Utilisation de la technologie active pour tester les protocoles multicast IPv6

Mohamed Salah Bouassida

2001/2002 Soutenu à la session de juillet 2002

Université Manouba

ECOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE

RAPPORT
De Mémoire de Fin d'Etudes

Présenté en vue de l'obtention du titre
d'INGENIEUR EN INFORMATIQUE

Par :

BOUASSIDA MOHAMED SALAH

Sujet :

UTILISATION DE LA TECHNOLOGIE ACTIVE POUR TESTER LES PROTOCOLES MULTICAST IPV6

Nom du responsable : Mme Chrisment Isabelle

Encadré par : Mme Chrisment Isabelle

Organisme : LORIA / RESEDAS

Adresse : Campus Scientifique BP 239 54506 Vandoeuvre Les Nancy Cedex

Tel : 0383592000 Fax : 0383413079

A mes très chers parents

Nulle dédicace n'est susceptible de vous exprimer ma profonde affection, mon immense gratitude pour tous les sacrifices que vous avez consacrés pour mon éducation et mes études.

A ma petite sœur Marwa

A toute ma famille

A tous mes amis

A tous ceux qui m'aiment

Remerciements

Je tiens à remercier Monsieur André shaff et Monsieur Olivier Festor pour m'avoir donné la possibilité d'effectuer mon stage au sein de l'équipe RESEDAS.

J'adresse mes plus sincères remerciements à mon encadrante Mme Isabelle Chrisment pour sa prestigieuse aide et toute l'attention dont j'avais été l'objet tout au long de la réalisation de ce projet.

Je remercie vivement tous mes amis au LORIA pour l'ambiance chaleureuse qu'ils ont pu toujours maintenir, je remercie en particulier Monsieur Abdelkader Lahmadi pour son aide précieuse tout au long de mon stage.

Je tiens à exprimer l'honneur que me font tous les membres du jury en acceptant de juger mon travail.

Que tous trouvent ici l'expression de mes sentiments les plus respectueux.

Résumé

Ce stage de fin d'étude a pour objectif la mise en œuvre d'une suite de tests pour le protocole MLD d'IPv6, basés sur les réseaux actifs. Ces tests ont pour finalité de contrôler le fonctionnement et la conformité de MLD par rapport à sa spécification formelle, ils doivent ainsi vérifier que les fonctionnalités spécifiées sont bien implémentées.
L'environnement d'exécution utilisé est l'environnement actif FLAME, réalisé dans le projet RESEDAS du LORIA.

Mots clés : IPv6, MLD, FLAME, Multicast, Groupe, Tests, API.

Abstract

The goal of this final-year project is to test MLD protocol (IPv6 protocol) using active technology. The tests have to verify the conformity of the MLD protocol with its formel specification.
The execution environment used is the active environment called FLAME which was developed with the RESEDAS project at LORIA.

Keywords : IPv6, MLD, FLAME, Multicast, Group, Tests, API.

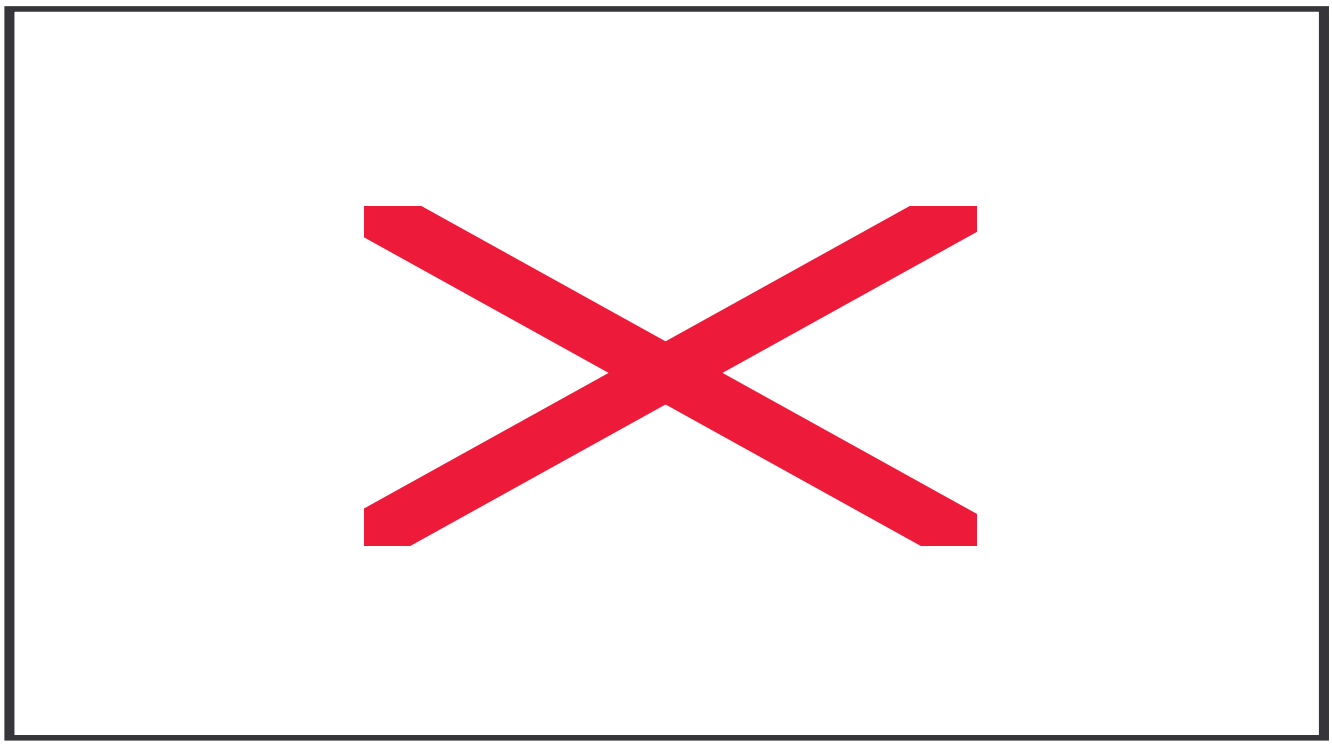


Table de Matières

Table des figures

Chapitre 1

INTRODUCTION

1.1 Contexte général :

L'avènement d'IPv6 a permis de remédier aux différents problèmes rencontrés dans la version IPv4. En effet, il a pu apporter des solutions au problème de saturation de l'espace d'adressage et au problème de la croissance incontrôlée des tables de routage. Plusieurs fonctionnalités sont plus élaborées dans Ipv6 telles que la configuration automatique, la mobilité, la sécurité, le support de la QOS et le multicast.

Plusieurs travaux ont été effectués pour contribuer à la mise en place des nouveaux protocoles d'IPv6. Dans ce sens, il y a eu, d'un côté la proposition de spécifications informelles sous forme de RFCs et de Drafts IETF en vue d'apporter des solutions répondant aux objectifs visés par IPv6. D'un autre côté, il y a eu le développement et le déploiement de ces protocoles à travers des projets tel que KAME.

Les protocoles de communication, comme tous les logiciels, passent par plusieurs étapes lors de leur cycle de développement à savoir : l'expression des besoins, la spécification formelle, la vérification, l'implémentation, les tests et la maintenance. Les tests sont actuellement l'un des meilleurs moyens pour contrôler le fonctionnement et la conformité des protocoles par rapport aux spécifications. Ils permettent de s'assurer que les fonctions spécifiées sont correctement implantées.

La génération automatique des tests à partir de spécifications formelles a fait l'objet de plusieurs travaux de recherche. Cependant, les différentes méthodes de génération automatique de test restent impraticables pour les systèmes complexes. D'un autre côté, le respect des étapes du cycle du développement d'un protocole prend un temps énorme. Ceci a poussé des organismes tels que UNH et TAHI à procéder à l'écriture des tests à partir des spécifications informelles d'IPv6.

La réalisation des tests de protocoles peut se baser sur une approche traditionnelle de réseaux, mais l'apport d'une technologie récente comme les réseaux actifs semblait être intéressant pour développer une infrastructure facilitant le déploiement et l'exécution de tests de conformité et d'interopérabilité par rapport au logiciel existant.

Dans le cadre de mon projet de fin d'études à l'Ecole Nationale des Sciences de l'Informatique, je suis affecté à l'équipe RESEDAS du LORIA pour réaliser un projet intitulé « Utilisation de la Technologie active pour tester les protocoles Multicast IPv6 » ; le but de mon stage était de mettre en œuvre des tests pour le protocole MLD V1 (Multicast Listener Discovery) en se basant sur l'environnement d'exécution des réseaux actifs FLAME réalisé à l'équipe de recherche RESEDAS.

1.2 Organisation du rapport :

Le présent rapport est structuré de la façon suivante. Le chapitre 2 donne un aperçu sur le protocole IPv6, le protocole MLD et les réseaux actifs. Dans le chapitre 3 est présentée la spécification des tests à réaliser. Le chapitre 4 présente la conception générale basée essentiellement sur l'application des réseaux actifs sur les tests MLD. Dans le chapitre 5 est détaillée la réalisation des tests et de l'API, enfin le chapitre 6 conclut le rapport.

1.3 Présentation de l'équipe de recherche:

1.3.1 Présentation de LORIA :

Le Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) est une unité mixte de recherche commune au CNRS (Centre National de la Recherche Scientifique), à l'INRIA (Institut National de Recherche en Informatique et en Automatique), à l'INPL (Institut National Polytechnique de Lorraine) et aux universités Henri Poincaré, Nancy 1 et Nancy 2.

Cette unité, dont la création a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Education Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires, succède ainsi au Centre de Recherche en Informatique de Nancy (CRIN) et aux équipes communes entre celui-ci et l'unité de Recherche INRIA de Lorraine.

L'équipe de recherche LORIA est répartie en 21 équipes de recherche et 7 services d'aide à la recherche.
Chaque équipe rassemble des chercheurs, des doctorants et des assistants techniques ou administratifs, pour la réalisation d'un projet de recherche.
Cinq instances ont été mises en place pour assister le directeur du laboratoire, garantir la cohérence de la politique scientifique et le bon fonctionnement au quotidien :

- l'Equipe de Direction : composée de plusieurs membres du laboratoire, elle assiste le directeur dans ses fonctions
- le Comité de Gestion : composé des chefs de service, il assiste le directeur dans le fonctionnement journalier du LORIA
- le Comité des Projets : il conseille le directeur sur la politique scientifique du LORIA, participe à l'évaluation des projets/équipes, et instruit les restructurations nécessaires
- le Conseil du LORIA : il émet des avis sur la politique scientifique mise en œuvre par le Comité des Projets.
- le Conseil des Orientations Scientifiques : composé de représentants des équipes de recherche, il conseille la direction dans la gestion scientifique du laboratoire.

Les cinq thématiques principales sur lesquelles les différentes équipes développent des recherches fondamentales et appliquées sont :

- Calculs, réseaux et graphismes à hautes performances.
- Télé-opérations et assistants intelligents.
- Ingénierie des langues, du document et de l'information scientifique et technique.
- Qualité et sûreté des logiciels et systèmes informatiques.
- Bioinformatique et applications à la génomique.

1.3.2 Présentation de RESEDAS :

RESEDAS : concepts et outils logiciels pour les télécommunications et les systèmes distribués.

Le projet RESEDAS développe ses activités de recherche sur les environnements logiciels pour la conception, le développement, le déploiement, l'exploitation et la supervision des applications, services et protocoles de communication dans des environnements hétérogènes.

Trois axes de recherche sont dans le projet RESEDAS :

- Calculs distribués et transfert des informations : contributions à l'approche "Machine Virtuelle" via les bibliothèques de communications, méthodes et langages assurant efficacité des communications et placement optimal des tâches, metacomputing.

- IPv6: expérimentation et validation de plate-forme, multicast, qualité de service, sécurité, supervision, couplage des réseaux actifs et services d'IPv6.

- Supervision et contrôle des réseaux et services: intégration d'approche, RGT, plates-formes Java pour le RGT, agents mobiles, modèles de l'information, réseaux actifs, SNMP, WBEM.

Chapitre 2

ETUDE DE L’EXISTANT

2.1 Introduction :

Tout au long de ce chapitre, sera présenté un aperçu du protocole IPv6, ses avantages par rapport à IPv4, le Multicast IPv6, le protocole MLD version 1(Multicast Listener Discovery), l'état de l'art des tests de protocoles et les réseaux programmables et actifs.

