatenating its own address to the path. When the targeted node receives the *Route Request*, it piggybacks a *Route Reply* to the sender and a route is established. Each time a packet follows an established route, each node has to ensure that the link is reliable between itself and the next node. DSR provides three successive steps to perform this maintenance: link layer acknowledgment, passive acknowledgment, and network layer acknowledgment. If a route is broken, the node which detects the failure sends by piggybacking a *Route Error* packet to the original sender.

5.2 DSR Formal Model

DSR is specified using SDL and the formal model describes the DSR draft 10 with the *Flow State Extension* (our specification is detailed in [18]). We do not specify all the possible features. Only some basic features for *Route Maintenance*, *Route Discovery* as the *Cached Route Reply* and all the main structures required by DSR (as the *Route Cache* or the *Send Buffer*) are described.

In order to represent the different links between nodes, we use a special block, called *Transmission* that contains a matrix where we define the connectivity in the networks. The matrix could be easily updated by sending information to create or remove a link. It means that we may modify dynamically the topology of the network in the purpose of representing the node mobility by changing their neighborhood. Details are given in [18]. Besides, we do not specify how to support multiple interfaces or security concepts.

5.3 Specification Reduction Using Nodes Self-similarity

First, a node may behave as a source S, a router N_i or a destination D. A route is defined as a succession of S, N_i *where* $i \in [1..n]$ and D (Figure 2). We consider the nodes in the route from the point of view of S which is the IUT. Two possible

cases arise during a communication between nodes on a particular route: either the communication between two successive nodes N_i and N_{i+1} succeeds, or it fails. We consider a communication as a success if a packet received by N_i is forwarded to N_{i+1} and forwarded after to N_{i+2} without provoking a *RteError* regardless of the meaning used for the acknowledgment. In the following $N_\circ$ illustrates the abstract node defined as:

$$N_\circ = \begin{cases} N_\varnothing & \text{where } N_\varnothing \text{ is the neutral element.} \\ N_x \circ N_{x+1} & \text{where } N_x \in \{N_\varnothing\} \cup \bigcup_{i=1}^{n} \{N_i\}. \end{cases}$$

$N_\varnothing$ represents a node that only forwards the packets without modifying anything in the packet.

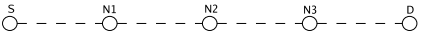

Fig. 2. A simple route between S and D

The process of nodes self-similarity may be illustrated as follows:

– *Transmission success:* If a transmission between N_i and N_{i+1} succeeds, we combine these two nodes in a new node $N_\circ$. The communications between N_i and N_{i+1} are considered as $N_\circ$ internal actions. If the communication between $N_\circ$ and N_{i+2} succeeds, we iterate the process and so on. Thus, in case that the packet from S reaches D without causing a *RteError*, we may combine all the intermediate nodes as illustrated in Figure 3.

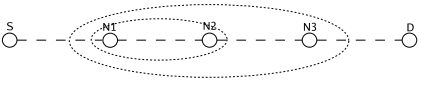

Fig. 3. Combination by self-similarity when all communications succeed

– *Transmission failure:* If a communication fails between N_i and N_{i+1}, it means that all the previous communications have succeeded. So the nodes between N_1 and N_i are combined. Finally, all the nodes after N_{i+1}, including D have the same behavior for an observer placed on the IUT. We therefore combine all the nodes from N_{i+1} to D into a new node D (Figure 4).

We can note two exceptions: if the failure occurs between S and N_1 or S and D when a direct link exists.

However, from the point of view of S and with respect to our test sequences, the length of the main route does not matter. Indeed, a direct connection or a multi-hop route (except for a route sorting in the RouteCache) are observationnaly equivalent to S. Therefore, we handle the exceptions as it follows.

- If a direct link between S and D is broken, we introduce $N_\circ = N_\varnothing$ as a node on the route matching our specification described in section 5.4

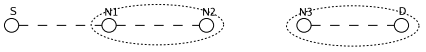

Fig. 4. Combination by self-similarity when a communication fails

- if the broken link takes place between S and N_1, we have combined all the N_i and D in a new node D. Thus this leads to a similar situation than in the first case, we also introduce $N_\circ = N_\varnothing$ between S and the combined node D

The topology is reduced by using self-similarity in each case for the DSR protocol. The nodes self-similarity allows to represent a large class of topologies with a small number of nodes and to execute test sequences regardless of the number of intermediate nodes. Thereby we can reduce the number of nodes used in our specification in order to generate test scenarios which is an important issue in the testing activities especially for wireless networks.

