A Validation Model for the DSR protocol

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

***Abstract*— This paper presents a validation model for the Dynamic Source Routing (DSR) protocol. This model includes a formal specification of the protocol and a set of scenarios. The scenarios test the conformance of a given implementation to some targeted system functionalities. The DSR protocol has been specified following the IETF draft [1]. The formal specification has been performed 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 this network, there is no administrative node, and each node participates in order to provide 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. Many researches on routing protocols for ad hoc networks have been led, 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 has many chances to be normalized soon by the IETF. Moreover, due to previous study showing that reactive routing protocols are better suited in ad hoc networks environment than proactive ones [7], some DSR implementations have emerged.

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

Conformance testing becomes a crucial phase of the ad hoc routing protocol design and development. It is needed to develop conformance testing methods for routing protocols. Indeed functional and security failures caused by poor testing may have a catastrophic effect on mobile wireless networks reliability. In the last years, 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 that 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 to generate test scenarios for only those parts. In general, this selection is made by human experts that identify the part of system's behavior or the expected properties that might be subject of testing and formulate test purposes or goals based on this identification [8], [9].

In this paper, we propose a validation model of DSR, based on a formal specification and a goal-oriented testing technique. This work is a contribution to the development of correct DSR implementations.

For the formal specification of the DSR protocol, we use the SDL language [10]. This language is well adapted to the description of the protocol and allows to provide a hierarchical description of the system using different architecture levels, to specify IP, DSR and link layer.

In order to perform goal-oriented testing, we use a method and a tool developed at INT. The method is based on test purposes and the tool allows to generate 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 2 introduces the DSR protocol. The two mechanisms of the protocol, Route Discovery and Route Maintenance are presented in this section. Section 3 presents the SDL specification of DSR based on [1]. In Section 4, the experimentation results of the application of the test procedure to the DSR protocol are presented. Finally, the Section 5 concludes and gives the perspectives of this work.

II. DSR protocol Overview

The *Dynamic Source Routing* protocol (DSR) [11], [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 the *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 protocol DSR. It is in charge to find routes between the nodes into 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 performs the Route Discovery 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. 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.

Finally, the source node obtains several routes to reach the destination. We will explain more precisely this mechanism in the section III-B.

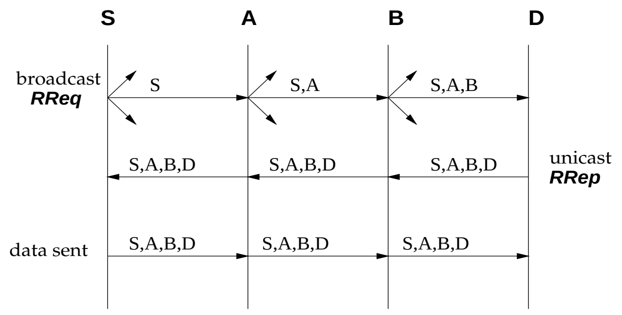

Fig. 1. Route Discovery mechanism of the DSR protocol.

B. Route Maintenance

In an Ad hoc network, the nodes are mobile, that is why before 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 D. Route Maintenance is a succession of three conditional procedures. First, DSR requests to the link layer to insure the maintenance. When this latter can not insure it, the node listen to every packet in its radio range to determine whether the links are still available. Finally, the confirmation of receipt in many cases may easily be provided, either as an existing standard part of the MAC protocol in use (such as the link-level acknowledgement frame define by IEEE 802.11) or by *passive acknowledgement*. Then, 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 be changed. 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. 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 [10] 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) [12]. 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 abstraction levels. The most abstract is the one describing the system, while the lowest is the specification of abstract machines composed by 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.). The second one describes the behaviors of the entities after an interaction with the environment. These reactions are described by tasks, transitions between states, and are based on the EFSMs.

A verification on local variable values 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 or informal text), procedure calls, dynamic creation of processes in order to include new mobile nodes into a system for instance (SDL contains the concepts of "type" and "instance of type"), arming and disarming timers, etc. SDL supports objects that permit to define generic types that could be validated and used in different contexts. It also supports ASN.1 [13], a standard defined for data transfer. 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 messages, tables, route caches 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 such as:

- 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.

Besides we did not specify the flow state extension, the security concepts, and how to support multiple interface. It is currently carried out as the future works.

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 then as well 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.

