Un Modèle de validation pour le protocole de routage DSR

S. Maag — C. Grepet — A. Cavalli

Institut National des Télécommunications
CNRS UMR 5157
9, rue Charles Fourier, F-91011 Evry Cedex
{Stephane.Maag, Cyril.Grepet, Ana.Cavalli}@int-evry.fr

RÉSUMÉ. *Nous proposons dans cet article une spécification formelle du protocole de routage Dynamic Source Routing (DSR) pour les réseaux ad hoc, ainsi qu'un modèle de validation pour ce protocole. Le protocole DSR a été spécifié en SDL en suivant le cahier des charges présenté dans le draft IETF [JOH 03]. La méthode de spécification utilisée pour DSR répond bien aux contraintes posées par les réseaux sans fil. Une technique de validation pour le test de conformité développée à l'INT est alors appliquée sur la spécification. Celle-ci est basée sur une technique d'objectifs de test qui expriment les propriétés spécifiques du système. Elle permet de générer automatiquement des scénarios de test sans nécessiter une expertise importante du langage et du modèle utilisé. Cette méthode peut notamment être utilisée sur d'autres protocoles de routage dans les réseaux ad hoc. Des résultats expérimentaux sur la spécification sont de surcroît illustrés.*

ABSTRACT. *We propose in this article a formal specification of the routing protocol Dynamic Source Routing (DSR) for the wireless ad hoc networks, as well a validation model for this protocol. The protocol DSR is specified using SDL according to the requirements provided by IETF [JOH 03]. The specification method used for DSR meets the requirements needed for the wireless networks. A validation technique for conformance testing that has been developed at INT is then applied on the specification. This latter is goal-oriented and expresses the specific properties of the system. It allows to automatically generate test scenarios without needing to be an expert using the language or the model proposed. This technique may also be used for other wireless ad hoc routing protocols. Furthermore, some experimental results on the specification are given.*

MOTS-CLÉS : *réseaux sans fil ad hoc, MANET, protocole de routage, DSR, test de conformité, SDL*

KEYWORDS: *wireless adhoc networks, MANET, routing protocol, DSR, conformance testing, SDL*

e soumission à *CFIP 2005*, le 18 janvier 2005.

1. Introduction

Un réseau sans fil Ad hoc (*Mobile ad hoc Network* i.e MANet) est une collection de noeuds mobiles qui communiquent les uns avec les autres sans avoir recours à une infrastructure préexistante. Dans ce type de réseau il n'y a donc pas d'autorité fédératrice, ni de hiérarchisation des noeuds (par exemple maître/esclave) et chaque noeud participe activement à la bonne transmission des données dans le réseau. Les noeuds étant mobiles, ils peuvent à tout moment quitter ou joindre le réseau. De fait, à mesure que les connexions entre les noeuds se créent et se détruisent, la topologie du réseau peut rapidement être modifiée. Par ailleurs, chaque mobile utilise les ondes radio avec une portée limitée comme medium de communication. Aussi, certains noeuds se comportent comme des routeurs de manière à transférer les données d'une source à une destination.

Les utilisations des réseaux ad hoc sont nombreuses et variées. Des scénarios à des fins militaires, d'opérations de secours lors de catastrophes naturelles, ou encore dans des réseaux de capteurs sont aujourd'hui avancés. Du fait de ces différents aspects et des ressources limitées des noeuds mobiles, un routage efficace est crucial dans les réseaux ad hoc. A partir des caractéristiques peu communes de ces réseaux, de nombreux pré-requis ont été définis pour la conception d'un protocole de routage. Ainsi, de nombreuses propositions ont été faites [WIK 04] et elles se déclinent en trois catégories principales : les protocoles de type *proactifs* [CLA 03], *réactifs* [JOH 03, PER 03] et *hybrides* [V.D 01, HAA 02]. L'un d'entre eux, *Dynamic Source Routing* (DSR)[JOH 03] est un protocole réactif simple et efficace permettant au réseau de se gérer et de se configurer sans l'aide d'une infrastructure ou d'une autorité fédératrice. C'est sur ce protocole candidat à la normalisation et proposé par le groupe de travail MANet de l'IETF [IET 04] que nous orientons nos travaux. De plus, certaines études laissent à penser que les protocoles de routage réactifs sont plus adaptés et plus efficaces que les protocoles proactifs pour résoudre les problèmes de routage des réseaux ad hoc [BRO 98]. Aussi plusieurs implantations de DSR ont été développées. L'un des constats fut alors les difficultés rencontrées pour obtenir un réseau performant. En effet, les implantations ne semblaient pas supporter un nombre important de noeuds et ne respectaient pas toujours les propriétés mentionnées dans le cahier des charges. Il existe plusieurs raisons à cela. Tout d'abord, il est vrai qu'il est difficile d'implanter de tels protocoles car cela est long et difficile, nécessite du matériel pour réaliser le développement et assurer que toutes les fonctionnalités du draft du RFC de l'IETF ont correctement été implantées. Ensuite, dans la majorité des travaux préliminaires, la validation est réalisée par simulation. Aussi, les conditions réelles d'un cas d'étude ne sont pas rencontrées et ne sont pas basées sur un modèle formel du protocole étudié. C'est pourquoi le test de conformité devient une phase cruciale à la création et au développement d'un protocole de routage ad hoc.

Dans la suite de cet article, et notamment dans la Section 2, nous mentionnons les travaux existants pour le test de tels protocoles et soulignons l'originalité de notre contribution. En Section 3, nous introduisons les concepts du protocole DSR. Puis en Section 4, nous présentons notre spécification SDL de DSR basée sur le draft IETF de

ce protocole [JOH 03]. Ensuite nous illustrons les résultats de notre expérimentation par l'application de notre technique de test à la spécification de DSR. Enfin, nous concluons dans la Section 6 et donnons les perspectives de notre travail.