2.2 Le protocole IPv6 :

D'abord une adresse IPv6 [GIZ97] est un mot de 128 bits. La taille d'une adresse IPv6 est donc le quadruple de celle d'une adresse IPv4.
La représentation textuelle d'une adresse IPv6 se fait en découpant le mot de 128 bits de l'adresse en 8 mots de 16 bits, chacun d'entre eux étant représenté en hexadécimal et séparés par le caractère « ; ».
La représentation des préfixes IPv6 se fait comme suit :

Adresse IPv6 / longueur du préfixe en bits

On pourrait combiner l'adresse d'une interface et la longueur du préfixe réseau associé en une seule notation, exemple :

3EDC :BA98 :3265 :9845 :AB98 :6972 :DCF9 :9872 / 64

IPv6 reconnaît trois types d'adresses, unicast, multicast et anycast. Le type unicast est le plus simple ; une adresse de ce type désigne une unique interface. Un paquet envoyé à une telle adresse sera donc remis à l'interface ainsi identifiée.
Une adresse de type multicast désigne un groupe d'interfaces qui en général appartiennent à des nœuds différents pouvant être situés n'importe où dans l'Internet. Lorsqu'un paquet a pour destination une adresse de type multicast, il est acheminé par le réseau à toutes les interfaces membres de ce groupe.
Le dernier type, anycast, est nouveau dans IPv6. Comme dans le cas du multicast, une adresse de type anycast désigne un groupe d'interfaces, la différence étant que lorsqu'un paquet a pour destination une telle adresse, il est acheminé à un des éléments du groupe et non pas à tous. C'est par exemple la plus proche au sens de la métrique des protocoles de routage.

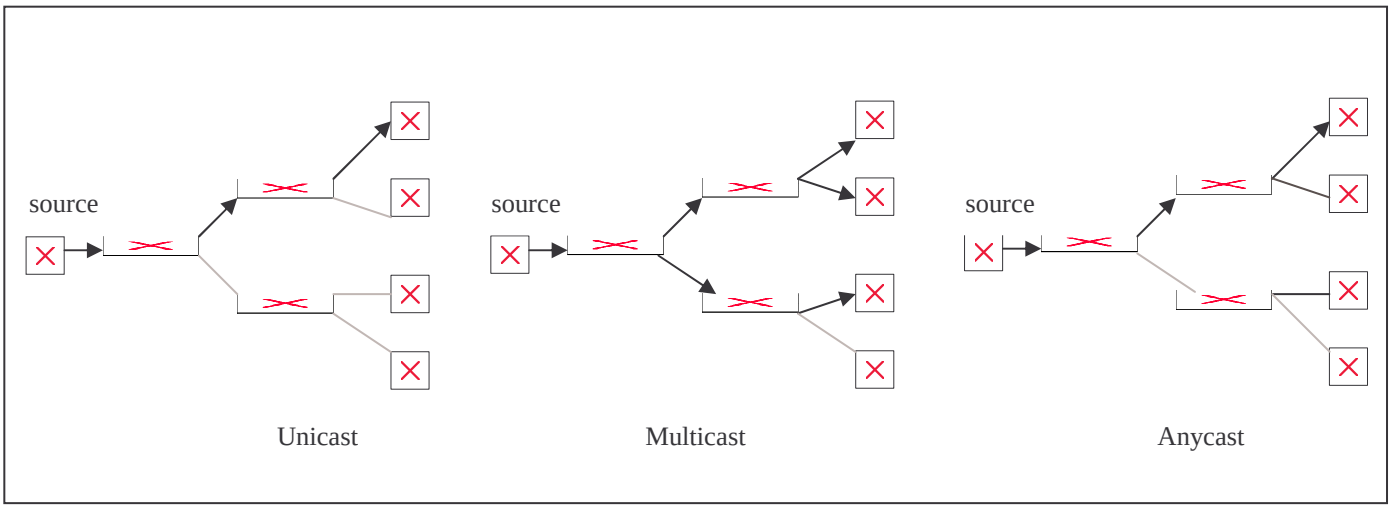


Fig 1- *Modes de transmission IPv6*

2.2.1 Multicast IPv6 :

Une adresse de type multicast désigne un ensemble d'interfaces [RFC 1883, RFC 1884, RFC 1885]. Elle est caractérisée par le préfixe FF00::/8. Son format est comme suit :

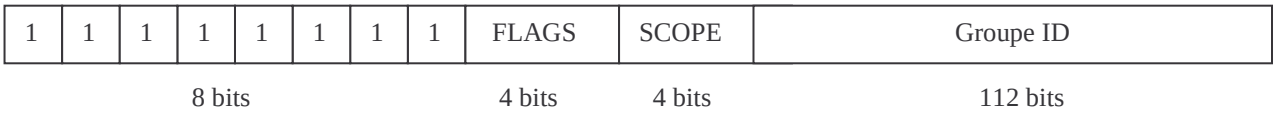


Fig 2- *Format d'une adresse multicast IPv6*

Le champ flags est un mot de 4 bits. Les trois premiers bits sont réservés et doivent être initialisés à 0. Le dernier bit, nommé T, s'il est positionné à 0, indique que la validité de l'adresse est permanente, sinon, l'adresse est temporaire.
Le champ Scope est le niveau de diffusion de l'adresse considérée. Les valeurs actuellement définies sont :

0 *réservé*
1 *nœud (node-local scope)*
2 *lien (link-local scope)*
5 *site (site local scope)*
8 *organisation (organization-local scope)*
E *global (global scope)*
F *réservé*

Exemple : pour le groupe 101 attribué aux serveurs NTP (network time protocol) :
FF02 : :101 à tous les serveurs NTP sur le même lien que l'expéditeur
FF05 : :101 à tous les serveurs NTP sur le même site que l'expéditeur
FF0E : :101 à tous les serveurs NTP sur Internet

2.2.2 Avantages IPv6 :

2.2.2.1 Adressage :

- Les adresses sont assignées aux interfaces (pas de changement par rapport à IPv4).
- Une interface peut avoir plusieurs adresses.
- Les adresses IPv6 ont des scopes et des lifetime.
- Solution au problème de saturation de l'espace d'adressage.
- Solution au problème de la croissance incontrôlée des tables de routage.

2.2.2.2 Gestion :

- Autoconfiguration dynamique des adresses.
- Plusieurs méthodes pour l'obtention d'adresses routables ont été définies:
  - § Link Local Address : pas de routeur ou serveur requis.
  - § Stateless Mechanism : c'est le routeur qui fournit les préfixes.
  - § Statefull Mechanism : c'est le serveur DHCP qui fournit les adresses.

- La mobilité est assurée avec IPv6.
- Performance et optimisation du traffic.

2.2.2.3 Sécurité :

- La sécurité IPv6 est standardisée et toutes les implémentations doivent l'offrir.
- L'architecture de sécurité englobe l'authentification (signature des paquets) et le cryptage (confidentialité des données).
- Protection end-to-end et router-to-router.

2.2.2.4 Qualité de service :

- Support pour diffserv (differenciated service) : le champ Class permet à la source d'identifier la classe de service désirée et d'ajouter une priorité à ses paquets.
- Support pour intserv (integrated service) : permet à la source d'identifier les flots ayant besoin d'un QOS particulier.

2.2.2.5 Multicast :

- Le Multicast IPv6 permet un routage efficace dans le cas d'une communication de groupe. Pour envoyer à un ensemble de destinataires, le multicast réduit énormément la charge du réseau en n'envoyant qu'une seule copie par liaison.
- Le Multicast IPv6 offre aussi une transmission efficace quand les adresses des destinataires sont inconnues ou changent régulièrement. L'utilisation des adresses de groupe au lieu des adresses individuelles permet d'atteindre des destinations sans être obligé de connaître leurs adresses exactes.
- L'intérêt du niveau de diffusion est de garantir le confinement des paquets à une zone déterminée. On peut ainsi s'assurer qu'il n'y aura plus de fuites de paquets vers le réseau mondial. Cette méthode est plus rigide mais plus sure qu'en IPv4, où la portée d'un paquet de multicast est définie en fonction du champ durée de vie.

2.2.2.6 Coexistence avec IPv4:

- Des millions de nœuds utilisent IPv4 aujourd'hui et une partie de ces nœuds ne vont jamais progresser vers IPv6 et donc il fallait une coexistence entre IPv6 et IPv4.
- La transition de IPv4 à IPv6 peut se faire sans isolation des nœuds IPv4.
- On parle alors d'interopérabilité et non de migration.
- Plusieurs solutions pour la transition IPv4-IPv6 : Dual IP layer, Tunnel, Translation d'adresses,…

2.3 MLD : Multicast Listener Discovery :

2.3.1 Objectif :

Le but du protocole MLD [RFC 2710] est de permettre à un routeur de détecter la présence de nœuds sur l'ensemble de ses liens désirant recevoir des paquets relatifs à un groupe multicast. Ces nœuds sont désignés par le terme listeners à l'adresse de ce groupe.
Le protocole MLD permet ainsi de tenir à jour, continuellement, la liste des adresses multicast disposant de listeners. Cette information est utilisée par les protocoles de routage multicast. Elle leur permet d'acheminer les paquets multicast aux différentes destinations intéressées par leur réception.
MLD est un protocole asymétrique dans le sens où il adopte deux comportements différents, selon qu'il s'exécute sur un routeur ou sur un nœud.

2.3.2 Messages :

MLD peut être considéré comme un sous ensemble d'ICMPv6. En effet, les messages MLD forment un sous ensemble de l'ensemble des paquets d'ICMPv6. Ils sont identifiés par la valeur 58 dans l'entête précédant le message MLD.
Un paquet MLD a la forme suivante :

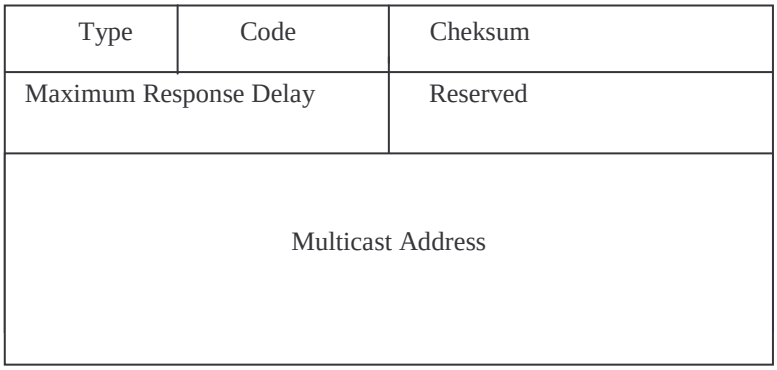


Fig 3 – *Format d'un paquet MLD*

- Champ Type : le champ type indique l'un des trois types que peut avoir un message MLD :
  - Multicast Listener Query dont la valeur est 130.
  - Multicast Listener Response dont la valeur est 131.
  - Multicast Listener Done dont la valeur est 132.

Le premier message peut être de deux types et ce selon la valeur du champ Multicast Address :
  - General Query
  - Multicast Address.

- Champ Code : le champ Code est initialisé à 0 par l'émetteur et ignoré par le récepteur.

- Champ Cheksum : ce champ correspond au cheksum standard ICMPv6 couvrant le message MLD ainsi que l'entête IPv6.

- Champ Maximum Response Delay : le champ Maximum Response Delay est significatif seulement dans les messages de type Query. Il spécifie le délai maximum permis avant l'envoi d'un paquet Report en réponse à ce Query. Il est mis à zéro, par l'émetteur, dans les autres types de messages MLD et ignoré par le récepteur.

- Champ Multicast Address : dans un message Query, ce champ est mis à zéro dans le cas d'un General Query et à 1 dans le cas d'un Multicast Address Specific Query(MASQ). Dans le cas d'un message Done, ce champ comporte une adresse multicast spécifique.

2.3.3 Description d'un nœud MLD :

Trois états sont définis pour une adresse donnée:
Etat 1 : Non Listener : état où le nœud n'écoute pas à cette adresse. Ceci signifie qu'aucune application du niveau supérieur n'a demandé de recevoir des paquets multicast relatifs à cette adresse. C'est l'état initial que prend un nœud pour toute adresse multicast.

Etat 2 : Delaying Listener : état où le nœud est en écoute à l'adresse alors que le Report Delay timer est en exécution.