5.4 Test Scenarios Equivalence

In order to generate adjustable test scenarios to a large class of topology taking into account the nodes mobility, we minimize our specification. Due to nodes self-similarity that allows to reduce a route between two nodes in a two-hop one, we can keep only the smallest number of nodes required to generate a test sequence according to specific test objectives. To test functional properties of DSR, no test objectives requiring more than two routes in the network were found out. Then, our specification can be reduced into four nodes, S, N_0, N_1 and D which compose two routes $[S, N_0, D]$ and $[S, N_1, D]$ as represented in Figure 5.

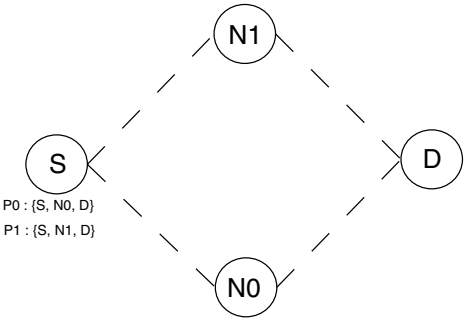

Fig. 5. Specification topology

Despite nodes self-similarity allows to reduce the length of route, the specification describes only two routes and the real network could have more than two between the IUT and D. The main idea is therefore to create a relation between the specification and the implementation by two sets $P0$ and $P1$ defined as:

Definition 4. *Definition of sets*
Let S_{spec} and D_{spec} be respectively the representations of S and D in the specification Spec and S_{imp}, D_{imp} their representations in the implementation Imp. Let $(p_n(x) \mid (x,n) \in \mathbb{N})$ be the n^{th} route chosen by S_{imp} to reach D_{imp} and composed by x nodes.

- *In Spec:*
  - $P0_{spec} = \{(S_{spec}, N0, D_{spec})\}$
  - $P1_{spec} = \{(S_{spec}, N1, D_{spec})\}$
- *In Imp:*
  - $p_n(x) \in P(n \bmod 2)_{imp}$ *i.e all the route with 0 or an even subscript will be in the set $P0_{imp}$ and those with an odd subscript will be in the set $P1_{imp}$. Thus, if there's a RouteError we preseve the route alternance between the set.*

All along the test execution, the *Test Coordination Procedure* (TCP, see section 3.1) will preserve a relation between $P0_{spec}$ and $P0_{imp}$, and also between $P1_{spec}$ and $P1_{imp}$. For instance, if the routing metric is "minimal hop count" (assumed in the rest of the paper), TCP will affect in $P0_{imp}$ the shortest path as $p_0(x)$, in $P1_{imp}$ the second as $p_1(y)$, in $P0_{imp}$ the third as $p_2(z)$ and so on with $x \leqslant y \leqslant z \leqslant ...etc.$ Both sets save the theoretical *RouteCache* in the TCP. Here "theoretical" is used because the *RouteCache* could eventually be filled in an other way during the *RouteDiscovery* mechanism. This problem will not be discussed in this paper because we focus here our study on functional properties.

With respect to *Spec*, $P0_{imp}$ and $P1_{imp}$ match possible routes described in the specification. For instance, if a test sequence implies that $P0_{spec}$ disappears: the TCP will detect the RouteError packet as an input, will erase the first element of $P0_{imp}$ i.e $p_0(x)$ and will select $p_1(y) \in P1_{imp}$ as the new route that IUT must use.

We have to underline the fact that a node N could belong to several routes (a *GratuitousRouteReply* sent to S by the node involved into different routes).

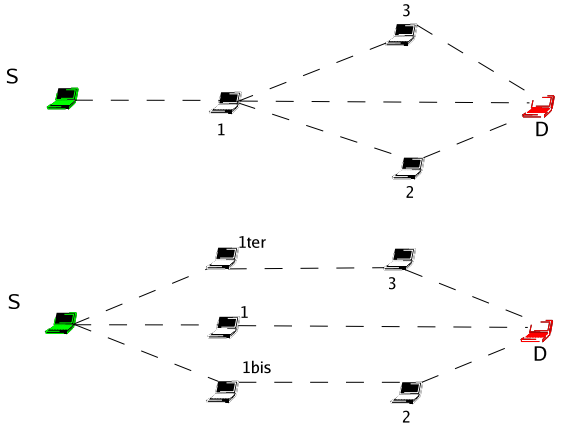

Fig. 6. Self-Similarity of a node involved in different routes

In this case, with respect to our sets, we have to duplicate this node *nbr* times where *nbr* is the number of routes containing N in the network representation of the TCP. Thus we create *nbr* routes in order to apply nodes self-similarity to each one and to split up these routes into our two sets $P0_{imp}$ and $P1_{imp}$ (Figure 6).