2. Travaux existants

Le test de conformité des protocoles de routage ad hoc est crucial au bon fonctionnement de tels réseaux. Mais paradoxalement, très peu de travaux existent aujourd'hui sur l'établissement de modèles formels de validation de protocole de routage pour les réseaux ad hoc [OBR 02]. Dans [YI 03] et [BAE 01] deux protocoles de routage sont validés en utilisant un testbed et en mesurant les performances du réseau. Mais de nombreuses contraintes sont appliquées et aucun modèle formel n'est utilisé.
Dans [LIN 03], un modèle formel pour la validation de tels protocoles, nommé RNS, est étudié. Celui-ci est bien adapté au test de conformité mais présente deux inconvénients. Le premier est que seuls des experts du langage proposé peuvent établir des tests, ce qui ne facilite pas le tests de nouveaux protocoles. De surcroît, leur modèle de validation ne permet que de tester des métriques du protocole étudié (contrôle de l'overhead, dépenses énergétiques, etc.) et non des fonctionnalités de base (bonne réception du Route Reply, etc.). Ceci réduit les résultats à du test en isolement sans prendre en compte l'interopérabilité d'un noeud avec ses voisins.
Un modèle formel utilisant des automates abstraits distribués permet de spécifier le protocole LTLS [U.G 03]. Cependant aucune analyse fonctionnelle du protocole n'est réalisée et la possibilité de tester une implantation à partir du modèle n'est pas envisagée. Nous présentons de fait dans ce document une méthode de validation du protocole DSR à partir d'un modèle formel. Celle-ci a l'avantage de permettre à des utilisateurs non experts du protocole de générer les tests, mais aussi de valider une implantation à partir d'une spécification formelle sans devoir utiliser un simulateur tel que NS2 [ISI] ou de devoir mettre en place un réseau couteux à la topologie variable.

La génération de nos tests est basée sur une technique d'objectifs de tests que nous avons développée. Elle consiste en la sélection d'une propriété spécifique du système et en la génération des scénarios de test pour cette propriété [CAV 96, ROU 02]. Notre méthode est basée sur une spécification formelle de notre protocole. Cette dernière est développée en utilisant le langage SDL [ITU 99]. Ce langage largement utilisé en industrie est bien adapté à la spécification de protocoles et autorise une description hiérarchique du système en utilisant différents niveaux dans l'architecture (les couches IP, DSR, et liaison de données en ce qui nous concerne).

3. Le Protocol DSR

Le protocole Dynamic Source Routing (DSR) [JOH 03, JOH 96] est un protocole de routage réactif unicast, simple et efficace, dédié aux réseaux Ad Hoc mobile multisauts. DSR est composé principalement de deux mécanismes : *Route Discovery* et *Route Maintenance*. Le premier permet de déterminer automatiquement les routes nécessaires à la communication entre noeuds, tandis que le second permet de s'assurer de la correction des routes tout au long de leur utilisation. Nous allons décrire ces deux mécanismes dans les sous sections suivantes.

3.1. *Le mécanisme Route Discovery*

Le *Route Discovery* a pour but de trouver, au sein du réseau, les routes entre les noeuds désirant communiquer. Ainsi, DSR étant un protocole réactif, un noeud source *S* va rechercher une route uniquement s'il veut émettre un paquet vers un noeud destinataire *D*, et qu'il ne possède aucune route vers celui-ci dans son cache. Pour trouver la route depuis *S*, DSR initie un *Route Discovery* en émettant un paquet en diffusion (broadcast) d'en-tête *Route Request* (RReq), qui va inonder le réseau. Chaque noeud intermédiaire entre *S* et *D* qui reçoit un RReq non dupliqué va concaténer son adresse à la liste contenue dans le RReq et le diffuser à son tour.

Quand le noeud destinataire *D* reçoit le paquet, il retourne à la source un paquet d'en-tête *Route Reply* (RRep). En outre, dans le réseau, les noeuds peuvent enregistrer dans leur cache des informations de routage obtenues au travers des différents paquets *Route Discovery* reçus et des paquets de données. De plus, si un noeud intermédiaire qui recoit un RReq possède en cache une route vers la destination *D*, alors il envoie un RRep à *S* en ajoutant la route connue. Finalement, le noeud source obtient plusieurs routes pour atteindre le destinataire. Une fois ces routes connues, le noeud va pouvoir envoyer des paquets d'option *SourceRoute* (SrcR) contenant les données à échanger.

3.2. *Le mécanisme Route Maintenance*

Dans un réseaux Ad Hoc, les noeuds étant mobiles, il faut vérifier, après l'envoi d'une donnée, que la topologie est toujours la même et que la source *S* peut utiliser une route pour atteindre la destination *D*. Pour ce faire, DSR utilise le mécanisme de *Route Maintenance* qui est une succession de trois procédures conditionnelles. Tout d'abord DSR interroge la couche liaison de données pour savoir si elle assure la maintenance des liens. Si cette dernière ne le fait pas, DSR va écouter tous les paquets dans sa portée radio. Chaque paquet est examiné pour savoir si c'est le paquet retransmis par le noeud suivant ou un autre paquet. Dans le premier cas cela signifie que le lien était valide. Dans de nombreux cas, la confirmation de la réception d'un paquet peut être obtenue soit en utilisant les fonctionnalités de la couche MAC (le standard IEEE 802.11 fournit cette capacité) soit par un acquittement passif. Si finalement les deux premiers tests échouent, alors DSR va réémettre le paquet originel en y ajoutant une option de demande d'acquittement ou *Acknowledgment Request* (AckReq) auquel le noeud suivant devra répondre par un paquet d'acquittement (Ack). En cas d'échec total du *Route Maintenance*, le noeud détectant la rupture du lien mettra à jour son cache de route et enverra un paquet de type *Route Error* (RErr) en direction de la source. Celle-ci pourra choisir une nouvelle route ou recommencer une procédure de *Route Discovery* le cas échéant.

