A Validation Model for the DSR protocol

Ana Cavalli, Cyril Grepet and Stéphane Maag
GET / Institut National des Telecommunications
9, rue Charles Fourier
F-91011 Evry Cedex, France
email : {Ana.Cavalli, Cyril.Grepet, Stephane.Maag}@int-evry.fr

***Abstract*— This paper presents a validation model for the Dynamic Source Routing (DSR) protocol. This model is based on a formal specification of the protocol. It also provides a verification technique to verify the protocol against the IETF DSR draft requirements [1] as well as a testing technique for the generation of a set of scenarios to check the conformance of a given implementation. The DSR protocol has been specified following the IETF DSR draft. The formal specification has been designed using the SDL language and the scenarios have been generated from the specification using a method and a tool developed at INT [2]. The test generation method is based on a set of test purposes that express specific system properties and is completely automated. In this paper, we also present the experimentation results of the application of our tool to the DSR protocol.**

Keywords: Ad hoc wireless networks, routing protocol, DSR, conformance testing, SDL.

I. Introduction

A wireless mobile Ad hoc network (MANET) is a collection of mobile nodes which are able to communicate 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 end-to-end connections. Due to these aspects and the fact that the resources are limited in mobile nodes, efficient routing in ad hoc networks is a crucial and challenging problem.

From these unique characteristics of ad hoc networks, many requirements for routing protocol design are raised. There have been many proposals for routing protocols for ad hoc networks, and several protocols have emerged. They can be classified into three main categories: the *proactive* [3], *reactive* [1], [4] and *hybrid* [5], [6] protocols. The Dynamic Source Routing protocol (DSR) [1] is a simple and efficient reactive routing protocol allowing the network to be completely self-organizing and self-configuring without the need of infrastructure or administration. This protocol will certainely be standardized soon by the IETF. Moreover, due to a previous study showing that reactive routing protocols are better suited to ad hoc networks than proactive ones [7], some DSR implementations have emerged [8].

However, most of the time, the research efforts dedicated to these protocols are focused on simulation. Only a few implementations have been produced, and these proposed implementations environments had no more than a dozen nodes.
The reasons are the difficulties to implement such routing protocols, the cost and material requirements to develop them and to insure that all the functionalities presented in the IETF standards have been implemented [9].

As a consequence, conformance testing becomes a crucial phase of the ad hoc routing protocol design and development. Indeed, functionality and security failures caused by poor testing may have a catastrophic effect on the reliability of mobile wireless networks. For example, we may easily lose data, which may be used by other terminals outside the confidential network. An intruder may have access to data, etc. In the past, different conformance testing methods have been developed for distributed communicating systems. Some of them can be easily adapted to the validation of such routing protocols. This is the case for goal-oriented testing techniques which consist of selecting a specific property of the system that is likely to be false or the behavior of a specific component of the global system that is likely to be faulty, and generating test scenarios for only those parts. In general, this selection is made by human experts to identify the part of system's behavior or the expected properties that might be subject of testing and to formulate test purposes or goals based on this identification [10], [11].

In this paper, we propose a validation model of DSR, based on a formal specification, a verification technique and a goal-oriented testing technique. This work is a contribution to the development of correct DSR specifications and implementations by using formal specification techniques, verification techniques and test scenarios generated from the specification.

For the formal specification of the DSR protocol, we use the SDL language [12]. This language is well adapted to the description of protocols. It allows to provide a hierarchical description of the system using different architecture levels. This is well fitted to specify IP, DSR and the link layer. Verification of the DSR protocol is based on model checking techniques and we use a model checker integrated in the tool used for SDL protocol specification and simulation [13]. In order to perform goal-oriented testing, we use a method and a tool developed at INT [2]. The method is based on test purposes and the tool allows the generation of tests based on these test purposes. In this paper, we apply this tool to

generate test scenarios for some functionalities (for instance the execution of a Route Request), expressed as properties or purposes to be tested. These test scenarios will be used to check that different implementations of the DSR protocol satisfy the required functionality.

The paper is organized as follows. Section II provides an outline of the DSR protocol. The two mechanisms of the protocol, Route Discovery and Route Maintenance are presented in this section. Section III presents the SDL specification of DSR based on IETF DSR draft requirements [1]. Section IV presents the validation of the DSR protocol, including the protocol verification, the test scenarios generation and the results of the application of the test procedure to the DSR protocol . Finally, Section V concludes the paper and gives the perspectives of this work.