5.5 Experiment Context

Our approach is applied on a experimentation through the DSR-UU implementation [19]. The test sequences are provided by one of our tools TESTGEN-SDL [20] and some test purposes. Direct emulation is used. It allows to use a real implementation of a protocol stack with a simulator to represent the mobility and to manage the communications.

The direct emulation is performed on a focal machine with the following characteristics:

- Pentium©M 1,6 GHz
- 512 Mo Ram
- Fedora-2.6.15 kernel with skas patch
- TUN/TAP interfaces activated

We use *User Mode Linux* [21] to create virtual machines with existing prepared kernel and file system [22]. DSR-UU was added in the kernel.

NS-2 patched for emulation was performed to manage mobility and wireless communication between the virtual machines. We may note that a maximum of six virtual nodes is possible on a same focal machines. If we want a larger collection of nodes, it is necessary to distribute the virtual machines on more than one focal computer. The proposed emulation and testing architecture are depicted in Figure 7.

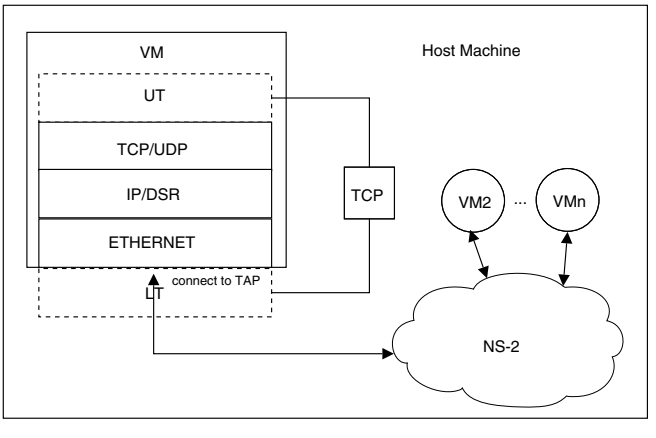

Fig. 7. Direct emulation and testing architecture

5.6 Sets Management for Unexpected Messages

In case of a broken link or detected unexpected *RouteError* from a node belonging to multiple routes, or unexpected *RouteError*, an inconclusive verdict could be obtained. The experimentations have shown that despite our assumptions, the number of inconclusive verdicts is important depending on the mobility. Thus, we automatize an error recovery for this kind of messages. An algorithm is defined to maintain the relation between $P0_{imp}$ and $P1_{imp}$ with the routes of *Spec* taking into account unexpected *RouteError* packets. A received *RouteError* could be characterized by two criterias:

1. expected/unexpected packet
2. the failure is/is not on the route used by the test scenario

Global Set Management Algorithm receiving a RouteError
Let $P_{imp} = P0_{imp} \cup P1_{imp}$ (w.r.t *spec*). Let P be a pointer in the TCP selecting the set containing the route of the test scenario and $\bar{P}$ its complement.
Let p be a pointer in the TCP on the first element of P.
Let lr be a broken link built from the address couple (i,j) of identified nodes carried by the RouteError packet.
Let $a = 1$ if the RouteError is expected else 0

1. **If** $lr \in p$ et $a = 1$ **then**
   (a) $P_imp := P_imp \setminus \{p_n(x) | lr \in p_n(x)\}$.
   (b) **if** $P = P0_{imp}$ **then** index each route by $n \in [1..m]$ **else** $n \in [0..m-1]$, with m the number of known routes.
   (c) To build sets $P0_{imp}$ and $P1_{imp}$.
   (d) $P := \bar{P}$.
   (e) $p := P(1)$ the first element of P.
   (f) To continue the test if it is possible.(*)
2. **If** $lr \in p$ and $a = 0$ **then**
   (a) $P_imp := P_imp \setminus \{p_n(x) | lr \in p_n(x)\}$.
   (b) **if** $P = P0_{imp}$ **then** index each route by $n \in [0..m-1]$ **else** $n \in [1..m]$, with m the number of known routes.
   (c) To build sets $P0_{imp}$ and $P1_{imp}$.
   (d) $P := P$.
   (e) $p := P(1)$.
   (f) To restart the test one step before(*)
3. **If** $lr \notin p$ and a **then**
   (a) $P_imp := P_imp \setminus \{p_n(x) | lr \in p_n(x)\}$.
   (b) **if** $P = P0_{imp}$ **then** index each route by $n \in [0..m-1]$ **else** $n \in [1..m]$, with m the number of known routes.
   (c) To build $P0_{imp}$ and $P1_{imp}$.
   (d) $P := P$.
   (e) $p := p$.
   (f) To continue the test.

(*) i.e the previous SourceRoute sent. If the test scenario needs a route to send a message and $P_{imp} = \varnothing$, an inconclusive verdict will be obtained.