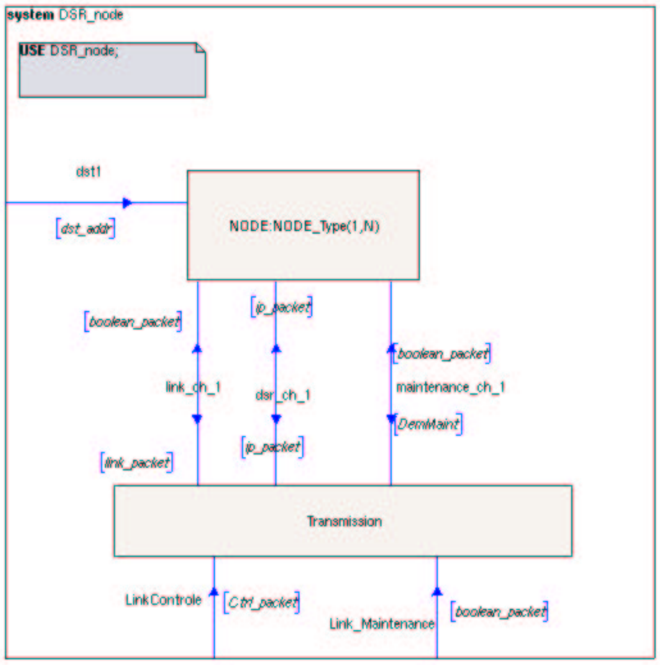

Fig. 2. The DSR protocol system specification.

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 manage the topology of the wireless network by opening or closing some links between any nodes. This system allows to simulate a wireless Ad hoc network efficiently even whenever a very small change in position 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 the Figure 3.

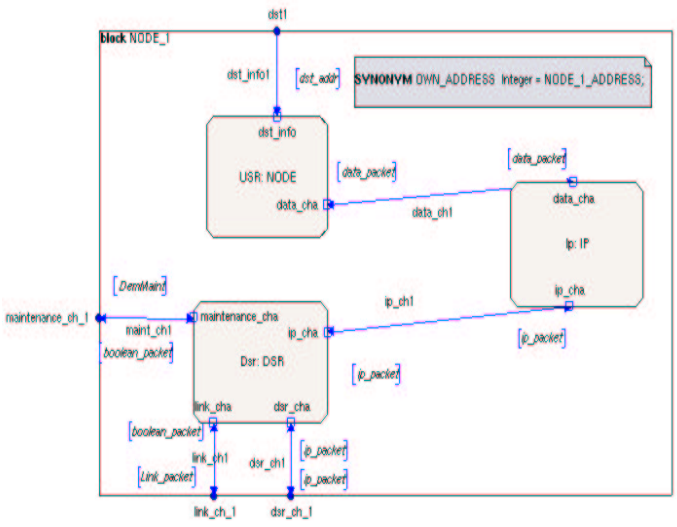

Fig. 3. The node specification in our network.

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 IP packet. While the last process *DSR* represents the behavior of DSR protocol including the two mentioned mechanisms Route Discovery and Route Maintenance. Within these nodes, three main data structures have been specified:

- the *Route Cache* that is a table containing every learned route between nodes,
- the *Send Buffer* that 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,
- the *Route Request Table* that contains every Route Request packet information.

Through these nodes, different kind 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 II-A, contains a list of addresses updated by each met node. The *Route Reply* (RRep) packet is the response to a RReq arrived to 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, 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 carried by SDL signals as illustrated in the following for a SrcR packet.

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

We do not detail the different fields composing the Source Route packet. However it allows to see that the 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 made of approximately 5000 lines of SDL. To give a general idea of the complexity of the SDL system specification, we present by the 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.

We have to note that such a specification may contain few errors during its design even from the requirements. For this reason we have used model checking techniques [14] in order to verify our specification concerning the three purposes mentioned in the next section. Indeed, before validating an implementation we need to make sure that the used specification corresponds to the requirements.

Therefore, In order to simulate the network, we use a configuration file containing some information provided to the *USR* process. It initializes some variables such as the number of active nodes and the different possible topologies. First in analyzing the EFSM and its reachability graph, we verify that the specification is free from deadlocks and livelocks within the simulated space. Indeed, the presence of such deadlocks or livelocks reveals that the DSR protocol system does not behave as expected.
For the generation of the test scenarios, we have used a test scenario generation method described in the following section.