II. DSR PROTOCOL OVERVIEW

The *Dynamic Source Routing* protocol (DSR) [14], [1] is a simple and efficient reactive Ad hoc unicast routing protocol. It has been designed specifically for use in multi-hop wireless Ad hoc networks of mobile nodes. By using DSR, the network does not need any network infrastructure or administration and it is completely self-organizing and self-configuring. This routing protocol consists of two mechanisms: the *Route Discovery* and *Route Maintenance* that permit to completely maintain and automatically determine routes. These mechanisms are described in the following subsections.

A. Route Discovery

Route Discovery is one of the two mechanisms of the DSR protocol. It is in charge of finding routes between the nodes in the network. DSR is a reactive protocol, a source node S starts searching routes only if it needs to send a packet to a destination D and if no routes are enclosed in its cache. To find routes, DSR employs the Route Discovery mechanism by broadcasting Route Request (RReq). Any intermediate node that receives a non-duplicate RReq appends its address to the source route list in the RReq packet and rebroadcast it (Figure 1).

When the destination node receives the packet, it sends a Route Reply (RRep) back to the source node. Further, in the network, the nodes may cache routing information obtained from Route Discovery and data packets. Moreover, if an intermediate node has some requested information, that is the route to destination, in its cache, it may send a RRep back to the source node.

By using this mechanism, the source node obtains several routes to reach a destination. We will explain more precisely this mechanism in the Section III-B.

B. Route Maintenance

In an Ad hoc network, the nodes are mobile, that is why after sending data, we need to make sure that the topology has not changed and that a source node may use a route to reach a destination. Route Maintenance is a succession of three conditional procedures providing the confirmation that a link

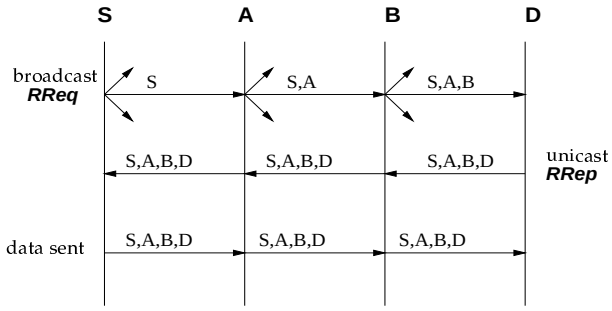

Fig. 1. Route Discovery mechanism of the DSR protocol.

may carry data. First, DSR requests from the link layer to insure the maintenance. This may be provided by an existing standard part of the MAC protocol in use (such as the link-level acknowledgement frame define by IEEE 802.11) [15]. When this layer cannot insure it, the other procedure consists in listening to every packet in the radio range of the node to determine whether the links are still available. This is called a *passive acknowledgement*. Finally, if the two first conditional procedures fail, the confirmation of receipt may be provided using network layer acknowledgments. To do so, the node inserts an Acknowledgement Request option in the DSR Options header of the packet. Therefore, when the link state is denoted as broken, the corresponding node sends a Route Error to the source node.

The Ad hoc networks topologies rapidly change since the intermediate nodes to reach a destination from a source may move and their number may change. All these changes must be taken into account in order the implementations to be conformed to the standard [1]. That is why it is necessary to apply conformance testing steps at the beginning of the protocol development phase. A formal description of the DSR protocol is necessary to ensure reliable implementations using conformance testing methods. This is the subject of the following section.

III. THE DSR PROTOCOL SPECIFICATION

A. The SDL language

The Specification and Description Language SDL standardized by ITU-T [12] is widely used to specify communicating systems and protocols. This language has evolved according to user needs. It provides new concepts needed by designers to specify systems more and more complex. SDL is based on the semantic model of Extended Finite State Machine (EFSM) [16]. Its goal is to specify the behavior of a system from the representation of its functional aspects. The description of the functional aspects is provided at different levels of abstraction. The most abstract level is the one describing the system, while the lowest one is the specification of abstract machines composed of signals, channels, tasks, etc. Two kinds of properties may describe these functional aspects: the architectural and behavioral properties. The first one denotes the architecture of the system, that is the connection and organization of the elements (blocks, processes, etc.) with

the environment and between themselves. The second one describes the behaviors of the entities in terms of their interactions with the environment and among themselves. These interactions are described by tasks, transitions between states, and are based on the EFSMs.