6 Conclusion

In this paper we present a new approach to test the conformance of a wireless ad hoc routing protocol, namely DSR. This approach is based on a formal specification of the protocol described in SDL. This work has as a main advantage to formally test such kind of protocols and to take into account the nodes mobility and the network volatility aspect as well. Test scenarios are automatically generated by a tool developed in our laboratory and a method is illustrated enabling to execute them on a real implementation into a real network. This technique called the nodes self-similarity allows to bridge the gap between the dynamic topologies of a real network and the ones of the specification. This allows to reduce the number of nodes in a specification in order to generate the sequences and then avoiding the eventual state space explosion. The main advantage of this method is the possibility to execute a test sequence generated from a usable specification on an implementation running in a real mobile ad hoc network. An algorithm is depicted in order to illustrate the relationship during the testing process between the tester, the specification and the IUT. Finally, an application with an emulator is illustrated in which only four nodes are necessary to generate some test scenarios.

References

1. Obradovic, D.: Formal Analysis of Routing Protocols. PhD thesis, University of Pennsylvania (2002)
2. NS: The network simulator (2004) http://www.isi.edu/nsnam/ns
3. OPNet: The opnet modeler (2005) http://www.opnet.com/products/modeler/home.html
4. Bhargavan, K., Gunter, C., Lee, I., Sokolsky, O., Kim, M., Obradovic, D., Viswanathan, M.: Verisim: Formal analysis of network simulations. IEEE Trans. Softw. Eng. 28(2), 129–145 (2002)
5. Yi, Y., Park, J.S., Lee, S., Lee, Y., Gerla, M.: Implementation and validation of multicast-enabled landmark ad-hoc routing (m-lanmar) protocol. In: IEEE MILCON'03 (2003)
6. Bae, S., Lee, S.J., Gerla, M.: Multicast protocol implementation and validation in an ad hoc network testbed. In: Proc. IEEE ICC, pp. 3196–3200 (2001)
7. Lin, T., Midkiff, S., Park, J.: A framework for wireless ad hoc routing protocols. In: Proc. of IEEE Wireless Communications and Networking Conf (WCNC) (2003)
8. Glasser, U., Gu, Q.P.: Formal Description and Analysis of a Distributed Location Service for Mobile Ad Hoc Networks. Frazer Univ. (2003)
9. Zakkiudin, I.: Towards a game theoretic understanding of ad-hoc routing. In: ENTCS. vol. 119, pp. 67–92 (2005)
10. Lee, D., Chen, D., Hao, R., Miller, R., Wu, J., Yin, X.: A formal approach for passive testing of protocol data portions. In: Proceedings of the IEEE International Conference on Network Protocols, ICNP'02 (2002)
11. Alcalde, B., Cavalli, A., Chen, D., Khuu, D., Lee, D.: Network protocol system passive testing for fault management - a backward checking approach. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 150–166. Springer, Heidelberg (2004)

12. Arnedo, J., Cavalli, A., Nunez, M.: Fast testing of critical properties through passive testing. In: Hogrefe, D., Wiles, A. (eds.) TestCom 2003. LNCS, vol. 2644, pp. 295–310. Springer, Heidelberg (2003)
13. ISO: information technology - Open Systems Interconnections - Conformance testing methodology and framework (1992)
14. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - a survey. IEEE Transactions on Computers 84, 1090–1123 (1996)
15. ITU-T: Recommandation Z.100: CCITT Specification and Description Language (SDL) Technical report, ITU-T (1999)
16. Djouvas, C., Griffeth, N., Lynch, N.: Using self-similarity for effecient network testing. Technical report, Lehman College (2005)
17. Johnson, D., Maltz, D., Hu, Y.C.: The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR) - Experimental RFC. IETF MANET Working Group (2004) http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt
18. Maag, S., Grepet, C., Cavalli, A.: Un Modèle de Validation pour le Protocole de Routage DSR. In: Hermes, (ed.) CFIP 2005, pp. 85–100. Bordeaux, France (2005)
19. Nordstrom, E.: Dsr-uu v0.1. Uppsala University, http://core.it.uu.se/core/index.php/DSR-UU
20. Cavalli, A., Lee, D., Rinderknecht, C., Zaidi, F.: Hit-or-jump: An algorithm for embedded testing with application to IN services. In: Wu, J., Chanson, S., Gao, Q. (eds.) Formal Method for Protocol Engineering and Distributed Systems, FORTE XII/PSTV XIX'99. IFIP Conference Proceedings, Beijing, China, vol. 156, Kluwer, Dordrecht (1999)
21. Dike, J.: user-mode-linux, http://user-mode-linux.sourceforge.net/
22. Wehbi, B.: Dynamic remote access solution to a hot-zone. Master's thesis, Université Pierre et Marie Curie (2005)