IV. The test of the DSR protocol

A. Test scenarios generation

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 the following main steps:

- Step 1. To obtain 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 (test architecture, test interface, etc.). We use the SDL specification of the DSR protocol system described in the previous section.
- Step 2. To *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 or the behavior of a specific component of the system (for instance node behavior).
- Step 3. To *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 applied to the implementation under test, verifies 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. To *format* the tests. That is, to produce test scenarios in some accepted formalism. In our case, test scenarios are produced in Message Sequence Charts (MSC), a formalism widely used in industry to describe processes messages exchanges, and in Tree and Tabular Conformance Notation (TTCN), the ITU-TS standard language used for test specification [15].

B. 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 by the Figure 5.

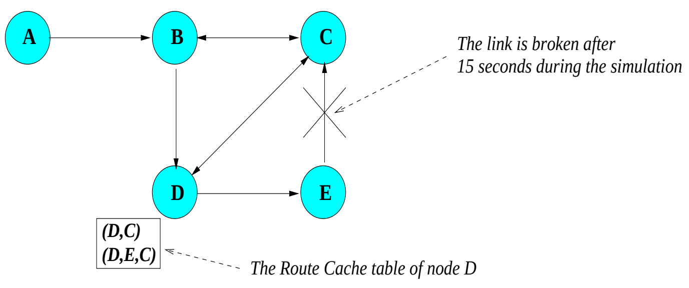
A
B
C
D
E
The link is broken after
15 seconds during the simulation
(D,C)
(D,E,C)
The Route Cache table of node D

Fig. 5. Network configuration.

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 crucial errors into an implementation of a DSR protocol. Indeed, they allow to detect some errors during 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. Therefore, that the Send Buffer is properly used and the Route Request Table correctly updated after receiving a RRep.
Let us notice that the test purposes given above are usable for any network configuration. Moreover, we may generalize our current configuration by modifying the connectivity of every link 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 the Figure 6.

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

Fig. 6. Results obtained.

Once all the tests purposes have been exercised, we can merge them in a single test 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 7 presents a part of the obtained test scenario and in Figure 8 the corresponding generated MSC (see Step 4 in section IV-A).

This scenario shows the test of the second test purpose which is in bold in the Figure 7. The scenario is generated until the two other test purposes are achieved. 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 method and a tool for the automated test generation of scenarios. This validation model presents several advantages. First, the design of a formal specification from which tests are generated contributes to eliminate 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 eliminated in a dynamic way. 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. And the proposed methodology can be applied to large-scale system and be easily applied to extended ad hoc networks. As a perspective, we are working on this last point, that is we are starting to apply our methodology on a real ad hoc network executing a real DSR implementation. Also, we are currently specifying the DSR features that were not in our work such as flow state extensions, etc.