A verification on local variable values in these EFSMs imposes a condition (predicate) on moving to the next state. The actions associated with a transition include: verification on local variable (that can impose conditions, predicates, to move to the next state), the execution of tasks (assignment, statement or informal text), procedure calls, dynamic creation of processes in order to remove or include new mobile nodes into a system for instance, arming and disarming timers, etc. SDL contains the concepts of "type" and "instance of type" that allows to specify the introduction or removal of nodes in the network. SDL also supports objects that permit to define generic types that could be validated and used in different contexts. ASN.1 [17], a standard defined for data transfer, is also supported. Specifically, data are defined as abstract data type.

B. Specification of DSR

In this section, we describe the DSR protocol specification modelled using SDL. The SDL model has been designed in such a way that it is very easy to add, remove and observe functionalities. The specification follows the IETF draft [1] and each message, table, route cache information have been scrupulously respected. Nevertheless we did not specify all the features described in this draft. Indeed, at the end the verification of the whole specification would be very long. Therefore, we have specified the relevant basic and additional features which are:

- Basic and additional Route Discovery features,
- Basic and additional Route Maintenance features (salvage, route shortening, etc.),
- Some conceptual data structures: Route Cache, Send Buffer, RReq and gratuitous RRep tables,
- The DSR options header format.

We did not specify the flow state extension, the security concepts, and how to support multiple interface. The specification of these features is currently being carried out.

Our system includes N blocks denoting the nodes of a network. These N blocks describe the protocol behavior as we will detail further. They are dynamically generated by a block type, that is each node is an instance of this block type. Therefore, we may provide by this manner any number of nodes we wish, and thus a small or a big network. This block type is linked to another block named *Transmission* (see Figure 2) that takes care of transmitting the messages from one node to another.

The role of this previous block is to receive the packets and send them in unicast or broadcast in the network generated by the N nodes. It provides the mobility of each node and manages the topology of the wireless network by opening or closing some links between nodes. Thus, the behavior of each node is modified according to the packets they receive. This system allows to simulate a wireless Ad hoc network efficiently even whenever a very small topological change occurs that leads to a significant change in connectivity.

In our system, each mobile node is represented by three connected processes called *USR*, *IP* and *DSR* as illustrated by Figure 3.

The first process is the one connected to the environment. It receives the necessary information in order to set up the network (number of nodes, topology, etc.) by initializing one or several packets. The process *IP* is in charge of encapsulating the information provided in the packets by the *USR* process, in a new IPv4 packet. While the last process *DSR* represents the behavior of DSR protocol including the two mentioned mechanisms Route Discovery and Route Maintenance. It also provides the modification of the IP datagrams adding a DSR header. Within these nodes, four main data structures have been specified:

- the *Route Cache* which is a table containing every learned route between nodes. In [1], several methods are proposed to define and manage this cache. We have decided to use the following format for each information: $\{Source, Destination, Route\}$.
- the *Send Buffer* which is a list of cached IP packets with DSR headers when a route to a destination is still undiscovered or a link has been broken. Here the IP packet is just stored.
- the *Route Request Table* which contains every Route Request packet information of type $\{Initiator, Target, TTL, noftransmissions, Id\}$.
- the *Maintenance Buffer* that contains all the packets waiting for maintenance.

For all of these structures, we add an information denoting the current size of the cache without assigning maximum sizes. Indeed, each implementation may offer different data structure sizes and that is why we let them unassigned which is left for a real implementation. In this last case, it is very easy to modify the specification to apply maximum sizes.

Through these nodes, different kinds of packets are transmitted by using unidirectional or bidirectional links (the environment of the system will set them at the beginning of a simulation). The *Route Request* (RReq) packet as mentioned in Section II-A, contains a list of addresses updated by each node encountered on the route. The *Route Reply* (RRep) packet is the response to a RReq received by the destination. This packet is sent to the initiator of the RReq using the Route Cache or by piggybacking in a new RReq to the source. The specification also enables to piggyback other small data packets such as TCP SYN packet [18], on a RReq using the same mechanism. The packet *Source Route* (SrcR) is finally used to transmit data once a route is known. All these packets and the others in the network are specified using ASN.1 and carried by SDL signals as illustrated in the following for a SrcR packet.