Les aspects susmentionnés ainsi que les spécificités de DSR doivent être correctement spécifiés afin de pouvoir ultérieurement valider une implantation de DSR. Cependant, dès que nous souhaitons modéliser un protocole de communication d'un réseau sans fil, quelques difficultés apparaissent. Nous allons de fait présenter dans ce qui suit notre spécification ainsi que les contraintes et les concepts utilisés.

4. Spécification du protocol DSR

4.1. *Le langage SDL*

Le langage de spécification et de description (SDL pour *Specification and Description Language*) standardisé par l'ITU-T [ITU 99] est très largement utilisé pour spécifier des systèmes et des protocoles de communication. Ce langage a évolué pour répondre aux attentes des utilisateurs et fournir les concepts nécessaires à la spécification de systèmes de plus en plus complexes. SDL est basé sur le modèle sémantique des machines d'états finies étendues (EFSM pour *Extended Finite State Machine*) [LEE 96]. Leur but est de spécifier le comportement de systèmes par la représentation de leurs aspects fonctionnels avec différents niveaux d'abstraction : le niveau le plus abstrait décrit le système et le plus bas spécifie la machine abstraite composée de signaux, de tâches et de canaux de communication. Grâce à ce langage nous pouvons décrire deux types de propriétés : les architecturales et les comportementales. Les premières dénotent l'architecture du système que sont les connexions et l'organisation en éléments de l'architecture (bloc, processus, etc...). Les secondes décrivent le comportement des différentes entités après une interaction avec l'environnement. Ces réactions sont décrites par différentes tâches et transitions entre états, et reposent directement sur les EFSMs.

Une vérification d'une variable locale impose une condition au mouvement vers l'état suivant, ou transition, autrement appelée prédicat. Chaque transition est associée à différentes actions possibles : vérifications de variables locales (qui peuvent imposer un prédicat), exécutions de tâches (attribution de valeurs ou textes informels), appels de procédures, créations dynamiques de processus dans le but d'inclure une nouvelle instance d'un noeud mobile (SDL dispose des concepts de "*type*" et d'"*instance de type*"), déclenchement ou annulation des horloges, etc. SDL supporte des objets qui permettent de définir des types génériques, pouvant être validés et utilisés dans différents contextes, ainsi que la notation ASN.1 [DUB 99] de transfert de données. Les données, ici, sont définies comme un type de données abstraites.

4.2. *La spécification de DSR*

La spécification SDL a été réalisée de façon à ce qu'il soit aisé d'ajouter, de supprimer et d'observer des fonctionnalités. Notre modèle est basé sur le draft IETF de DSR [JOH 03] et la spécification des messages, des tables, et des caches a été scrupuleusement respectée. Nous avons donc spécifié les composantes basiques et additionnelles suivantes :

– Les composantes basiques du *Route Discovery* ainsi que le *Cached Route Reply*

– Les composantes complètes du *Route Maintenance*

– Des structures de données conceptuelles indispensables : *Route Cache, Send Buffer, Route Request Table* et *Maintenance Buffer*

– Le formatage des en-têtes DSR

Il est à noter que nous n'avons actuellement pas spécifié le *flow state extension*, les concepts de sécurité et le support de plusieurs interfaces. En effet, nos deux objectifs principaux sont l'identification des fonctionnalités pertinentes d'un protocole pour le routage dans un réseau sans fils ad hoc, ainsi que la génération automatique de scénarios de test. C'est pourquoi nous nous sommes limités à des composants "de base". Mais nous envisageons par la suite de spécifier complètement DSR.

Les questions qui se sont soulevées lors de la spécification du protocole sont de trois types.

– Comment modéliser le broadcast ?

– Comment modéliser la connectivité ?

– Comment modéliser le changement dynamique de la topologie du réseau ?

Nous répondons à ces questions dans ce qui suit.

4.2.1. *Broadcast*

Du à la nature des communications radio, le broadcast est largement utilisé dans les réseaux ad hoc. Dans notre spécification, le broadcast est spécifié en envoyant les paquets en unicast à tous les noeuds. Une matrice est utilisée dans notre configuration de spécification (cf. 4.2.2 ci-dessous). Un processus nommé *Transmission* a été créé dans cet perspective.

4.2.2. *Connectivité*

Les communications entre les noeuds dans notre spécification sont spécifiées à l'aide d'une matrice dynamique (le nombre de noeuds varie), symétrique de taille $N \times N$ où N est le nombre de noeuds de notre réseau. La matrice est symétrique car nous supposons que tous les liens sont bidirectionnels. Par ailleurs, pour réduire la complexité, nous prenons comme hypothèse que les noeuds sont connectés ou pas. Nous ne modélisons pas d'états intermédiaires comme cela pourrait être le cas avec un modèle réel de couches liaisons/physique.

4.2.3. *Topologie*

La topologie est modifiée en changeant de matrice de connectivité soit manuellement soit de manière aléatoire.

Dans notre système, chaque noeud est représenté par une instance d'un *bloc type* générée dynamiquent et ayant un unique PID (Process IDentifier). Chaque noeud est donc une description du comportement du protocole comme nous le détaillerons plus loin. Avec cette spécification dynamique, nous pouvons génerer autant de noeuds que nous le désirons, et donc avoir un petit ou un gros réseau. Ce *block type* est relié à un autre bloc, nommé *Transmission*, qui assure la communication entre les différents noeuds (cf. Figure 1) ; *Transmission* simule le comportement d'une couche liaison de données (cf. 4.2.2 ci-dessus).