Etat 3 : Idle Listener : état où le nœud est en écoute à l'adresse alors que le timer relatif n'est pas en exécution.

L'automate relatif à un nœud MLD sera donc comme suit :

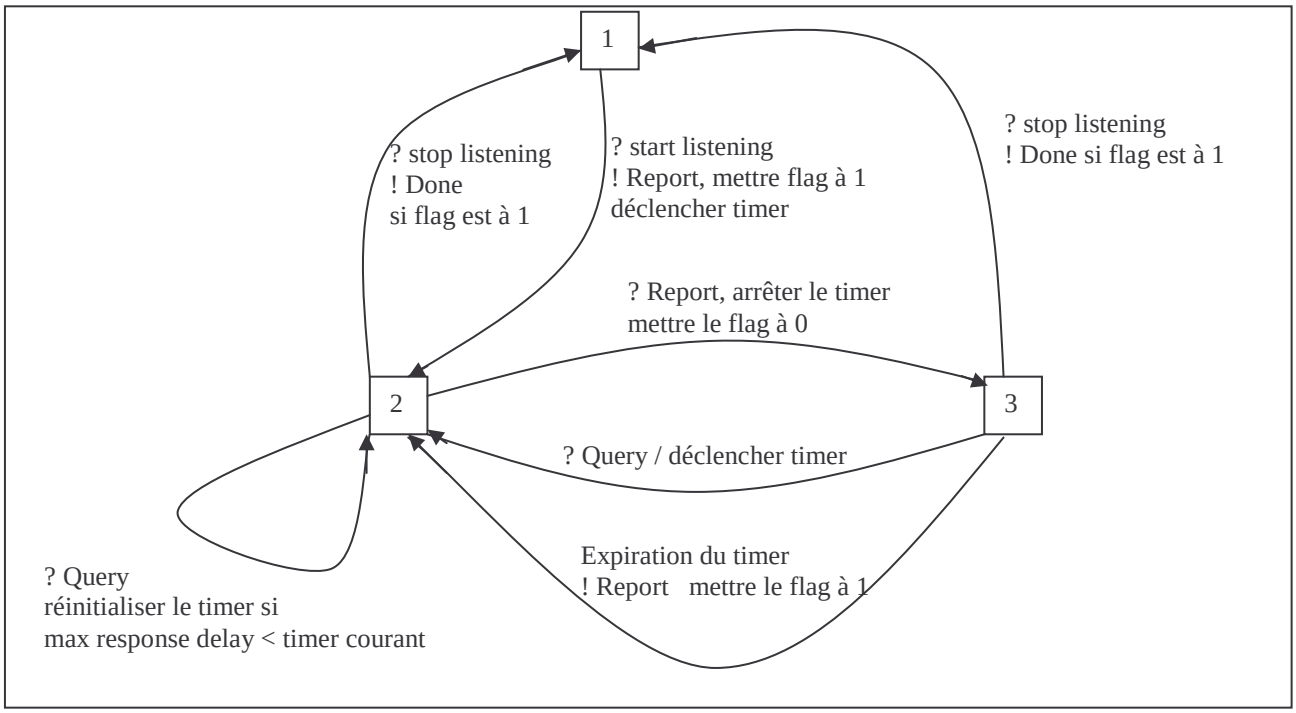


Fig 4 – Automate relatif à un nœud MLD

Les transitions de l'automate sont les suivantes :

- Transition 1 vers 2 : déclenchée par la réception du message Start Listening. Elle donne lieu aux deux actions suivantes : envoi d'un Report et déclenchement du temporisateur relatif à l'adresse concernée.

- Transition 2 vers 1 : déclenchée par la réception du message Stop Listening. Elle donne lieu à l'envoi d'un Done si le flag est mis à 1 (le flag est utilisé pour indiquer si un Report a été récemment envoyé).

- Transition bouclant sur l'état 2 : déclenchée par la réception du message Query. Elle donne lieu à la réinitialisation du temporisateur Maximum Response Delay si sa valeur est inférieure à la valeur courante du temporisateur.

- Transition 2 vers 3 : déclenchée par la réception du message Report, donne lieu à l'arrêt du temporisateur et à la mise du flag à 0.

- Transition 2 vers 3 : déclenchée par la réception du message Timer expired, donne lieu à l'envoi du message Report et à la mise du flag à 1.

- Transition 3 vers 2 : déclenchée par la réception du message Query, elle donne lieu au déclenchement du temporisateur.

- Transition 3 vers 1 : déclenchée par la réception du message Stop Listening, elle donne lieu à l'envoi d'un Done si le flag est mis à 1.

2.3.4 Description d'un routeur MLD :

Trois états pour un routeur MLD :

Etat 1 : initial state.

Etat 2 : Querier.

Etat 3 : Non Querier.

L'automate relatif à un routeur MLD sera donc comme suit :

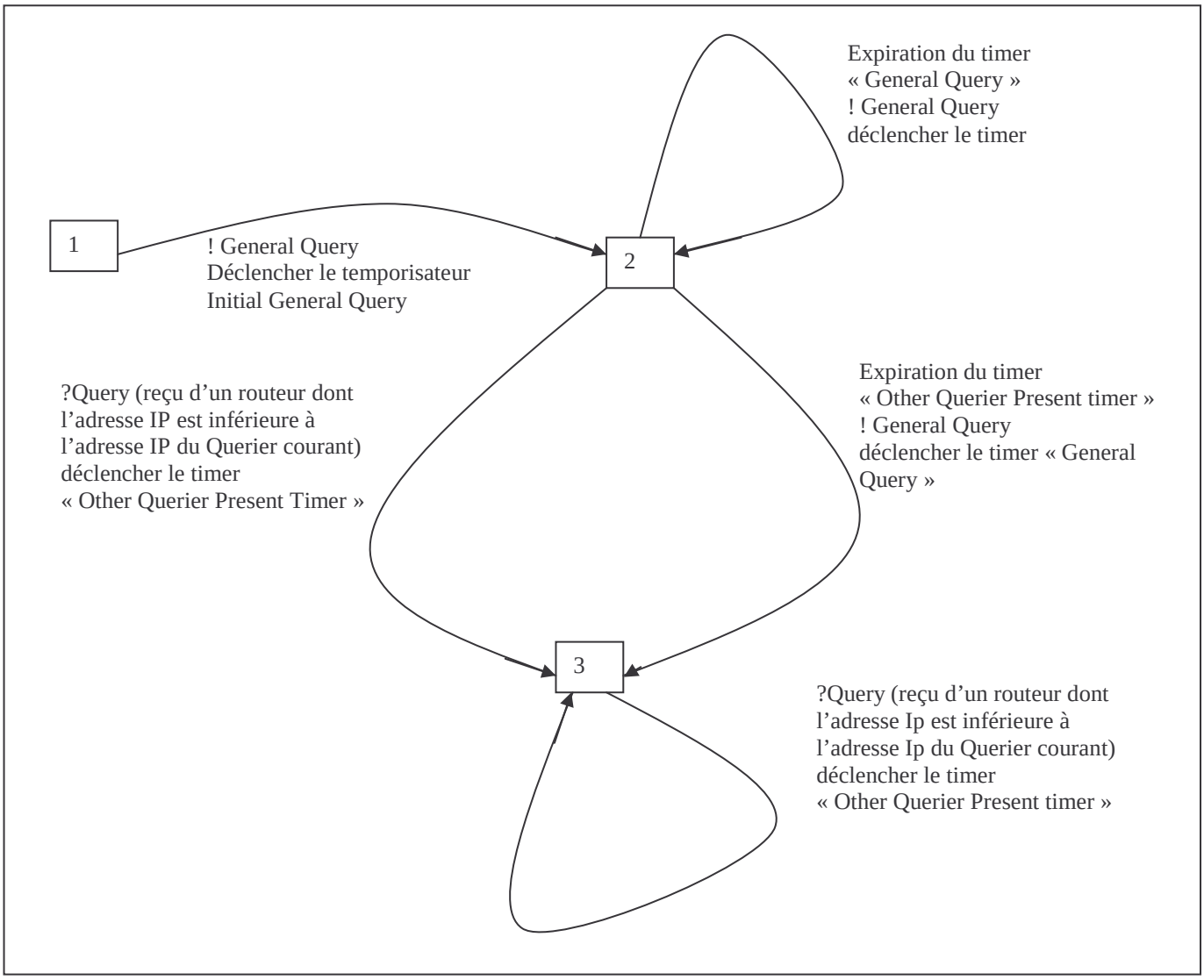


Fig 5 – Automate relatif à un routeur MLD

A son initialisation, un routeur commence par envoyer des General Query, initialise le timer General Query et passe à l'état Querier. Il continue à envoyer un message General Query chaque fois que le timer General Query expire. Si un routeur ayant une adresse IP inférieure à l'adresse IP du routeur Querier courant, émet un message General Query, ce dernier passe à l'état Non Querier et déclenche le timer Other Querier Present Router.

2.3.5 Description d'un routeur Querier :

Pour une adresse donnée, les états que peut prendre un routeur Querier sont les suivants :

Etat 1 : No listeners Present : état où il n'y a pas de listeners pour l'adresse en question.

Etat 2 : Listeners Present : état où il y a des listeners pour l'adresse en question.

Etat 3 : Cheking Listeners : état où le routeur vérifie s'il y a encore des listeners pour l'adresse en question.

L'automate relatif à un routeur Querier sera donc comme suit :

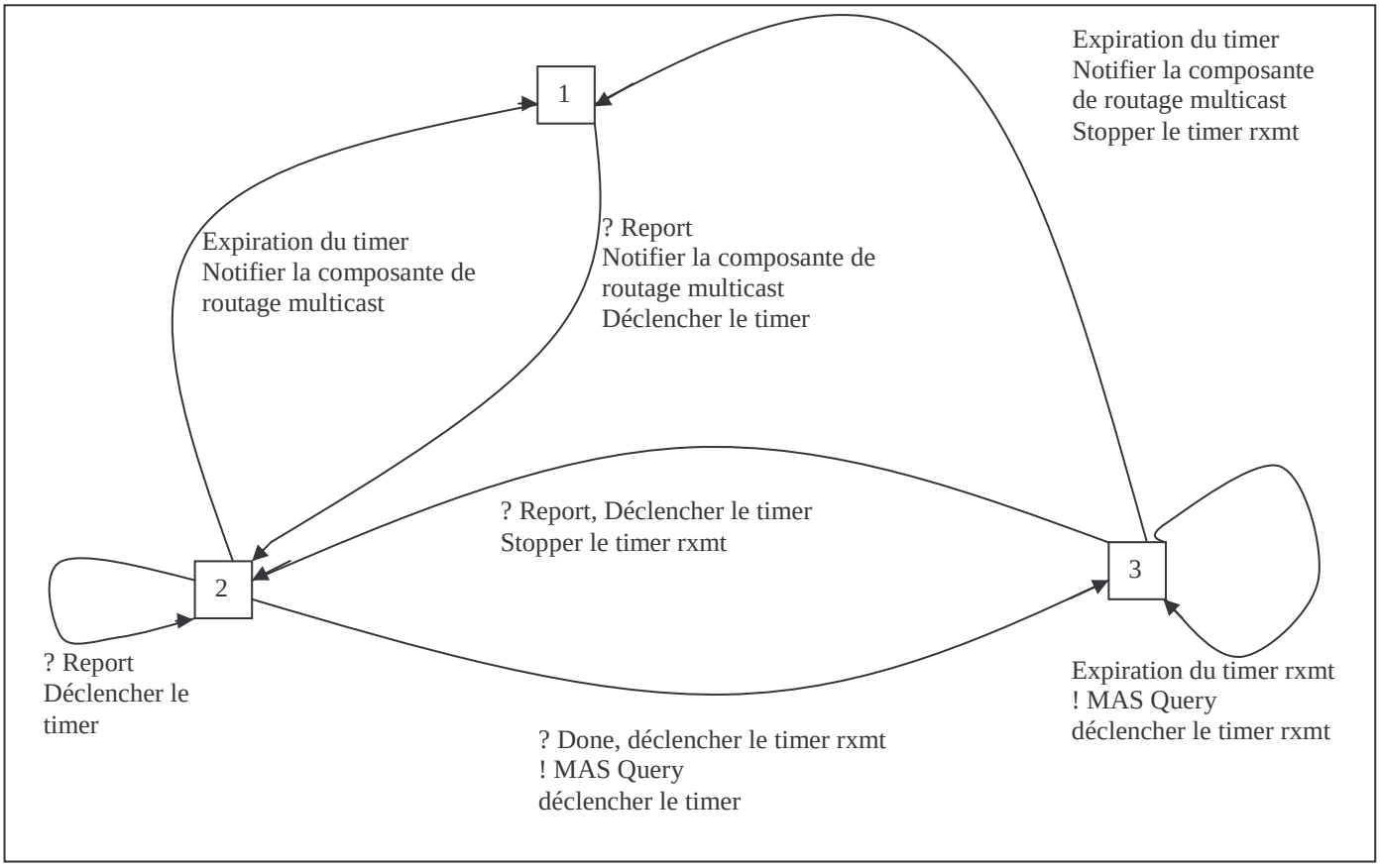


Fig 6 – Automate relatif à un routeur Querier

Les transitions de l'automate sont les suivantes :

- Transition 1 vers 2 : activée par la réception d'un Report, suite à cette transition, la composante de routage multicast est notifiée par la présence de Listeners pour l'adresse rapportée.