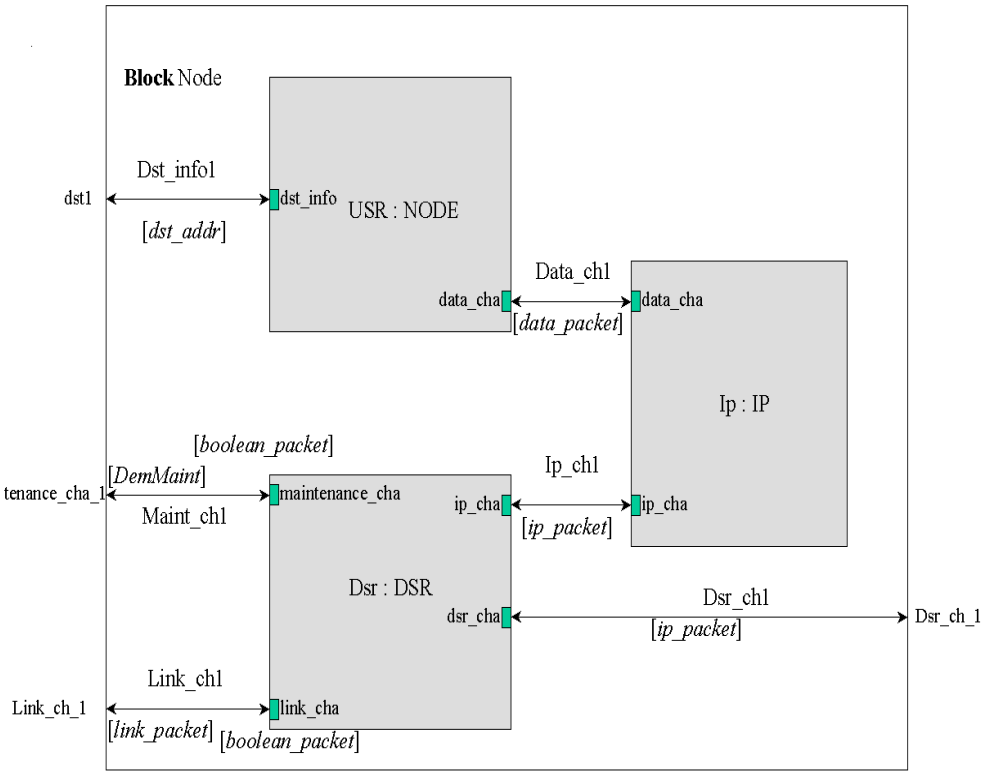

Fig. 2. The DSR protocol system specification.

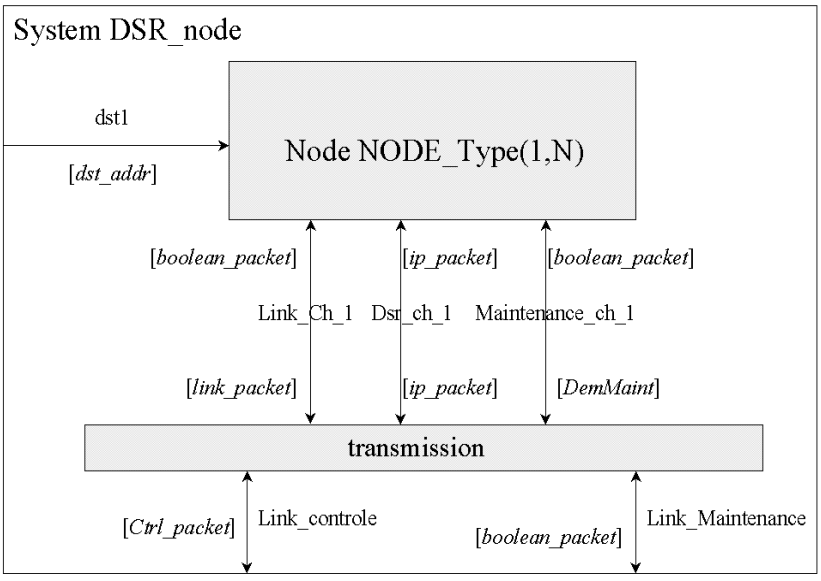

Fig. 3. The node specification in our network.

```
NEWTYPE SourceRoute_T STRUCT
option_type,
data_len Integer;
F,
L Boolean;
salvage,
segs_left Integer;
address AddressList_T;
ENDNEWTYPE;
```

The DSR Source Route option in a DSR options header corresponds to what is mentioned in [1]. The different fields are described as follows:

- `option_type`: it allows a node to drop this packet when it does not understand this option
- `data_len`: it provides the length of the option taking into account the addresses contained in the packet
- `F` for First Hop External, it denotes that the first hop indicated by the DSR Source Route option is an arbitrary path in a network outside the DSR networks
- `L` for Last Hop External, denoting the same that `F` but for the last hop
- `Salvage`: it is used to count how many times the packet has been salvaged
- `segs_left`: it provides the number of route segments remaining
- `address`: it defines a list of addresses for the Source Route.