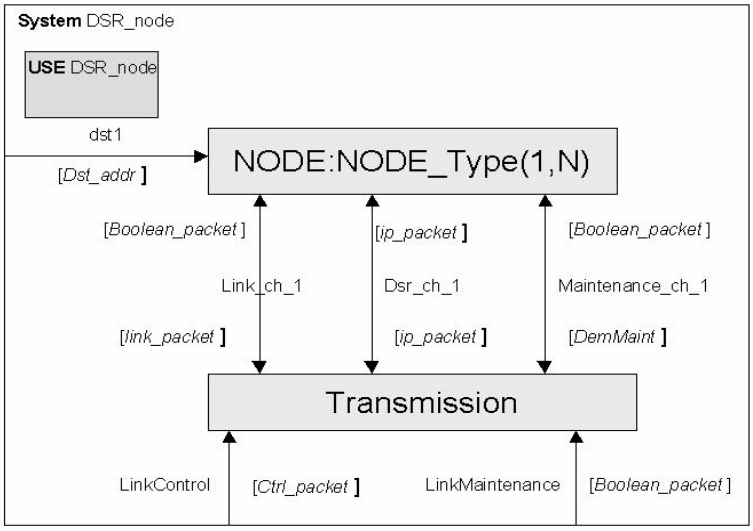

Figure 1. *Spécification du système*

Ce bloc reçoit les paquets venant d'un noeud et assure la transmission, soit en unicast soit en broadcast, de ceux-ci vers un ou plusieurs noeuds du réseau constitué de nos N noeuds. La représentation de la mobilité, et donc des changements de topologie, est assurée par l'ouverture ou la fermeture de connexion dans le bloc *Transmission*, supportée par une matrice de connectivité.
Dans notre système un noeud est donc représenté par trois processus interconnectés que nous avons nommé USR, IP et DSR comme le montre la Figure 2.

Le premier processus est connecté à l'environnement et reçoit les informations nécessaires à la mise en oeuvre et à la configuration du réseau, comme le nombre de noeuds à générer, la matrice de connectivité, ou l'initialisation d'un ou plusieurs paquets. De plus, USR reçoit de l'utilisateur les paquets de données contenant l'adresse du destinataire. Le processus IP a pour seule tâche d'encapsuler les paquets provenant d'USR dans un paquet de type IP, et de décapsuler ceux provenant de DSR. Le dernier processus, DSR, est le coeur de notre spécification puisqu'il décrit le comportement du protocole. DSR comprend donc les mécanismes de *Route Discovery* et de *Route Maintenance* ainsi que des structures de données, telles que celles présentées ci-dessus, que nous allons maintenant détailler.

– *Route Cache* : contient toutes les routes connues par le noeud, et non encore invalidées par un *Route Error*

– *Send Buffer* : liste de trames IP avec un en-tête DSR qui n'ont pas encore été envoyées, soit parce que le noeud attend de découvrir une route, soit parce qu'un lien

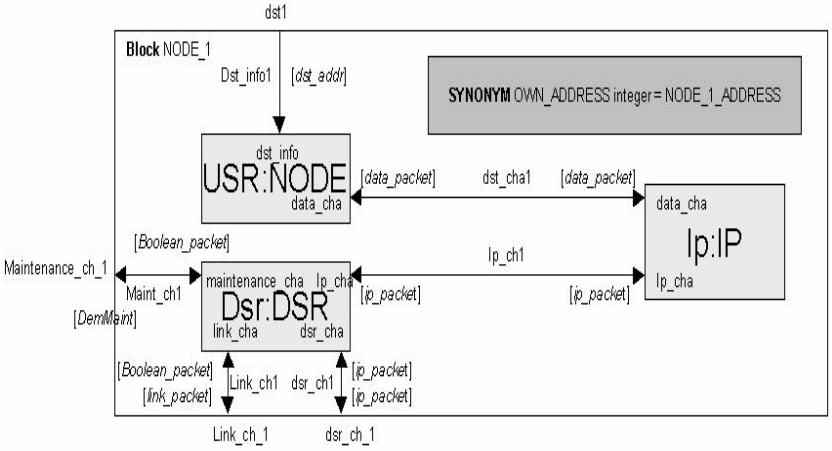

Figure 2. *Spécification d'un noeud du réseau*

a disparu.

– *Route Request Table* : contient toutes les informations sur les paquets *Route Request* émis et en attente de retour.

– *Maintenance Buffer* : contient tous les paquets émis par le noeud en attente de maintenance.

Nous avons aussi défini les en-têtes DSR qui viennent s'incorporer au sein de la trame IP ; par exemple pour le SrcR qui transporte les données, en syntaxe ASN.1 :

```
NEWTYPE SourceRoute_T STRUCT
Option_type, data_len Integer ;
F, L Boolean ;
Salvage, segs_left Integer ;
Address AddressList_T ;
ENDNEWTYPE ;
```

Où le champ `Address` est une liste d'addresses IP dénotant la route que devra suivre le paquet. Il est à noter que bien que nous n'ayons pas encore spécifié la totalité du protocole, les en-têtes sont complets. Pour décrire le comportement d'un réseau ad hoc généré par notre spécification, nous avons spécifié de nombreux processus. Notre spécification est composée actuellement de plus de 11000 lignes. Pour donner une idée de la complexité de celle-ci, nous présentons quelques métriques significatives du système global dans la Figure 3.