- Transition 2 vers 1 : activée par l'expiration du timer relatif à l'adresse en question. La composante de routage multicast est notifiée de l'absence de listeners pour cette adresse.

- Transition bouclant sur 2 : activée par la réception d'un Report. Le timer relatif à l'adresse rapportée est ainsi réinitialisé à la valeur Multicast Listener Interval et est déclenché.

- Transition 2 vers 3 : activée par la réception d'un Done. Suite à cette transition, un message Multicast Address Query est envoyé et le timer rxmt est déclenché (le temporisateur rxmt spécifie l'intervalle de temps pour la retransmission d'un autre Multicast Address Specific Query). Elle permet aussi de déclencher le temporisateur timer dont la valeur est Last Listener Query Interval * Last Listener Query Count.

- Transition 3 vers 2 : activée par la réception d'un Report. Le timer relatif à l'adresse rapportée est déclenché.

- Transition bouclant sur 3 : activée par l'expiration du timer rxmt. Cette transition provoque l'envoi d'un message Multicast Address Specific Query pour l'adresse concernée et la réinitialisation et le déclenchement du timer rxmt.

- Transition 3 vers 1 : activée par l'expiration du timer relatif à l'adresse concernée. Suite à cette activation, la composante de routage multicast est notifiée de l'absence de Listeners pour cette adresse.

2.3.6 Description d'un routeur Non Querier :

Un routeur Non Querier a pratiquement les mêmes transitions et les mêmes états qu'un routeur Querier. La seule différence est que le routeur Non Querier ne réagit pas à la réception des différents événements.

L'automate relatif à un routeur non Querier sera donc comme suit :

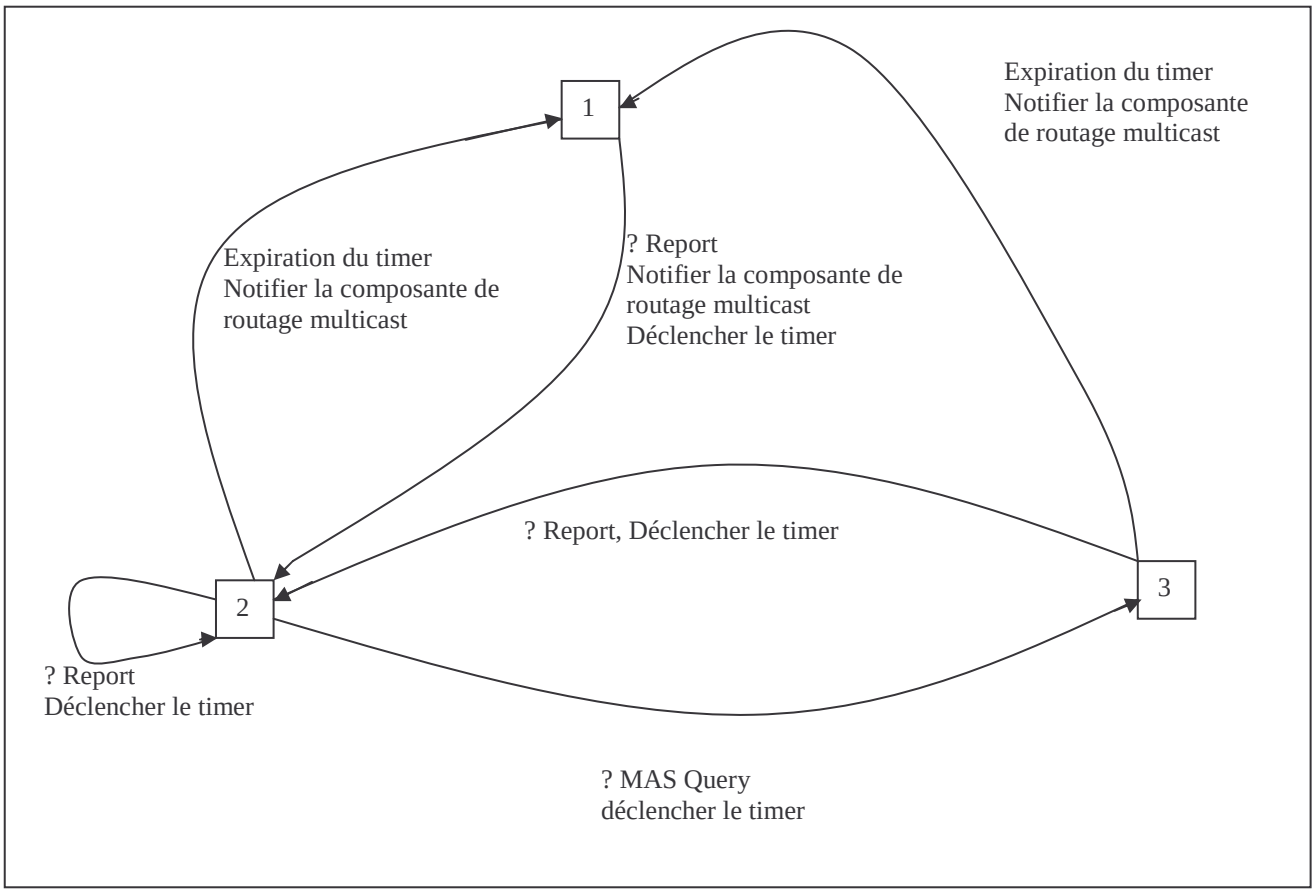


Fig 7 – Automate relatif à un routeur Non Querier

2.4 Etat de l'art des tests de protocoles :

Les tests sont actuellement l'un des meilleurs moyens pour contrôler le fonctionnement et la conformité des protocoles par rapport aux spécifications. Ils permettent de s'assurer que les fonctions spécifiées sont correctement implantées. Plusieurs travaux de recherche ont été réalisés pour mettre au point des générateurs de tests dès la phase de la spécification informelle des protocoles.

2.4.1 Terminologie OSI :

- Test boite noire : ou test fonctionnel consiste à observer le comportement externe du logiciel vis à vis de sa spécification

- Test boite blanche : se base sur la structure interne du logiciel, le but est ainsi de vérifier chaque partie du code du logiciel.

- IUT : entité de protocole à tester : Implementation Under Test. Le testeur simule l'environnement dans lequel l'IUT doit fonctionner correctement. L'IUT est considérée ainsi comme une boite noire qui communique avec le testeur via des points d'accès appelés PCO (point of control and observation).

- Les contraintes statiques pour les tests de conformité : définissent les capacités minimales que doit assurer une implémentation.

- Les contraintes dynamiques, quant à elles, définissent le comportement observable de l'implémentation lors de la communication avec son environnement.

2.4.2 Méthodologie OSI :

Pour s'assurer qu'une implémentation d'un protocole est conforme à sa spécification standard, on doit réaliser des tests de conformité. [VOL 1] [LOU 00]
En effet, ces tests sont d'un énorme intérêt pour les constructeurs et pour les utilisateurs d'un produit ou d'un service de communication.
Pour les constructeurs, l'activité de test fournit un support de vérification du produit OSI durant toutes les étapes de son implémentation. Par conséquent, ceci leur permet de s'assurer que le produit réalisé peut interopérer avec d'autres produits provenant d'autres constructeurs.
Pour les utilisateurs, les tests de conformité leur permettent de s'assurer que le produit OSI acquis supporte les clauses spécifiées par les standards.

Dans le domaine de test de conformité des protocoles, on s'intéresse surtout au comportement visible du protocole. Ainsi, seules les interactions de l'implémentation avec son environnement externe sont prises en compte dans ce type de test. C'est par conséquent un test boite noire. On préfère ce type de tests pour mettre en évidence les fonctions incorrectes ou absentes, les interfaces inadéquates et les performances temps réel.

Le principe de ce type de tests consiste à exécuter le programme avec des jeux d'essais établis à partir d'une analyse des spécifications, et à s'assurer que les résultats obtenus sont conformes à ce qui a été défini dans un document de référence.

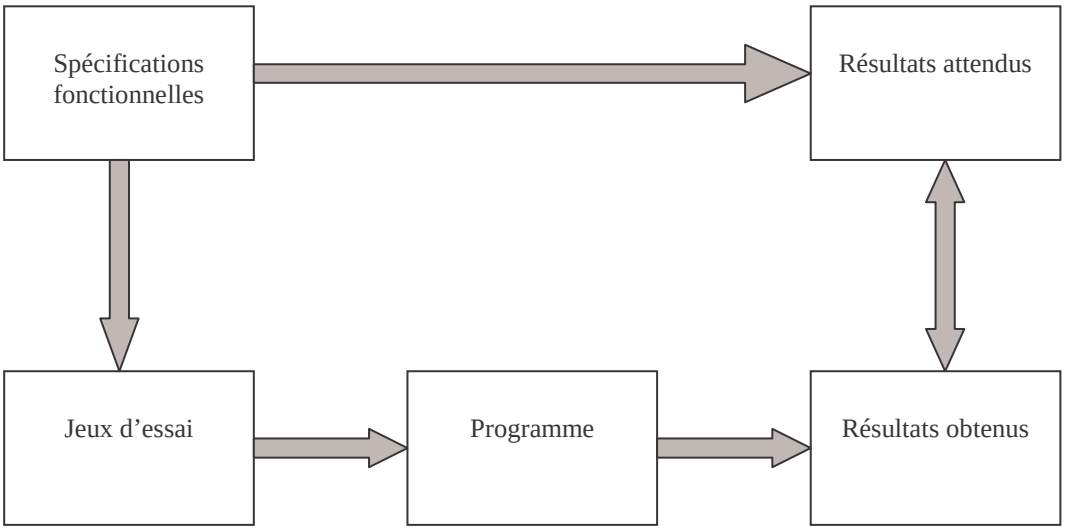


Fig 8 – *Stratégie de test de conformité*

v Développement de suites de test :

Consiste à concevoir et produire des suites de test abstraites. Ces suites de tests sont des collections de tests, appelés cas de tests. Les suites de tests abstraites sont conçues pour tester toutes les options possibles d'un protocole. Lorsqu'on dérive une suite de test d'une suite de test abstraite pour une implémentation spécifique, on parle alors d'une suite de tests exécutable.

ISO 9646 [LOU 00] [ISO-9646] présente différentes méthodes de test pour le développement des suites de test abstraites. Chaque méthode fournit une architecture de test permettant différents niveaux de contrôle. C'est au développeur de suites de test de déterminer la méthode appropriée pour le protocole à tester. Il est possible que plus d'une méthode de test soit utilisée, et dans ce cas une suite de test est construite pour chaque méthode.

On distingue ainsi quatre types d'architecture de test de conformité : test local, test distribué, test coordonné et test à distance.
Le test local correspond au test traditionnel de logiciel, l'IUT et le testeur sont localisés dans le même endroit.

Dans l'architecture distribuée, le système de test est décomposé en deux testeurs, un testeur pour l'interface inférieure de l'implémentation (LT : Lower Tester) et un testeur pour l'interface supérieure (UT : Upper tester). Les deux testeurs communiquent avec l'implémentation à travers des PCO.