The DSR signals carry many variables contained in various ASN.1 types, that are built in particular from lists and tables.

In order to describe the whole behavior of our network (using the IP layer), many processes have been specified. The DSR protocol specification is approximately 5000 lines of SDL in textual format. To give a general idea of the complexity of the SDL system specification, we present in Figure 4 some significant metrics of the global system.

Lines	5168
Blocks type	6
Blocks	13
Processes type	12
Processes	6
Procedures	38
States	114
Signals	23
Macro definitions	6
Timers	3

Fig. 4. Metrics of the DSR protocol specification.

This system has been simulated applying our technique we describe in the next section. Our test scenario generation method has been used for the generation of the test scenarios. It is described in the following.

IV. Validation of the DSR protocol

A. Protocol verification

A first step of our validation approach, as illustrated by figure 5, is to check that the protocol specification respects the IETF DSR draft requirements and that it is free of livelocks and deadlocks. This is the reason we have designed some scenarios based on the IETF DSR draft requirements that we have verified on the specification using model checking techniques [13].

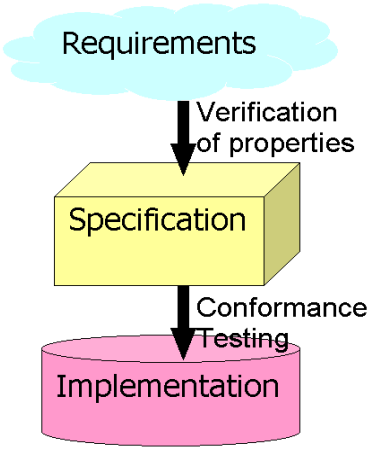

Fig. 5. Validation steps.

This verification step precedes the goal-oriented test generation. We need to make sure that the protocol specification satisfies the requirements. By the analysis of the SDL specification and its reachability graph, we verify that the specification is free from deadlocks and livelocks within the simulated state space. The presence of such deadlocks or livelocks reveals that the DSR protocol system does not behave as expected. Secondly, the behaviors of each process described in the IETF DSR draft have been written using Message Sequence Charts (MSC). These MSCs have been applied on the constructed specification to show that the behaviors represented by the specification are correct with respect to the requirements. It must be noted that the scenarios produced for this verification are just applied on the involved processes whereas the test scenarios that are dynamically generated (see next section) by our tool are much more detailed. Indeed during the simulation for tests generation, all the process instances are taken into account. In particular, this allows to generate each case of interleaving and to validate different orders of events (transitions) that may appear in a real network.

B. Goal-oriented test scenarios generation

In the previous section we have verified the protocol, i.e. checking that the requirements of the standard draft were true on the specification. In this section, we generate a set of detailed scenarios to be applied on the protocol implementation to check its conformance to the specification as shown in the Figure 5.

The main purpose of conformance testing is to determine whether an implementation behaves as indicated in the corresponding specification. Most testing proposals are based on a method called *active testing*. Intuitively, the tester sends an input to the implementation and waits for an output. If the output lies within the expected outcome, according to the specification, the process continues; otherwise, a fault is detected in the implementation. This kind of testing is called *active testing* because the tester has total control over the inputs sent to the implementation under test.

For the DSR protocol, we have applied a method based on active testing. Our main objective is to generate a set of scenarios to test some expected properties of the system implementation. These properties are expressed as test purposes. In order to produce the scenarios we apply a test tool that we have developed at INT. The generation procedure is completely automated and follows these main steps:

- Step 1. Construct a precise and concise *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.). We use the SDL specification of the DSR protocol system described in the previous section.
- Step 2. *Select* the appropriate tests. This selection can be performed according to different criteria. 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 variables' values.
- Step 3. *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 calculate 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 represents a test purpose. This algorithm has been implemented in our tool called TESTGEN-SDL [2].
- Step 4. *Format* the test scenarios. That is, to produce test scenarios in some accepted formalism. In our case, test scenarios are produced in Message Sequence Charts, a formalism widely used in industry to describe messages exchanges [19], and in Tree and Tabular Conformance Notation (TTCN), the ITU-TS standard language used for test specification [20].

C. Test experimentation results

In this section, we present the experimentation results of the application of our method and tool to the DSR protocol system and particularly the Route Discovery mechanism as detailed in the following. We have initialized the specification via the *USR* process in order to obtain the network configuration illustrated in Figure 6.