Pour ce qui est de la simulation du réseau, nous utilisons des fichiers de configuration contenant les informations transmises à USR, qui initialisent le réseau : nombre

Lignes	11444
Agents totaux	66
Agents de Route Maintenance	14
Agents de Route Discovery	20
horloges	3
Blocs	13 dont 6 Block Type
Processes	56 dont 3 Process Type
Procédures	42
Etats	152
Signaux	23
Macros	3

Figure 3. *Métriques de la spécification de DSR*

de noeuds, adresses des noeuds, topologie, etc. Ensuite en analysant l'EFSM et son graphe d'accessibilité, nous vérifions que la spécification est exempte de deadlock ou de livelock représentant un blocage ou un comportement cyclique dans le réseau. Une spécification d'une telle taille peut contenir certaines erreurs durant sa réalisation et, de ce fait, nous avons utilisé des méthodes de model checking [VER 97] (cf. Section 5.3) pour vérifier la correction de notre spécification sur les parties que nous voulions tester par la suite. Les propriétés testées et la méthode de génération des scénarios de test sont présentées dans la section suivante.

5. Génération automatique de séquences de test

5.1. *Génération de séquences de test*

Notre principal objectif est de générer des jeux de scénarios de test pour tester certaines propriétés attendues du système qui sera implanté ; elles sont exprimées comme des objectifs de test. Pour produire ces scénarios nous utilisons un outil de test développé au sein de l'INT. La procédure de génération est complètement automatisée et suit les étapes suivantes en s'inspirant de [KER 99] :

– Etape préliminaire. Obtenir une spécification formelle du système à tester. Celle-ci doit prendre en compte autant les fonctionnalités du système que les données spécifiques nécessaires au test (architecture de test, interface de test,etc...). Pour cette étape, nous utilisons la spécification du protocole DSR décrite dans la section précédente.

– Etape 1. Sélectionner les tests appropriés. Cette sélection peut s'effectuer suivant différents critères. Cette étape correspond à la définition des objectifs de test : un objectif de test peut être une propriété spécifique du système ou le comportement d'un composant donné du système (pour nous, ce sera le comportement d'un noeud).

– Etape 2. Générer les scénarios de test. Les objectifs de test sont utilisés comme guide par un algorithme basé sur la simulation pour produire des scénarios de test sans nécessiter la génération du graphe d'accessibilité global. Notre algorithme produit

un scénario de test qui, appliqué à l'implantation sous test, va vérifier l'objectif de test. On définit un scénario comme une séquence d'interactions entre le système et l'environnement, qui inclut celle représentant notre objectif de test. Cet algorithme a été implanté dans un outil nommé TESTGEN-SDL [CAV 99].

– Etape 3. Formater les tests. Cette étape est nécessaire pour produire un test dans un formalisme acceptable. Ici nos scénarios de test sont mis au format *Message Sequence Charts* (MSC) [ITU 95], un formalisme largement utilisé par les industriels pour décrire un processus d'échange de message, et au format TTCN (*Tree and Tabular Combined Notation*), le standard ITU-T utilisé pour la spécification de test [ETS 03].

5.2. *Résultats expérimentaux : test du Route Discovery*

Dans cette section nous présentons les résultats expérimentaux de l'application de notre méthode et de nos outils au système du protocole DSR, et plus particulièrement au mécanisme de *Route Discovery* comme nous allons le montrer ci-après. Nous avons initialisé notre spécification via USR de façon à obtenir la configuration de réseau représentée par la Figure 4.

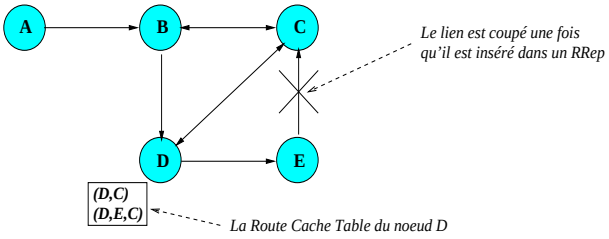

Figure 4. *Configuration du réseau test*

Nous instancions cinq noeuds dont le *Route Cache* est vide, exception faite de celui du noeud *D* contenant les routes (*D,C*) et (*D,E,C*). Les sources et destinations des différents paquets qui vont parcourir le réseau ont été générées aléatoirement. Le composant en charge de la spécification de *Route Discovery* contient de multiples processus et donc de nombreuses fonctions sont exécutées dans ce module. Dans le but de générer des scénarios de test, et comme expliqué dans la section précédente, nous devons définir des objectifs de test. Nous nous sommes focalisés sur la définition de trois objectifs du *Route Discovery* qui doivent être absolument atteints dans une implantation de DSR. Voici ces trois objectifs :

– Objectif 1 : Tester que le *Route Cache* est consulté dans chaque noeud instancié avant que celui-ci envoie un RReq.

– Objectif 2 : Tester qu'avant d'écarter un paquet du *Send Buffer*, il est bien utilisé trois fois.

– Objectif 3 : Tester que le champ "*nombre de Route Discovery initiés pour chaque destination*" de la *Route Request Table* est mis à jour avec chaque *Route Reply* valide reçu.

Ces trois objectifs de test peuvent révéler des erreurs cruciales dans l'implantation de DSR principalement durant la phase de *Route Discovery*. Par ces objectifs nous testons que le *Route Cache* est bien consulté avant chaque *Route Request*, pour ne pas inonder le réseau alors qu'on connaît déjà une route, que le *Send Buffer* est utilisé correctement et que la *Route Request Table* est correctement mise à jour après chaque RRep.

Il est à noter que les objectifs de test choisis sont utilisables dans toutes les configurations du réseau puisqu'ils ne dépendent pas d'autres noeuds que celui que nous testons. Après application de notre technique de génération de test automatique, un scénario est produit pour chacun des objectifs de test comme illustré à la Figure 5.