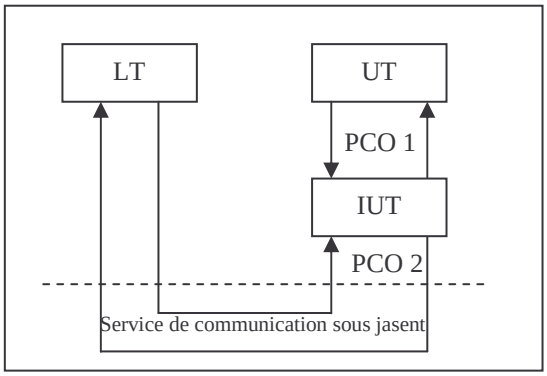


Fig 9 – Architecture de test distribué

L'architecture de test coordonné diffère de la précédente par le fait que les deux testeurs disposent de moyens de coordination.
L'architecture de test à distance est une architecture distribuée où n'est utilisé que le testeur de l'interface inférieure de l'implémentation.

La structure d'une suite de test comprend les éléments suivants :

- Groupe de test de capacité : vérifie les contraintes de conformité statiques de l'IUT.

- Groupe de test de comportement valide : permet d'effectuer des tests sous des conditions de communication normales.

- Groupe de test de comportement invalide : teste la réaction de l'IUT lors d'un comportement anormal.

- Groupe de test d'interconnexion : vérifie la conformité de l'interconnexion de base.

2.4.3 Génération de cas de tests pour les machines à états finis :

Un automate M à états finis est défini à travers 5-uplets (Q, s, E, $\sum$, Ω) :

Q : un ensemble d'états.
s : l'état initial.
E : ensemble d'états finaux.
$\sum$: ensemble d'inputs/outputs.
Ω : ($\sum$ * Q) à Q est une fonction de transition.

Un automate est déterministe si pour tout état de Q et pour tout input de $\sum$, il existe une et une seule transition étiquetée avec cet input. On parle d'automate non déterministe si cette condition n'est pas réalisée.

Un automate est complètement spécifié s'il existe une transition pour chaque input de $\sum$ et pour chaque état de Q. Il est fortement connexe si pour tout couple d'états (s1, s2), il existe une séquence de transitions qui mène de l'état s1 à l'état s2.

Un système peut comporter un nombre très élevé de fautes complexes. De ce fait, une approche de test pratique doit éviter de les manipuler directement, elle doit cependant viser la détection de leur présence ou de leur absence.

De plus une erreur peut être à l'origine de plusieurs fautes. Une des méthodes qui permet de remédier à ce problème consiste à utiliser un modèle de fautes. Ce modèle permet de décrire les effets des fautes à un certain niveau d'abstraction. On peut distinguer des fautes d'output, des fautes de transfert, des fautes de transferts avec des états supplémentaires et des fautes de perte ou d'ajout de transitions :

- Faute d'output : on dit qu'une transition présente une faute d'output si l'automate de l'implémentation fournit un output différent de celui qui a été spécifié.

- Faute de transfert : on dit qu'une transition présente une faute de transfert si l'automate de l'implémentation prend un état différent de celui qui a été spécifié.

- Faute de transfert avec états supplémentaires : certains types d'erreurs peuvent être modélisés par des états supplémentaires et par des fautes de transfert menant à ces états.

- Perte ou ajout de transitions : dans le cas d'une machine d'états non complètement spécifiée ou indéterministe, le modèle de faute doit prendre en compte l'ajout ou la perte de transitions.

2.5 Réseaux Programmables et actifs :

2.5.1 Les réseaux programmables :

Un réseau programmable [OLI 00] est un réseau de transmission de données ouvert et extensible disposant d'une infrastructure dédiée à l'intégration et à la mise en œuvre rapide de nouveaux services sur l'ensemble de ses composants.
L'infrastructure d'un réseau programmable peut être vue comme un réseau de transport de données étendu par un environnement de programmation à l'échelle du réseau comportant un modèle de programmation des services, une architecture de déploiement et d'exploitation pour ces derniers et un contexte d'exécution.
Pour atteindre un niveau de programmabilité suffisant, il faut nécessairement abstraire les différents composants d'un réseau et en faire des objets informatiques.
Ainsi, un réseau programmable est un réseau dans lequel toute ou partie de l'infrastructure de communication est virtualisée.
C'est sur cette virtualisation que reposent tous les concepts et architectures développés dans la communauté réseaux.

à Apports des réseaux programmables :

- Les traitements à appliquer sur les données échangées entre applications, la topologie logique des flux entre les applicatifs permettent au réseau d'agir de manière optimale sur ces données et d'accorder au mieux la topologie logique du flux de l'application à sa topologie physique.

- La standardisation des services n'est plus nécessaire : au travers d'interfaces de programmation standardisées une fois pour toutes, on pourrait développer et déployer rapidement de nouveaux services et protocoles.

- Les réseaux programmables permettent de mettre en place des architectures souples, ouvertes et extensibles : disposer de bases solides.

- Les capacités de traitement et de mémoire suivent une croissance exponentielle : meilleure utilisation de ces capacités de traitement et meilleure utilisation du réseau en optimisant les flux grâce à des traitements dans les nœuds intermédiaires.

- Offrir une infrastructure unifiée et intégrée pour réaliser des applicatifs qui infèrent directement sur le comportement du réseau, et ceci à différents niveaux : application, présentation, session, transport, réseau.

2.5.2 Les réseaux actifs :

Un réseau actif [OLI 00] est un réseau dans lequel tout ou partie de ses composants dans les différents plans (signalisation, supervision, données) sont programmables dynamiquement par des entités tierces (opérateur, fournisseur de services, applications, usagers).
Cette approche est plutôt issue des travaux sur l'insertion de services applicatifs dans le réseau Internet par opposition aux réseaux programmables issus de travaux sur la signalisation pure dans les réseaux de télécommunication de type ATM. Elle étend le concept de

programmation en partant du principe que virtuellement tout usager peut concevoir, déployer et utiliser un nouveau service de manière dynamique dans le réseau.
Son caractère ouvert lui laisse également une liberté bien plus grande sur les choix d'architectures et de technologies. Cette famille d'approche peut être affinée en deux classes : approche *nœud actif* et approche *paquet actif*.

2.5.2.1 Approche Paquet Actif :

Dans les approches paquet actif, le code des services déployés dans les nœuds du réseau est transporté jusqu'au nœuds dans le même flux que les données traitées par le service visé. On parle alors d'une approche intégrée. Le cas standard consiste en l'envoi de paquets de code pour le déploiement de service avant l'envoi de paquets de données. Le cas extrême permet à chaque paquet de données de véhiculer également un mini programme contenant la logique du programme à lui appliquer.

2.5.2.2 Approche Nœud Actif :

Dans les approches nœud actif, les services sont également déployés dynamiquement sur les nœuds du réseau mais ce déploiement ne se fait pas conceptuellement dans le même flux que celui des données utilisateurs. Ceci se rapproche fortement du concept du réseau programmable. La différence avec ce dernier repose sur le fait que dans l'approche nœud actif, les composants de service sont déployés physiquement sur les nœuds du réseau, plus souvent à des techniques de code mobile. Dans les réseaux programmables, la logique de service et son implantation ne nécessitent la plupart du temps pas de déploiement sur les nœuds, celle ci étant exécutée au sein de l'environnement de programmation du réseau. Cette programmation externe permet également de donner le nom d'approche discrète à cette famille qui intègre également un sous ensemble des propositions autour de la signalisation ouverte.

Le concept de réseau actif peut être schématisé comme suit :

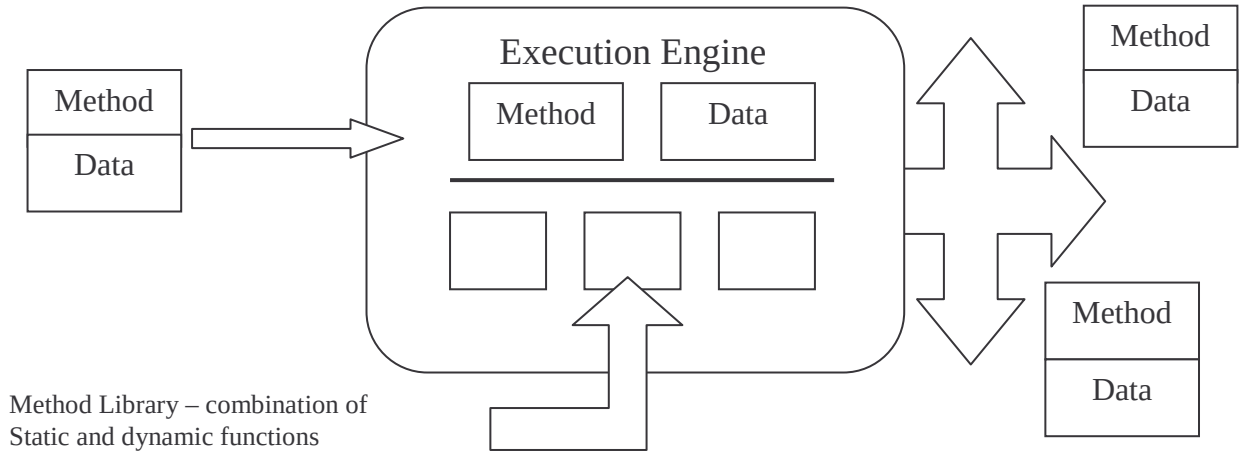


Fig 10 – *Concept du réseau actif*

2.6 Conclusion :

L'avènement d'IPv6 a pu apporter des solutions au problèmes rencontrés avec IPv4 mais aussi il a élaboré de nouvelles fonctionnalités tels que le Multicast.
Le Multicast offre deux apports majeurs, d'abord, il permet un routage efficace dans le cas d'une communication de groupe, il permet aussi une transmission efficace quand les adresses des destinataires sont inconnues ou changent régulièrement.
Le protocole MLD (Multicast Listener Discovery) permet dans ce sens à un routeur de détecter la présence de nœuds sur l'ensemble de ses liens désirant recevoir des paquets relatifs à un groupe multicast.
Vu les apports de la technologie active par rapports à l'approche traditionnelle, il serait plus avantageux d'utiliser une infrastructure active sur laquelle seront mis en œuvre les différents tests du protocole MLD.

Chapitre 3

SPECIFICATIONS DES TESTS POUR MLD v1

3.1 Introduction :

Le protocole MLD étant un protocole asymétrique, l'ensemble des cas de tests peut être scindé en deux sous ensembles [LOU 00]: un sous ensemble pour le routeur et un sous ensemble pour le nœud.
Les tests à réaliser concernent essentiellement le comportement global du protocole MLD dans un environnement d'exécution actif et se sont élaborés en se basant sur la spécification formelle du protocole.

Chaque cas de tests comporte les éléments suivants :

- Un objectif de test qui consiste en une brève description du but du cas de test.

- Une référence qui représente la documentation pouvant être utile pour la compréhension du cas de test.

- Un commentaire qui explique le cas de test. Il est généralement pris des sections de la spécification du protocole à tester (RFC).

- Une configuration de test qui décrit les composants de test ainsi que les composants à tester.

- La procédure qui liste les échanges de messages entre les différentes composantes de la configuration de test ainsi que les résultats à observer.

Tout au long de ce travail, la terminologie suivante sera adoptée:

- RUT : pour routeur sous test.
- NUT : pour nœud sous test.
- TR : pour routeur testeur.
- TN : pour nœud testeur.

3.2 Spécifications des tests à réaliser :

3.2.1 Tests relatifs au routeur :

3.2.1.1 Test 1 : Seul_Routeur_Querier_Par_Lien :

Ø Objectif du test :

Ce test permet de vérifier qu'il ne doit y avoir qu'un seul routeur Querier par lien.

Ø Référence : RFC 2710

Ø Commentaire :

A un instant donné, il ne doit y avoir qu'un seul routeur dans l'état Querier.
A son initialisation, un routeur commence par envoyer des General Query, initialise le timer General Query] et passe à l'état Querier. Il continue à envoyer un message General Query chaque fois que le timer General Query expire.
Une fois qu'un routeur reçoit un General Query d'un autre routeur dont l'adresse IP est inférieure à l'adresse du premier routeur, ce dernier arrête l'envoi des General Query, passe à l'état Non Querier et déclenche le Other Querier Present Timer. Si par contre, ce routeur ne reçoit plus de General Query sur un intervalle de Other Querier Present Timer, il repasse à l'état Querier.