In order to configure the network, our tool processes the generation by the following three steps used to drive the

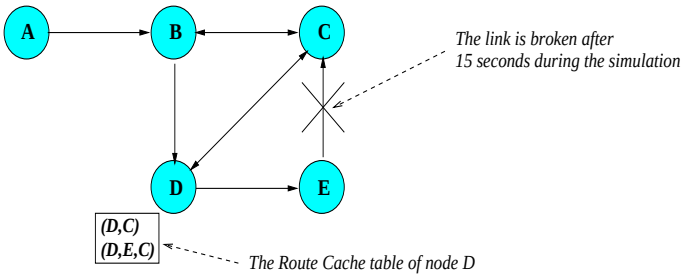

Fig. 6. Network configuration.

simulator. First, it loads the possible inputs to *USR*, the current scenario and the simulation mode (depth-first search, exhaustive, etc.). The configuration file contains some information provided to the *USR* process. It initializes some variables such as the number of initial active nodes and the different possible topologies. Then, the tool identifies the covered transitions, to achieve the test purpose. Finally, the test scenarios are extracted and a partial reachability graph is provided.

Five nodes have been instantiated in which all the Route Caches are empty except for the node D containing the routes (D, C) and (D, E, C). The destinations and the sources from where the data have to be sent are randomly generated.

The component in charge of specifying the Route Discovery aspects contains many processes and therefore many functions are executed in this module. In order to generate test scenarios, we need to define test purposes. We focused here on the following three test purposes that may be provided by a DSR implementation for the route discovery mechanism:

- *Test purpose 1:* To test that the Route Cache table of each instantiated node is consulted whenever a RReq is sent;
- *Test purpose 2:* To test that the Send Buffer of node A is used three times for each contained packet before discarding them;
- *Test purpose 3:* To test that the field *"number of consecutive initiated Route Discoveries for one destination"* of the Route Request Table is updated whenever a node receives a valid RRep.

These three test purposes may reveal some errors in an implementation of a DSR protocol. Indeed, they may detect some errors in the Route Discovery mechanism. Firstly, we want to make sure that the Route Cache is demanded for each Route Request which is crucial for an on-demand routing protocol. Then, we want to make sure that the Send Buffer is properly used and the Route Request Table is correctly updated after receiving a RRep.

Let us note that the test purposes given above are usable for any network configuration. Moreover, we may generalize our current configuration by modifying the connectivity of links between the nodes.

After application of the test generation procedure, a test scenario is produced for each one of the test purposes. The results obtained are illustrated in Figure 7.

Once all the tests purposes have been exercised, we can merge the three scenarios in a single test scenario. Indeed,

Test purpose	Test scenario length	Duration
♯1	36	219s
♯2	11	28s
♯3	14	32s

Fig. 7. Results obtained.

most of the time the produced scenarios come from linked behaviors for a same entity. Therefore, it is often interesting to merge them in order to test this entity with a single scenario. The obtained scenario is of a length of 61 transitions and the execution time for its generation is relatively short (on a Sun Sparc Ultra 5). Let us note that the generated test scenario includes the behavior of other components (such as *Transmission* or *USR*). Notice also that we have generated one test scenario for each test purpose. Figure 8 presents a part of the obtained test scenario and in Figure 9 the corresponding generated MSC (see Step 4 in Section IV-B).

This scenario shows the test of the second test purpose which is in bold in Figure 8. In this scenario we do not detail the `Preamble(1)` which is the scenario allowing to reach the component specifying Route Discovery. We also note that many tasks are carried out between each state. This modifies the values of the variables. Further, with these values, this test scenario may be applied on a real implementation of the DSR protocol in order to find erroneous behaviors of the implemented protocol.

V. Conclusion

This paper has presented a validation model for the DSR protocol. It includes a formal specification of the protocol, a verification technique to verify the formal specification with respect to the requirements, and a method and a tool for the automated generation of test scenarios. This validation model presents several advantages. First, the design of a formal specification that can be verified contributes to the elimination of design errors and ambiguities. And, in particular, this formal model is well adapted for the description of DSR, it takes into account one of the main characteristics of ad hoc networks: nodes can be added and deleted in a dynamic manner. Secondly, the use of test purposes for test generation can be very useful to meet user requirements. Also, automated test generation is less costly that tests written manually, reducing the time to market. And finally, the test scenarios we have generated can be re-used for non functional testing such as system capacity and response time testing.