Objectif de test	Longueur des scénarios de test	Durée
♯1	36	219s
♯2	11	28s
♯3	14	32s

Figure 5. *Résultats obtenus.*

Nous pouvons appliquer chaque scénario séparément ou bien les unifier en un seul scénario de test. Ce scénario a une longueur de 61 transitions et un court temps d'execution (sur un *Sun Sparc Ultra 5*). Nous devons préciser que notre scénario de test inclut le comportement d'autres composants, comme Transmission ou USR. Les Figures 7 et 8 présentent respectivement les scénarios de test obtenus et la MSC générée correspondante. Ce scénario montre le deuxième objectif de test, en gras sur le schéma 7. Le scénario est produit jusqu'à ce que les deux autres objectifs de test soient atteints. Dans ce scénario nous ne détaillons pas le `Preamble(1)` qui est le scénario permettant d'atteindre la composante spécifique de *Route Discovery* du système.

En outre, l'application de notre technique de génération automatique de scénarios de test nous a permis d'éprouver nos choix de spécifications. En effet, les scénarios ont été correctement générés, et notamment les connectivités et les changements de topologie ont été respectées.

5.3. *Verification de notre spécification par le langage GOAL*

Comme nous le mentionnons dans la Section 3, afin d'assurer la validité de notre spécification, nous devons vérifier de façon formelle, la correction de certaines de ses parties que nous utilisons pour le test. C'est à dire que notre spécification vérifie bien les fonctionnalités testées et définies dans le cahier des charges. Pour ce faire, nous avons utilisé GOAL (*Geode Observation Automata Language*) [ALG 95] qui fait partie des outils de GEODE. GOAL repose sur le principe d'observateur. Un observateur GOAL est un module de type SDL associé à une spécification donnée, nommée modèle, dont il peut connaître les valeurs des variables et monitorer les événements. Nous définissons un observateur par une série de *probes*, ou sonde, que nous associons à un

objet du modèle. Pour nos besoins, nous allons associer une sonde principalement au processus DSR. Une sonde associée à un processus permet l'accès aux valeurs des variables, aux états et horloges, à la file des signaux et aux événements divers (appels de procédures ou échanges de signaux). Pour définir formellement un observateur, GOAL s'appuie sur une variante de SDL, donc sur le modèle des EFSMs, pour obtenir des états de controles :

– *accepting state* : état qui dénote une évolution licite de l'observateur

– *success state* : indique que le modèle respecte le comportement attendu

– *error state* : détecte une erreur dans la séquence que suit le modèle

Ainsi nous créons un observateur en caractérisant ses états, les signaux provenant des sondes qui entrainent un changement d'état ou une série de tâches menant à un nouvel état. Ce graphe doit mettre en évidence les propriétés que nous cherchons à vérifier. Ensuite, à partir du graphe d'accessibilité, nous pouvons automatiquement vérifier que toutes les parties du graphe qui nous intéressent remplissent les propriétés attendues. Ces propriétés peuvent être exprimées en utilisant une logique temporelle linéaire [Z.M 92] puis être traduites en GOAL.
Une formule temporelle est constituée de prédicats, booléens, opérateurs ($\vee, \wedge, \neg, \Rightarrow, \Longleftrightarrow$), quantificateurs ($\forall, \exists$) et d'opérateurs temporels tels que $\Box$ ("à chaque moment dans le futur"), $\Diamond$ ("éventuellement"), $\blacklozenge$ ("à un moment dans le passé"), afin de résonner sur les états courants, postérieurs et précédents. Nous utilisons une logique temporelle afin de spécifier formellement nos propriétés et permettre une écriture plus aisée de celle-ci en GOAL. Nous définissons dans ce qui suit les trois objectifs de tests en utilisant cette logique.

– Objectif 1 :

$$P \equiv \forall n, SEND!RRep_n \Longrightarrow \forall n, \blacklozenge(rc_n!CONSULT)$$

– Objectif 2 :

$$P \equiv \forall n, \forall p(DISCARD_p!SendBuffer \Longrightarrow \blacklozenge(SENT_p!attempt) \geq 3)$$

– Objectif 3 :

$$P \equiv \forall n, \forall RRep, \Box(RRep_n!received \Longrightarrow UPDATED!RReqTable_n!rdn)$$

En d'autres termes, P est un prédicat décrivant l'ensemble des *accepting states* n vérifiant la propriété dénotée. Dans ce qui est susmentionné, p définit un paquet non typé et les actions sont en lettres capitales. Une fois ces propriétés écrites, elles sont traduites en GOAL en faisant intervenir le nom des états mis en jeux ainsi que les variables et paramètres considérés. Par souci de clarté, le code GOAL des deux premiers objectifs est donné à la fin du document.

Cette phase de vérification de notre spécification est très importante et montre la correction et la pertinence de nos séquences de test.

6. Conclusions et Travaux Futurs

Dans cet article, un modèle de validation pour le protocole de DSR a été présenté. Il inclut une spécification formelle du protocole, une méthode et un outil pour la génération automatisée de scénarios de test. Ce modèle de validation présente plusieurs avantages. D'abord, la conception des spécifications formelles, desquelles découlent les tests, contribue à éliminer des erreurs et des ambiguïtés de conception d'une implantation. Ce modèle formel est particulièrement bien adapté pour la description de DSR car il tient compte d'une des caractéristiques principales des réseaux ad hoc : nous pouvons ajouter et éliminer des noeuds d'une manière dynamique tout comme le nombre de noeuds dans une topologie connexe change dynamiquement. Deuxièmement, l'utilisation des objectifs de test pour la génération de test est très utile pour répondre aux exigences des utilisateurs. En outre, la génération automatisée de test est moins coûteuse qu'une génération manuelle, réduisant le délai d'arrivée sur le marché de l'implantation. Aussi cette génération est réalisée à partir d'objectifs aisément mis en place en utilisant des MSC. Ceci permet notamment à des non experts du langage de spécification de produire des scénarios de test. Et en conclusion, les scénarios que nous avons produits peuvent être réutilisés pour le test non fonctionnel tel que les tests de capacité de systèmes et les tests de temps de réponse. La méthodologie proposée peut être appliquée au système de grande taille et être facilement appliquée aux réseaux ad hoc étendus. De même, cette méthode peut être utilisée pour d'autres protocoles de routage tels que OLSR ou AODV.