Ø Configuration du test :

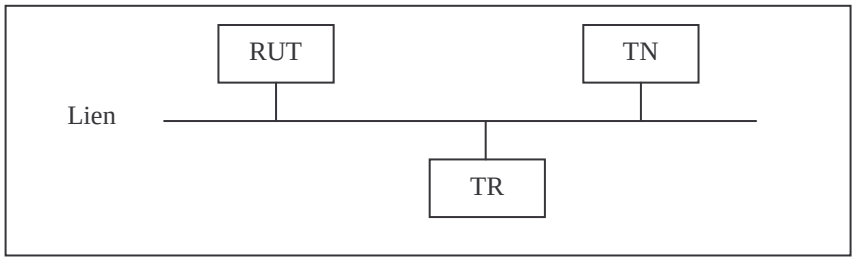


Fig 11 – *Configuration Test Seul Routeur Querier Par Lien*

Ø Procédure :

- Vérifier que le RUT est à l'état Querier.
- Rendre RUT Non Querier
  TR(ayant une adresse IP < à l'adresse IP de RUT) envoie un General Querier et déclenche un timer T.
- Si le TR reçoit un General Query du RUT après l'expiration du timer T, c'est que le RUT ne s'est pas rendu compte qu'il n'est plus Querier, et le résultat du test serait échec. Sinon, le résultat serait succès.

Ø Résultats :

- Le cas de succès consiste à ce que l'un des deux routeurs est à l'état Querier, l'autre est à l'état Non Querier.
- Le cas d'échec consiste à ce que les deux routeurs existant sur le même lien soient à l'état Querier.

3.2.1.2 Test 2 : Routeur_Reception_Report_1 :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur traite correctement le premier Report pour une adresse multicast.

Ø Référence : RFC 2710

Ø Commentaire :

Quand un routeur reçoit un premier Report pour une adresse Multicast, il ajoute cette adresse à la liste des adresses ayant des listeners, initialise un temporisateur à la valeur Multicast Listener Interval, le déclenche et notifie la composante de routage multicast de la présence de listeners pour cette adresse.

Ø Configuration du test :

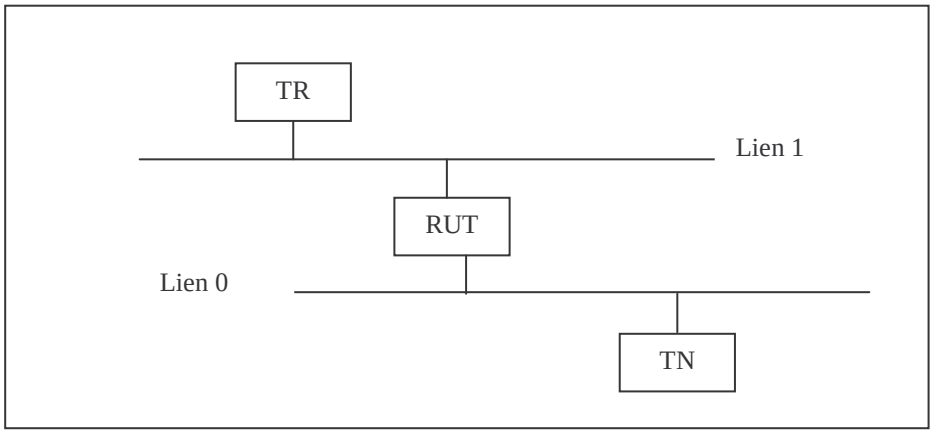


Fig 12 – *Configuration Test Routeur Réception Report 1*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.

- S'assurer que le RUT est à l'état Querier.
- Envoyer un Report sur le lien 0 : TN envoie un Report pour une adresse multicast M.

Ø Résultats :

- TN doit recevoir un General Query dans un intervalle de temps inférieur ou égal à Query Interval.
- TR doit observer un Join sur l'arbre multicast.

3.2.1.3 Test 3 : Routeur_Reception_Report_2 :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur doit réagir correctement à un deuxième Report pour une adresse qui figure dans la liste des adresses multicast ayant des listeners.

Ø Référence : RFC 2710

Ø Commentaire :

Si un routeur reçoit un Report pour une adresse multicast déjà rapportée, le temporisateur est réinitialisé de nouveau à la valeur Multicast Listener Interval.

Ø Configuration du test :

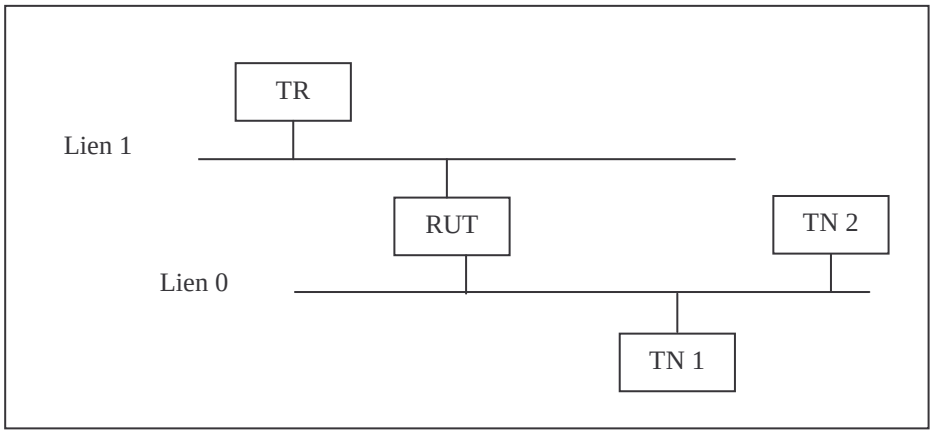


Fig 13 – *Configuration Test Routeur Réception Report 2*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.

- S'assurer que le RUT est à l'état Querier.
- Envoi d'un premier Report pour l'adresse multicast M : TN1 envoie un Report pour l'adresse multicast M.
- Envoi d'un deuxième Report pour la même adresse M sur le lien 0 et vérification que le RUT a bien initialisé le temporisateur relatif à l'adresse M : TN2 doit recevoir le Report envoyé par TN1, il attend un intervalle de temps de Multicast Listener Interval / 4 et envoie un deuxième Report pour la même adresse M.

Ø Résultats :

- TN1 doit recevoir un General Query.
- TR doit recevoir un Join.
- TR ne doit pas recevoir un deuxième Report.

3.2.1.4 Test 4 : Routeur_Reception_Done :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur Querier doit envoyer des Multicast Address Specific Query suite à la réception d'un message Done.

Ø Référence : RFC 2710

Ø Commentaire :

Si un routeur Querier reçoit un Done pour une adresse multicast déjà rapportée, il doit envoyer Last Listener Query Count « MASQ » périodiquement à des intervalles de Last Listener Query Interval pour cette même adresse.

Ø Configuration du test :

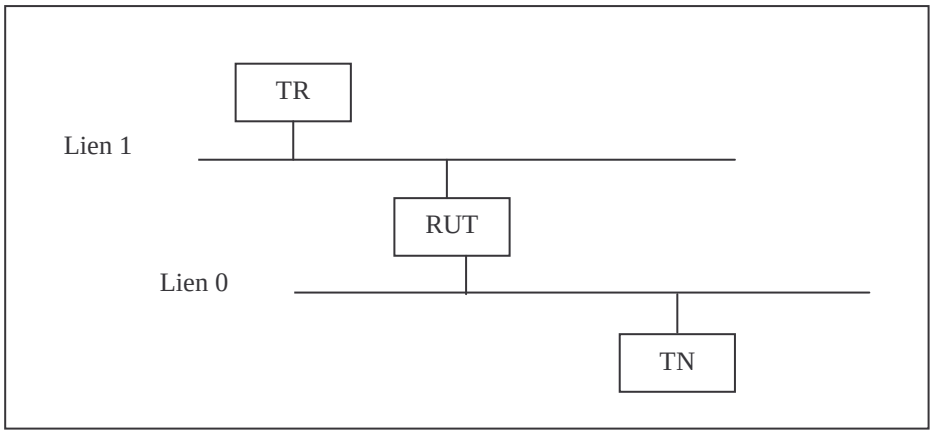


Fig 14 – *Configuration Test Routeur Réception Done*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.

- Vérifier que le RUT est à l'état Querier.
- Envoi d'un Report sur le lien 0
  TN envoie un Report pour l'adresse M.
- Envoi d'un Done
  TN envoie un Done pour l'adresse M après un intervalle de temps Multicast Listener Interval /4.

Ø Résultats :

- TN doit recevoir un General Query de RUT.
- TR doit recevoir un Join.
- TN doit recevoir Last Listener Query Count à des intervalles de Last Listener Query Interval.
- TR doit recevoir un Leave après un intervalle de temps de Last Listener Query Count * Last Listener Query Interval.

3.2.1.5 Test 5 : Routeur_Reception_Done_Report :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur Querier doit arrêter l'envoi des MASQs relatifs à une adresse multicast une fois que celle ci est rapportée.

Ø Référence : RFC 2710

Ø Commentaire :

Un routeur Querier, ayant reçu un message Done pour une adresse multicast M doit envoyer des MASQs. Une fois qu'un Report est reçu pour cette même adresse, le routeur arrête le processus d'envoi des MASQs.

Ø Configuration du test :

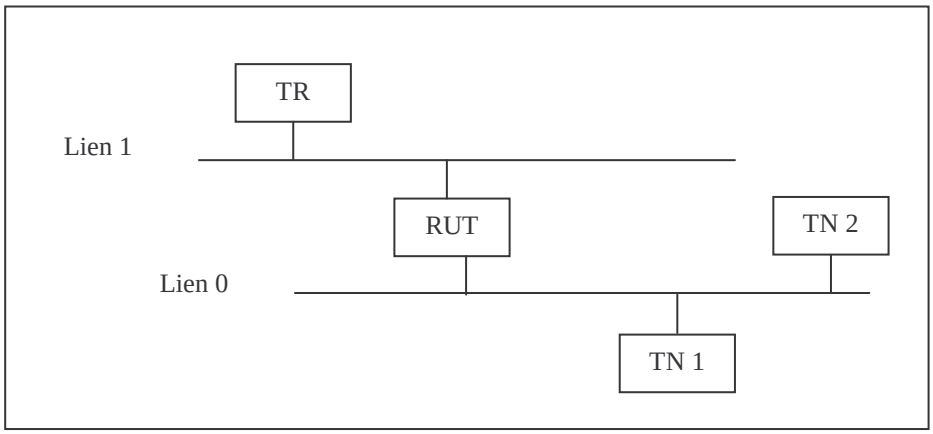


Fig 15 – *Configuration Test Routeur Réception Done Report*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.

- S'assurer que le RUT est à l'état Querier.
- Vérifier que l'adresse multicast figure dans la liste de celles ayant des listeners
  TN1 envoie un Report pour l'adresse M.

- Envoi d'un Done
  TN1 envoie un Done pour l'adresse M après un intervalle de temps de Multicast Listener Interval / 4.
- TN2 envoie un Report pour l'adresse M

Ø Résultats :

- TN1 doit recevoir un General Query de RUT.
- TR doit recevoir un Join.
- TN1 doit recevoir un MASQ.
- TN1 ne doit pas recevoir de MASQ sur un intervalle de Last Listener Query Interval.

3.2.1.6 Test 6 : Routeur_Non_Querier_Reception_Done :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur continue à envoyer des MASQs suite à un message Done même s'il change d'état de Querier à Non Querier.

Ø Référence : RFC 2710

Ø Commentaire :

Un routeur Querier, ayant reçu un message Done pour une adresse multicast M doit continuer à envoyer des messages MASQs pour cette adresse même s'il change d'état (Non Querier).

Ø Configuration du test :

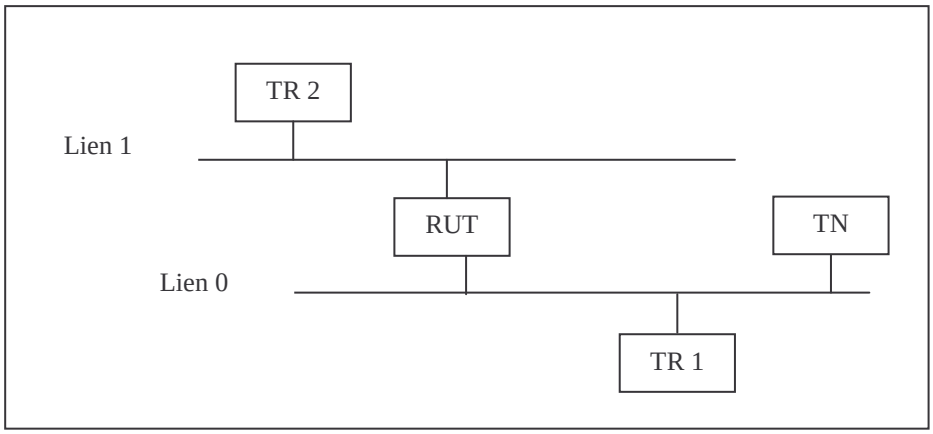


Fig 16 – *Configuration Test Routeur Non Querier Réception Done*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.
- S'assurer que le RUT est à l'état Querier.
- Mettre puis retirer un Listener sur le lien 0
  TN envoie un Report pour l'adresse M.