References

[1] D.B. Johnson, D.A. 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*, Jianping Wu, Samuel T. Chanson, and Qiang Gao, Eds. 1999, vol. 156 of *IFIP Conference Proceedings*, Kluwer, Beijing, China.
[3] C. Adjih, T. Clausen, and P. Jacquet et. al., "Optimized link state routing protocol," Internet-Draft, draft-ietf-manet-olsr-08.txt, March 2003, Work in progress.
[4] S. Das 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.J. Haas, M.R. 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.A. Maltz, and D.B. 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] A. Cavalli, B. Chin, and K. Chon, "Testing methods for SDL systems.," in *Computer Networks and ISDN Systems.*, 1996, vol. 28, pp. 1669–1683.
[9] 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.
[10] ITU-T, "Recommandation Z.100: CCITT Specification and Description Language (SDL)," Tech. Rep., ITU-T, 1999.
[11] D.B. Johnson and D.A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, chapter 5, pp. 153–181, Kluwer, tomasz imielinski and hank korth edition, 1996.
[12] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - a Survey," in *The Proceedings of IEEE*, August 1996, vol. 84, pp. 1090–1123.
[13] O. Dubuisson, *ASN.1*, Springer, 1999.
[14] Verilog, *ObjectGEODE Simulator*, 1997.
[15] ETSI. TTCN-3, *TTCN-3 – Core Language*.

```
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. 7. A part of the generated test scenario.

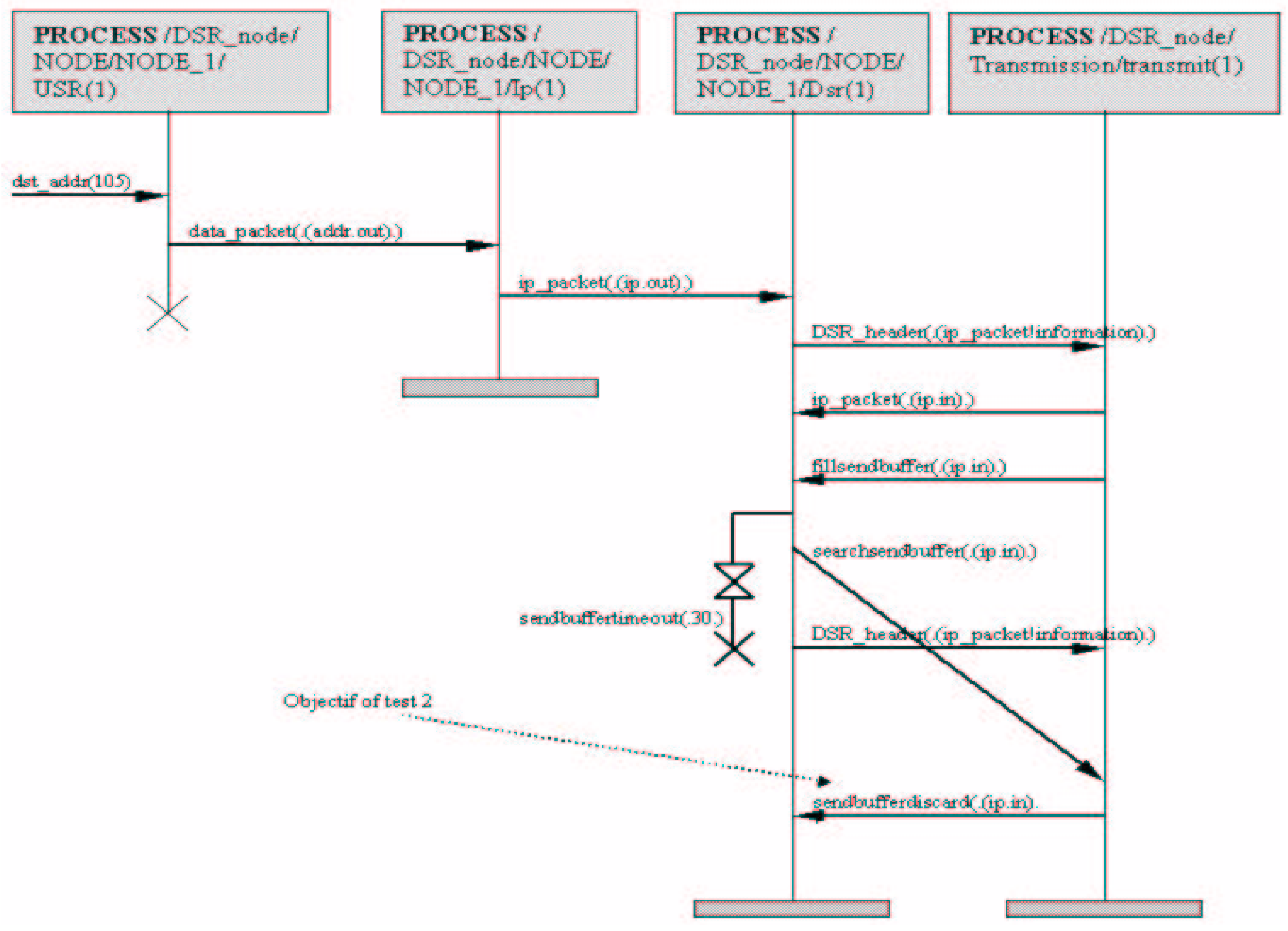
PROCESS /DSR_node/NODE/NODE_1/USR(1)
PROCESS /DSR_node/NODE/NODE_1/Ip(1)
PROCESS /DSR_node/NODE/NODE_1/Dsr(1)
PROCESS /DSR_node/Transmission/transmit(1)
dst_addr(105)
data_packet(.(addr.out).)
ip_packet(.(ip.out).)
DSR_header(.(ip_packet!information).)
ip_packet(.(ip.in).)
fillsendbuffer(.(ip.in).)
searchsendbuffer(.(ip.in).)
sendbuffertimeout(30)
DSR_header(.(ip_packet!information).)
Objectif of test 2
sendbufferdiscard(.(ip.in).

Fig. 8. An MSC from test scenario generation.