In addition, the proposed methodology is generic and can be easily applied to other routing protocols for ad hoc networks. It can also be applied to large-scale system and to extended ad hoc networks. As a perspective, we are working on this last point: we are starting to apply our methodology on a real ad hoc network executing a real DSR implementation.

References

[1] D. Johnson, D. Maltz, and Y.-C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (dsr)," Internet-Draft, draft-ietf-manet-dsr-09.txt, April 2003, work in progress.

[2] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaidi, "Hit-or-jump: An algorithm for embedded testing with applications to IN services." in *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99*, ser. IFIP Conference Proceedings, J. Wu, S. T. Chanson, and Q. Gao, Eds., vol. 156. Kluwer, 1999, beijing, China.

[3] C. Adjih, T. Clausen, and P. J. et. al., "Optimized link state routing protocol," Internet-Draft, draft-ietf-manet-olsr-08.txt, March 2003, work in progress.

[4] S. D. C. Perkins, E. Royer, "Ad hoc on demand distance vector (aodv) routing," Internet-Draft, draft-ietf-manet-aodv-13.txt, February 2003, work in progress.

[5] V. Park and S. Corson, "Temporally-ordered routing algorithm (tora) version 1," Internet-Draft, draft-ietf-manet-tora-spec-04.txt, July 2001, work in progress.

[6] Z. Haas, M. Pearlman, and P. Samar, "The zone routing protocol (zrp) for ad hoc networks," Internet-Draft, draft-ietf-manet-zone-zrp-04.txt, July 2002, work in progress.

[7] J. Broch, D. Maltz, and D. Johnson, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, October 1998, pp. 85–97.

[8] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/dmaltz/www/dsr.html.

[9] D. Maltz, J. Broch, and D. Jonhson, "Experiences designing and building a multi-hop wireless ad hoc network testbed," Carnegie Mellon University, Tech. Rep., 1999.

[10] A. Cavalli, B. Chin, and K. Chon, "Testing methods for SDL systems." in *Computer Networks and ISDN Systems.*, vol. 28, 1996, pp. 1669–1683.

[11] D. Rouillard and R. Castanet, "Generate certified test cases by combining theorem proving and reachability analysis," in *Proceeding of IFIP TC6 /14th International Conference on Testing of Communicating Systems TESTCOM 2002*, March 2002, pp. 249–266.

[12] ITU-T, "Recommandation Z.100: CCITT Specification and Description Language (SDL)," ITU-T, Tech. Rep., 1999.

[13] Verilog, *ObjectGEODE Simulator*, 1997.

[14] D. Johnson and D. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, tomasz imielinski and hank korth ed. Kluwer, 1996, ch. 5, pp. 153–181.

[15] I. C. S. L. M. S. Committee, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997, 1999.

[16] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - a Survey," in *The Proceedings of IEEE*, vol. 84, August 1996, pp. 1090–1123.

[17] O. Dubuisson, *ASN.1*. Springer, 1999.

[18] J. Postel, *Tranmission Control Protocol. RFC793*, 1981.

[19] ITU-T, *Recommandation Z.120, Annexe B: Algebraic semantics of Message Sequence Charts*, April 1995, geneva.

[20] E. TTCN-3, *TTCN-3 – Core Language*.

Biography

Ana R. Cavalli received the Doctorat d'Etat es Mathematics Sciences and Informatics in 1984 from the University of Paris VII, France. From 1985 to 1990, she was a staff research member of CNET (Centre National d'Etudes des Telecommunications), where she worked on software engineering and formal description techniques. Since 1990, she joined as professor the National Institute of Telecommunications. Currently, A.R. Cavalli is responsible at INT of the research group "Protocols and Services Validation". She has been chair of several conferences (IFIP FORTE/PSTV, IEEE ICNP, IFIP TESTCOM, IFIP CFIP) and published around 100 papers. Her research interest include formal description techniques and validation of Internet protocols and services, methods for conformance testing and interoperability testing, methodology of network computing, logics and proof methods for distributed systems.