TN envoie un Done pour l'adresse M après un intervalle de temps Multicast Listener Interval /4.

- RUT devient Non Querier
  TR1 ayant un adresse IP inférieure à celle de RUT envoie des General Query au nombre de Last Listener Query Count, périodiquement sur des intervalles de Last Listener Query Interval.

Ø Résultats :

- TN doit recevoir un General Query de RUT.
- TR2 doit recevoir un Join.
- TN doit recevoir un MASQ.
- TN doit recevoir Last Multicast Count –1 MASQs périodiquement à des intervalles de temps de Last Listener Query Interval.
- Après la réception DU Join, TR2 doit recevoir un Leave après un intervalle de temps inférieur ou égal à Last Listener Query Count * Last Listener Query Interval.

3.2.1.7 Test 7 : Routeur_Non_Querier_MAJ_Liste_Multicast_Address :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur Non Querier doit informer la composante de routage multicast lors de la mise à jour de la liste des adresses multicast (expiration du délai Multicast Listener Interval pour une adresse donnée).

Ø Référence : RFC 2710

Ø Commentaire :

Un routeur Non Querier doit avoir les mêmes états qu'un routeur Querier. La seule différence est qu'un routeur Non Querier ne réagit pas à la réception des événements par l'envoi d'autres messages. Cependant, il doit tenir à jour la liste des adresses multicast ayant des listeners et informer la composante de routage multicast.

Ø Configuration du test :

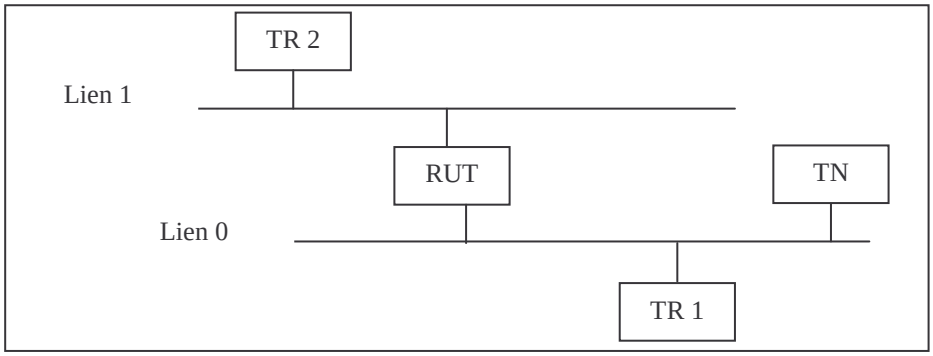


Fig 17 – *Configuration Test Routeur Non Querier MAJ Liste Multicast Address*

Ø Procédure :

Le RUT doit supporter le routage multicast genre PIM-SM.
TR doit faire partie de l'arbre multicast.

- Vérifier que le RUT est à l'état Querier.
- Mettre un listener sur le lien 0
  TN envoie un Report pour l'adresse M.
- TR1 ayant une adresse IP inférieure à celle de RUT envoie des General Query périodiquement sur des intervalles de Multicast Listener Interval.

Ø Résultats :

- TN doit recevoir un General Query de RUT.
- TR1 doit recevoir un Report.
- TR2 doit recevoir un Join.
- TR2 doit recevoir un Leave dans une durée inférieure ou égale à Multicast Listener Interval et ce, à partir de moment où il a reçu un Join.

3.2.1.8 Test 8 : Routeur_Non_Querier_Reception_Done_Report :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur Non Querier doit s'arrêter d'envoyer des MASQs une fois qu'un Report est reçu pour l'adresse figurant dans le champ Multicast Address des MASQs.

Ø Référence : RFC 2710

Ø Commentaire :

Un routeur Non Querier, ayant reçu un Done à l'état Querier, doit envoyer des MASQs. Une fois qu'il reçoit un Report pour la même adresse transportée dans le message MLD de ces MASQs, il doit arrêter leur envoi.

Ø Configuration du test :

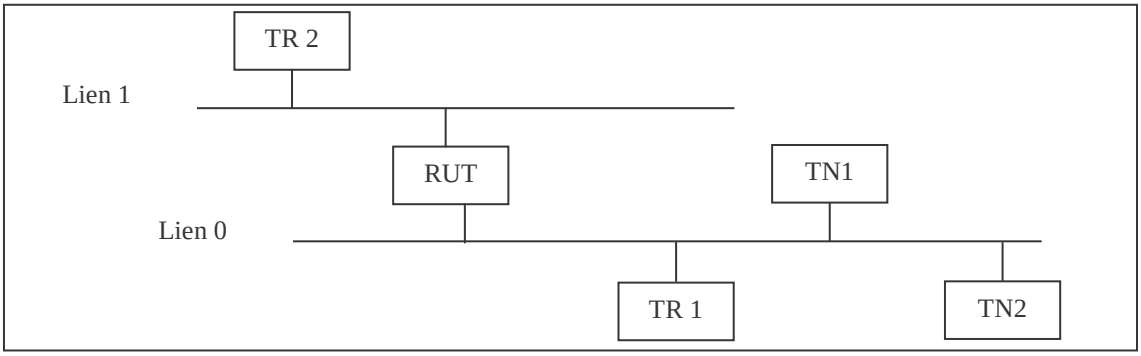


Fig 18 – *Configuration Test Routeur Non Querier Réception Done Report*

Ø Procédure :

- Vérifier que le RUT est à l'état Querier.
- Mettre puis retirer un listener sur le lien 0
  TN1 envoie un Report pour l'adresse M.
  TN1 envoie un Done pour l'adresse M après un intervalle de temps Multicast Listener Interval /4.
- Rendre RUT Non Querier et ajouter un autre listener à l'adresse M sur le lien 0
  TR1 ayant une adresse IP inférieure à celle de RUT envoie des General Query périodiquement sur des intervalles de Multicast Listener Interval.

Ø Résultats :

- TN1 doit recevoir un General Query de RUT.
- TR2 doit recevoir un Join.
- RUT doit envoyer un MASQ à TN1.
- TN1 ne doit pas recevoir d'autres MASQs sur un intervalle de Last Multicast Query Interval.
- TR2 doit recevoir un Leave après un intervalle inférieur ou égal à Last Multicast Query Count * Last Multicast Query Interval et ce, à partir de l'instant où il a reçu un Join.

3.2.1.9 Test 9 : Routeur_Non_Querier_Reception_Done_2 :

Ø Objectif du test :

Ce test permet de vérifier qu'un routeur Non Querier doit continuer à notifier la composante de routage multicast de la présence ou de l'absence de listeners pour une adresse donnée.

Ø Référence : RFC 2710

Ø Commentaire :

Tout en étant à l'état Non Querier, un routeur doit continuer à notifier la composante de routage multicast de la présence ou de l'absence de listeners pour une adresse multicast donnée. Suite à un Report, la composante MLD change d'état en passant de l'état No Listeners Present à Listeners Present et un message Join est envoyé par la composante de routage multicast au routeur le plus proche. Dans le cas où la composante MLD décide de supprimer une adresse de sa liste, elle repasse à l'état No Listeners Present pour cette adresse, et par conséquent, la composante de routage multicast envoie un Leave.

Ø Configuration du test :

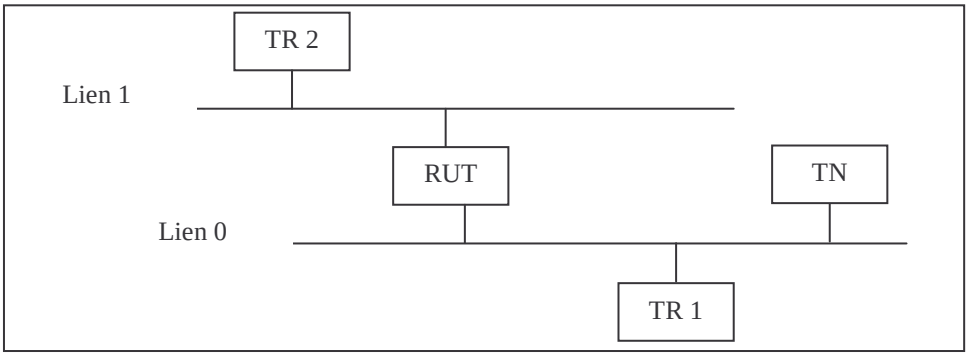


Fig 19 – *Configuration Test Routeur Non Querier Réception Done 2*

Ø Procédure :

- Vérifier que le RUT est à l'état Querier.
- Rendre RUT Non Querier
  TR1 ayant une adresse IP inférieure à celle de RUT envoie des General Query périodiquement sur des intervalles de Multicast Listener Interval.
- Mettre un listener pour l'adresse M sur le lien 0
  TN reçoit le General Query envoyé à TR.
  TN envoie un Report pour l'adresse M.
- Retirer les listeners de l'adresse M sur le lien 0
  TN envoie un Done après un délai de Multicast Listener Interval / 2.

Ø Résultats :

- TN doit recevoir un General Query.
- TR2 doit recevoir un Join.
- TR2 doit recevoir un Leave dans un délai inférieur ou égal à Multicast Listener Interval après la réception de Join.

3.2.2 Tests relatifs au nœud :

3.2.2.1 Test 10 : Mise_En_Ecoute_Nœud :

Ø Objectif du test :

Ce test permet de vérifier qu'un nœud doit envoyer un Report une fois qu'il se met à l'écoute sur une adresse multicast.

Ø Référence : RFC 2710

Ø Commentaire :

Quand un nœud commence à écouter sur une adresse multicast, il doit envoyer un Unsolicited Report pour cette adresse. Dans le cas où le nœud est le premier à émettre ce Report, il doit activer un temporisateur mis à une valeur comprise entre 0 et Maximum Response Delay.

Ø Configuration du test :

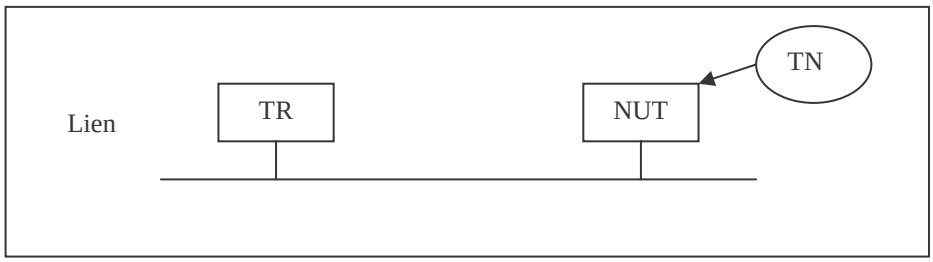


Fig 20 – *Configuration Test Mise En Ecoute Nœud*

Ø Procédure :

- Vérifier que le NUT est à l'état No Listener pour l'adresse M.
- TR envoie régulièrement des General Query.
- Mettre le NUT à l'écoute : TN met le NUT à l'écoute sur l'adresse M, le NUT passe donc à l'état Delaying Listener avec le flag mis à 1.

Ø Résultats :

- TR doit recevoir un premier Report pour l'adresse M.
- TR doit recevoir un deuxième Report pour l'adresse M, dans un intervalle de temps inférieur ou égal à Maximum Response Delay.
- Le cas de succès consiste à ce que les résultats pratiques soient conformes aux prévisions théoriques.
- Le cas d'échec consiste à ce que les résultats pratiques soient différents des prévisions théoriques.

3.2.2.2 Test 11 : Cesser_Ecoute_Nœud_1 :

Ø Objectif du test :

Ce test permet de vérifier qu'un nœud doit envoyer un Done une fois qu'il cesse d'écouter à une adresse multicast.

Ø Référence : RFC 2710

Ø Commentaire :

Quand un nœud cesse d'écouter à une adresse multicast, il doit envoyer un Done pour cette adresse dans le cas où il n'a pas reçu de récent Report sur cette même adresse.