Les perspectives de ces travaux sont nombreuses. Tout d'abord, nous allons spécifier complètement DSR par rapport à [JOH 03] puisque de nombreuses fonctionnalités doivent encore être ajoutées à l'actuelle spécification, principalement les algorithmes de *Route Shortening* et le *Flow State Extension*. Nous définirons les objectifs de test permettant de couvrir la majorité de la spécification de façon à mettre en place des scénarios de test utiles (*Route Maintenance* et autres fonctionnalités). Pour cela, des études de couvertures de spécifications sont en cours.

Ensuite nous essaierons d'appliquer notre méthode de test sur une description de DSR en simulateur. Nos premières études nous laissent penser que nous utiliserons le test de conformité de l'implantation de DSR sur NS2 [ISI]. Une fois assurés de la conformité de DSR sur le simulateur, nous utiliserons celui-ci pour guider notre recherche de topologies intéressantes en prenant en compte la couverture des tests.

A l'issue de cette phase, nous mettrons en application notre méthode sur une implantation réelle de DSR dont la topologie du réseau et son évolution seront guidées par nos études sur NS. Pour ce faire, il nous faudra vérifier que les architectures de test actuelles définies par l'OSI [ISO], dédiées aux réseaux classiques, sont applicables aux réseaux ad hoc ; dans le cas contraire, nous devrons définir une nouvelle architecture de test.

7. Bibliographie

[ALG 95] ALGAYRES B., LEJEUNE Y., HUGONNET F., « GOAL : Observing SDL Behaviors with GEODE », *The Proceedings of SDL'95*, Elsevier, 1995, p. 223–230.

[BAE 01] BAE S., LEE S.-J., GERLA M., « Multicast Protocol Implementation and Validation in an Ad Hoc Network Testbed », *Proc. IEEE ICC*, 2001, p. 3196-3200.

[BRO 98] BROCH J., MALTZ D., JOHNSON D., « A performance comparison of multihop wireless ad hoc network routing protocols », *the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Octobre 1998, p. 85–97.

[CAV 96] CAVALLI A., CHIN B., CHON K., « *Testing methods for SDL systems in Computer Networks and ISDN Systems* », vol. 28, p. 1669–1683, 1996.

[CAV 99] CAVALLI A., LEE D., RINDERKNECHT C., ZAIDI F., « An algorithm for embedded testing with applications to IN services », KLUWER, Ed., *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99*, vol. 156 de *IFIP Conference Proceedings*, Beijing, China, 1999, Jianping Wu, Samuel T. Chanson, and Qiang Gao.

[CLA 03] CLAUSEN T., JACQUET P., « Optimized Link State Routing Protocol (OLSR), RFC 3626 », IETF MANET Working Group, http ://hipercom.inria.fr/olsr/rfc3626.txt, 2003.

[DUB 99] DUBUISSON O., *ASN.1*, Springer, 1999.

[ETS 03] ETSI, « TTCN3 – Core Language », rapport, 2003, ETSI.

[HAA 02] HAAS Z., PEARLMAN M., SAMAR P., « The Zone Routing Protocol », Internet-Draft, draftietfmanetzonezrp 04.txt, Work in Progress, Juillet 2002.

[IET 04] IETF, « MANET Working Group, MANET Charter », `http://www.ietf.org/html.charters/manet-charter.html`, 2004.

[ISI] ISI, « The Network Simulator - NS-2 », `http://www.isi.edu/nsnam/ns/`.

[ISO] ISO, « ISO/IEC 9646- Technologies de l'information – Interconnexion de systèmes ouverts – OSI) – Cadre général et méthodologie des tests de conformité – ».

[ITU 95] ITU-T, « Recommandation Z.120, Annexe B : Algebraic semantics of Message Sequence Charts », April 1995, Geneva.

[ITU 99] ITU-T, « Recommandation Z.100 : CCITT Specification and Description Language (SDL) », rapport, 1999, ITU-T.

[JOH 96] JOHNSON D., MALTZ D., « *Dynamic Source Routing in Ad Hoc Wireless Networks* », chapitre 5, p. 153–181, Kluwer, tomasz imielinski and hank korth edition, 1996.

[JOH 03] JOHNSON D., MALTZ D., , HU Y., « The dynamic source routing protocol for mobile ad hoc networks (dsr) », draftietf manetdsr09.txt,Work in progress, Avril 2003.

[KER 99] KERBRAT A., JÉRON T., GROZ R., « Automated test generation from SDL specifications », *SDL Forums 1999*, June 1999, p. 135–152.

[LEE 96] LEE D., YANNAKAKIS M., « Principles and Methods of Testing Finite State Machines a Survey », *The Proceedings of IEEE*, vol. 84, Aout 1996, p. 1090–1123.

[LIN 03] LIN T., MIDKIFF S., PARK J., « A Framework for Wireless Ad Hoc Routing Protocols », *Proc. of IEEE Wireless Communications and Networking Conf. (WCNC)*, 2003.

[OBR 02] OBRADOVIC D., « Formal Analysis of Routing Protocols », PhD thesis, University of Pennsylvania, 2002.

[PER 03] PERKINS C., BELDING-ROYER E., DAS S., « Ad hoc On-Demand Distance Vector (AODV) Routing, RFC 3561 », IETF MANET Working Group, `http://www.ietf.org/rfc/rfc3561.txt`, Juillet 2003.