Stephane Maag obtained his PhD in 2002 at the University of Evry (France) in the area of telecommunication systems

```
ri/NULL @ (0,"NULL/started_procs",1).
Preamble(1) @ (1,‘‘dst_addr(105)/data_packet(addr.out)’’,2).
(2, ‘‘data_packet(addr.out)/ip_packet(ip.out)’’,3).
(3, ‘‘ip_packet(ip.out)/DSR_header(ip_packet!information)’’,4).
(4, ‘‘DSR_header(RReq_options)/ip_packet(ip.in),fillsendbuffer(ip.in)’’,5).
(5, ‘‘sendbuffertimeout(30)/DSR_header(ip_packet!information)’’,4)
(5, ‘‘searchsendbuffer(ip_in)/sendbufferdiscard(ip_in)’’,7).
```

Fig. 8. A part of the generated test scenario.

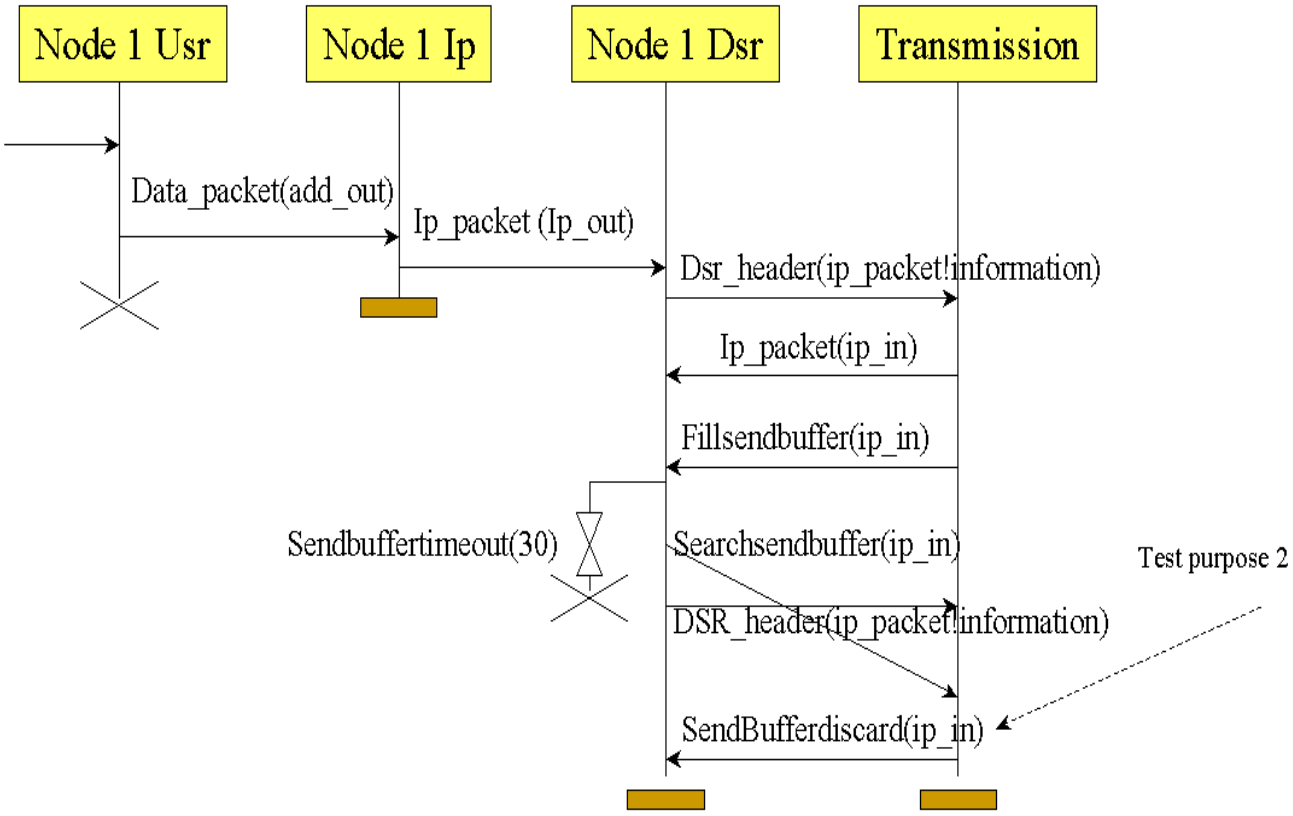

Fig. 9. An MSC from test scenario generation.

validation. Currently, he is working in the Professor A. Cavalli’s research team ”Protocols and Services Validation” at the National Institute of Telecommunications. His current interests are formal description techniques and test generation methods for protocols conformance and interoperability.

Cyril Grepet received his MSs from Evry University, France. Currently, he is a PHD student in the A. Cavalli’s research team ”Protocols and Services Validation” at the National Institute of Telecommunications.