[ROU 02] ROUILLARD D., CASTANET R., « Generate certified test cases by combining theorem proving and reachability analysis », *Proceeding of IFIP TC6 /14th International Conference on Testing of Communicating Systems TESTCOM 2002*, Mars 2002, p. 249-2-66.

[U.G 03] U.GLÄSSER, Q-P.GU, « Formal Description and Analysis of a Distributed Location Service for Mobile Ad Hoc Networks », rapport, 2003, Fraser Univ.

[V.D 01] V.D.PARK, M.S.CORSON, « Temporally-Ordered routing algorithm (TORA) Version1 », fonctionnal specification. IETF, draft-ietf-manet-tora-spec-03.text, Juin 2001.

[VER 97] VERILOG, « ObjectGEODE Simulator », 1997.

[WIK 04] WIKIPEDIA, « Ad hoc protocol list », `http://en.wikipedia.org/wiki/Ad_hoc_protocol_list`, 2004.

[YI 03] YI Y., PARK J.-S., LEE S., LEE Y.-Z., GERLA M., « Implementation and Validation of Multicast-Enabled Landmark Ad-hoc Routing (M-LANMAR) Protocol », *IEEE MILCON'03*, 2003.

[Z.M 92] Z.MANNA, PNUELI A., *The Temporal Logic of Reactive and Concurrent Systems :Specification*, Springer-Verlag, 1992.

```
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).
```

Figure 6. *Un fragment du scénario de test généré.*

Propriétés GOAL

1- Vérification de la consultation du *Route Cache* avant un envoi de *Route Request*.

```
Probe Objectif1 DSR_node!DSR;
ERROR STATE Fail;
SUCCES STATE Ok;

START;
NEXTSTATE Obj1;

STATE Obj1;
    WHEN OUTPUT IP_in FROM Objectif1;
    IF IP_in!Dsr_header!options(1)!present = RReq;
        THEN IF Objectif1!has_route = FALSE;
            THEN NEXTSTATE Ok;
            ELSE NEXTSTATE Fail;
        ELSE NEXSTATE Obj1;
```

2- Vérification qu'un noeud utilise trois fois le *SendBuffer* avant d'écarter un paquet.

```
Probe Objectif2 DSR_node !DSR ;
ERROR STATE Fail ;
SUCCES STATE Ok ;

START ;
NEXTSTATE O2s1 ;
STATE O2s1 ;
      WHEN OUTPUT IP_in FROM Objectif2 ;
      IF IP_in !Dsr_header !options(1) !present = RReq ;
      THEN IF Objectif2 !has_route = FALSE AND Objectif2 !PushSendBuffer != 1 ;
            THEN NEXSTATE Fail :
            ELSE NEXSTATE O2s2 ;
      ELSE O2s1 ;
STATE 02s2 ;
      WHEN INTPUT IP_in TO Objectif2 ;
      IF IP_in !Dsr_header !options(1) !present = RRep
      THEN IF IP¨_in !dst = Objectif2 !OWN_ADDRESS AND Objectif2 !IsInRouteCache = FALSE
      AND Objectif2 !iDinSendBuffer != 0
            THEN IF Objectif2 !rmSendBuffer = 1 ;
                  THEN NEXSTATE Ok ;
                  ELSE NEXSTATE Fail ;
            ELSE NEXTSATE O2s2 ;
      ELSE NEXSTATE 02s2 ;
```

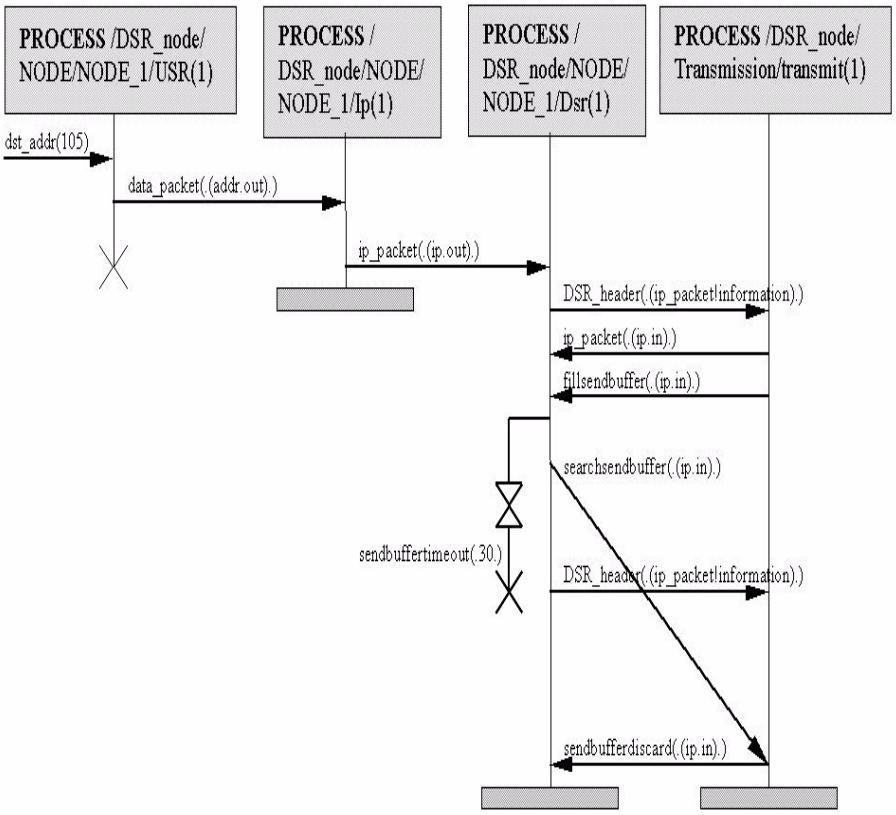

Figure 7. *MSC obtenu par la génération de scénario de test*