Ø Configuration du test :

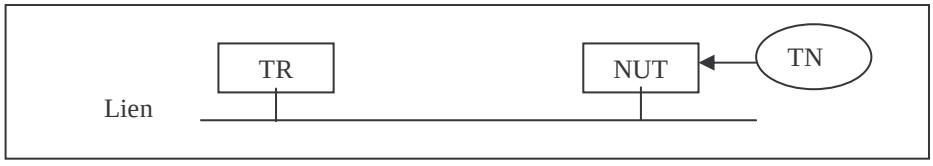


Fig 21 – *Configuration Test Cesser Ecoute Nœud 1*

Ø Procédure :

- Vérifier que le NUT est à l'état No Listener pour l'adresse M.
- TR envoie régulièrement des General Query.
- Mettre le NUT à l'écoute : TN met le NUT à l'écoute sur l'adresse M, le NUT passe donc à l'état Delaying Listener avec le flag mis à 1.
- Le NUT cesse d'écouter à l'adresse M.

Ø Résultats :

- TR doit recevoir un Done pour l'adresse M quand le NUT cesse d'écouter à cette même adresse.
- Le cas de succès consiste à ce que les résultats pratiques soient conformes aux prévisions théoriques.
- Le cas d'échec consiste à ce que les résultats pratiques soient différents des prévisions théoriques.

3.2.2.3 Test 12 : Cesser_Ecoute_Nœud_2 :

Ø Objectif du test :

Ce test permet de vérifier qu'un nœud ne doit pas émettre de Done s'il reçoit un Report pour l'adresse à laquelle il veut cesser d'écouter.

Ø Référence : RFC 2710

Ø Commentaire :

Si un nœud cesse d'écouter à une adresse M sachant qu'il a reçu un récent Report pour cette adresse, il ne doit pas envoyer de Done pour cette même adresse.

Ø Configuration du test :

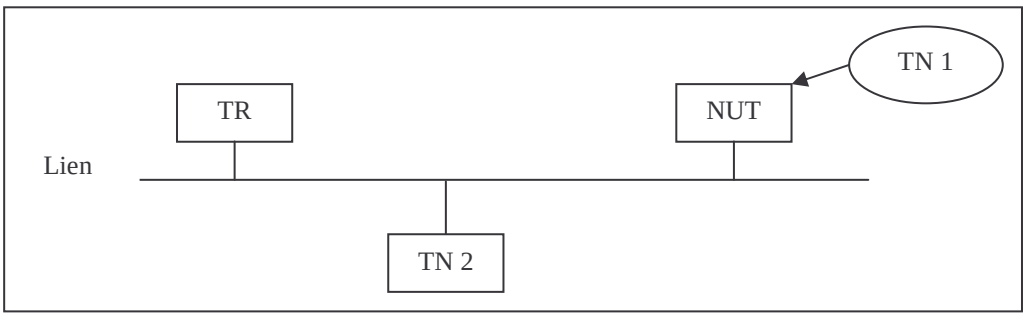


Fig 22 – *Configuration Test Cesser Ecoute Nœud 2*

Ø Procédure :

- Vérifier que le NUT est à l'état No Listener pour l'adresse M.
- TR envoie régulièrement des General Query.

- Mettre le NUT à l'écoute : TN1 met le NUT à l'écoute sur l'adresse M, le NUT passe donc à l'état Delaying Listener avec le flag mis à 1.
- Mettre TN2 à l'écoute sur l'adresse multicast M.
- Cesser TN1 de l'écoute à l'adresse M.

Ø Résultats :

- TR ne doit pas recevoir de Done pour l'adresse M quand le NUT cesse d'écouter à cette même adresse.
- Si le TN2 cesse d'écouter à l'adresse M, il envoie un Done au TR.
- Le cas de succès consiste à ce que les résultats pratiques soient conformes aux prévisions théoriques.
- Le cas d'échec consiste à ce que les résultats pratiques soient différents des prévisions théoriques.

3.2.2.4 Test 13 : Etat_Idle_Listener :

Ø Objectif du test :

Ce test permet de vérifier qu'un nœud doit changer d'état de Idle Listener à Delaying Listener une fois qu'il a reçu un Query.

Ø Référence : RFC 2710

Ø Commentaire :

Quand un nœud se met à l'écoute à une adresse multicast, il envoie un Report et passe à l'état Delaying Listener. S'il reçoit un autre Report pour cette même adresse, il passe à l'état Idle listener. Une fois qu'un General Query est reçu, il doit repasser à l'état Delaying Listener.

Ø Configuration du test :

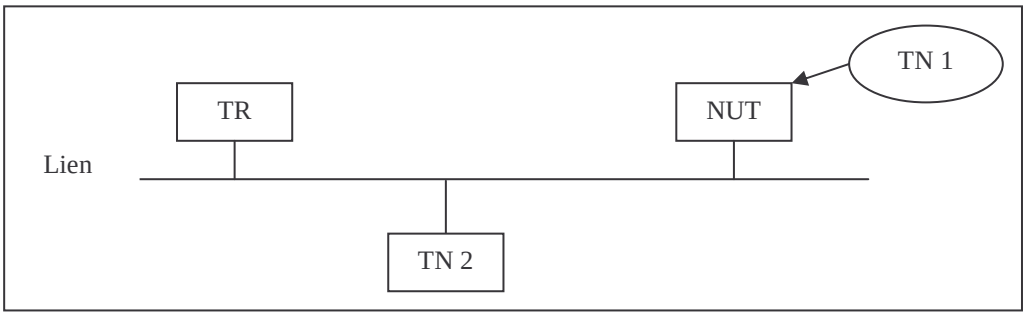


Fig 23 – *Configuration Test Etat Idle Listener*

Ø Procédure :

- Mettre NUT à l'écoute.
  TN1 met NUT à l'écoute sur l'adresse M.
- Emettre un deuxième Report pour l'adresse M
  TN2 envoie un deuxième Report pour l'adresse M
- Envoi d'un Query sur le lien
  TR envoie un General Query.

Ø Résultats :

- NUT doit envoyer un Report pour l'adresse M.
- TR doit recevoir un Report.
- TR reçoit le deuxième Report.
- TR doit recevoir un Report de NUT dans un intervalle de temps inférieur ou égal à Multicast Listener Interval.

3.2.3 Tests de Robustesse :

3.2.3.1 Test 14 : Reception_Massive_Query_1 :

Ø Objectif du test :

Ce test permet d'observer le comportement d'un nœud non mis à l'écoute d'une adresse multicast qui est bombardé par des messages de General Query du routeur Querier du même lien.

Ø Configuration du test :

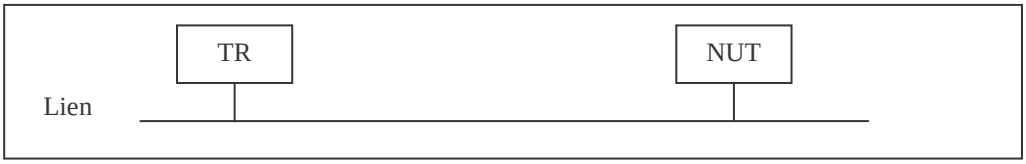


Fig 24 – *Configuration Test Réception Massive Query 1*

Ø Procédure :

TR envoie des General Query de façon massive sur le lien local pendant une durée déterminée de temps.

Ø Résultats :

Le NUT, ayant reçu les premiers General Query, doit ignorer le flux massif de General Query sinon il sera bombardé et planté.

3.2.3.2 Test 15 : Réception_Massive_Query_2 :

Ø Objectif du test :

Ce test permet d'observer le comportement d'un nœud mis à l'écoute d'une adresse multicast qui est bombardé par des messages de General Query du routeur Querier du même lien.

Ø Configuration du test :

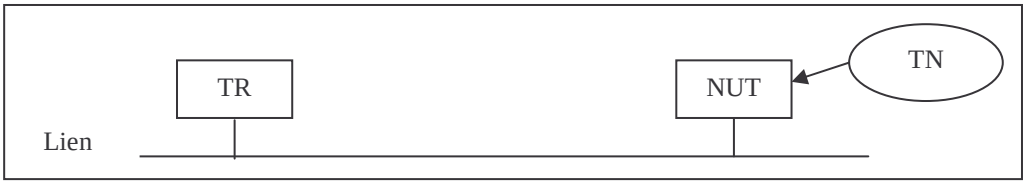


Fig 25 – *Configuration Test Réception Massive Query 2*

Ø Procédure :

- TN met le NUT à l'écoute sur une adresse multicast M.
  TN envoie des General Query de façon massive sur le lien local pendant une durée déterminée de temps.

Ø Résultats :

Le NUT, ayant reçu les premiers General Query, doit envoyer des messages Report puis à un moment donné il doit ignorer le flux massif de General Query sinon il sera bombardé et planté.

3.3 Conclusion :

La génération de tests ci dessus et bien d'autres est réalisée d'une façon informelle en se basant essentiellement sur la spécification du protocole et en essayant à chaque fois de tester la présence ou l'absence d'une fonctionnalité spécifiée et sa conformité par rapport à ce qui était prévu.
L'utilisateur d'une suite de test pose toujours la question concernant la couverture de tests qui donne la possibilité de déclarer un protocole conforme.
La couverture peut être mesurée comme le rapport du nombre des états rencontrés lors de l'exécution de la suite des tests et du nombre d'états de la spécification. Elle peut aussi être mesurée comme le rapport du nombre des transitions passées dans le test et du nombre total de transitions de la spécification.

Chapitre 4

CONCEPTION D'UN ENVIRONNEMENT DE TESTS

4.1 Introduction :

La mise au point de tests sur le protocole MLD peut être élaborée suivant deux approches : approche non active et approche active.
Tout au long de ce chapitre, sera présentée l'approche non active et l'approche active puis les critères de choix de la solution retenue et ses contributions à la bonne mise au point des tests du protocole MLD.

4.2 Approche non active :

L'approche non active est basée sur les réseaux traditionnels qui comportent un nombre restreint et fixe de services implantés dans les équipements et qui n'offrent aucun moyen d'en ajouter de nouveaux.
En conséquence, l'approche non active ne permet pas de modifier dynamiquement le comportement global du réseau.
Ainsi, des tests MLD réalisés suivant l'approche non active seront statiques et ne présenteront au testeur aucun aspect dynamique. Chaque scénario de test doit être manuellement installé sur les machines choisies et le testeur ne pourrait en aucun cas superviser tous les tests via une seule machine console.

4.3 Approche active :

Comme est présenté dans le chapitre Etude de l'existant, l'approche active se base sur le fait que tout ou une partie des composants d'un réseau actif dans les différents plans (signalisation, supervision et données) sont programmables dynamiquement par des entités tierces (opérateur, fournisseur de services, applications, usagers).
Ainsi, la contribution d'une infrastructure active pour la réalisation des tests MLD aura beaucoup d'avantages :

- Le déploiement dynamique du code. En effet, le code des différents tests peut être téléchargé d'une machine console et exécuté sans aucune installation manuelle.

- Capacité de télécharger les scénarios de tests à n'importe quel endroit du réseau.

- Souplesse et facilité de réalisation des tests du protocole pour l'administrateur testeur, en effet, à partir d'une machine console, il peut superviser tous les scénarios de tests qu'il a mis en place à n'importe quel endroit du réseau.

- Un test MLD aura la forme d'une sonde programmable qui exécute le scénario de test chargé d'une machine de contrôle, obtient les résultats du test, les affiche sur l'écran du même hôte de contrôle et enfin s'achève.

4.4 Solution retenue : FLAME :

Vu les avantages de l'approche active par rapport à l'approche traditionnelle pour la réalisation des tests de MLD, on a opté pour l'approche active.
Plusieurs travaux de recherche ont été menés [OLI 00] pour concevoir des architectures de réseau actif, parmi ces architectures on peut citer : SmartPackets développée chez BBN, Switchware développée chez SMI et ALE, Active IP développée chez MIT, ANTS suite de Active IP…
L'environnement d'exécution qui sera utilisé pour mettre en place les scénarios de tests MLD est le FLAME. FLAME est en fait un environnement d'exécution actif développé par RESEDAS en 2001 [Annexe A].

Le but de l'environnement d'exécution FLAME est d'avoir un environnement actif simple et facile pour gérer tout ce qui est IP.

Les tests MLD réalisés sous l'environnement FLAME suivront le mode de fonctionnement suivant :

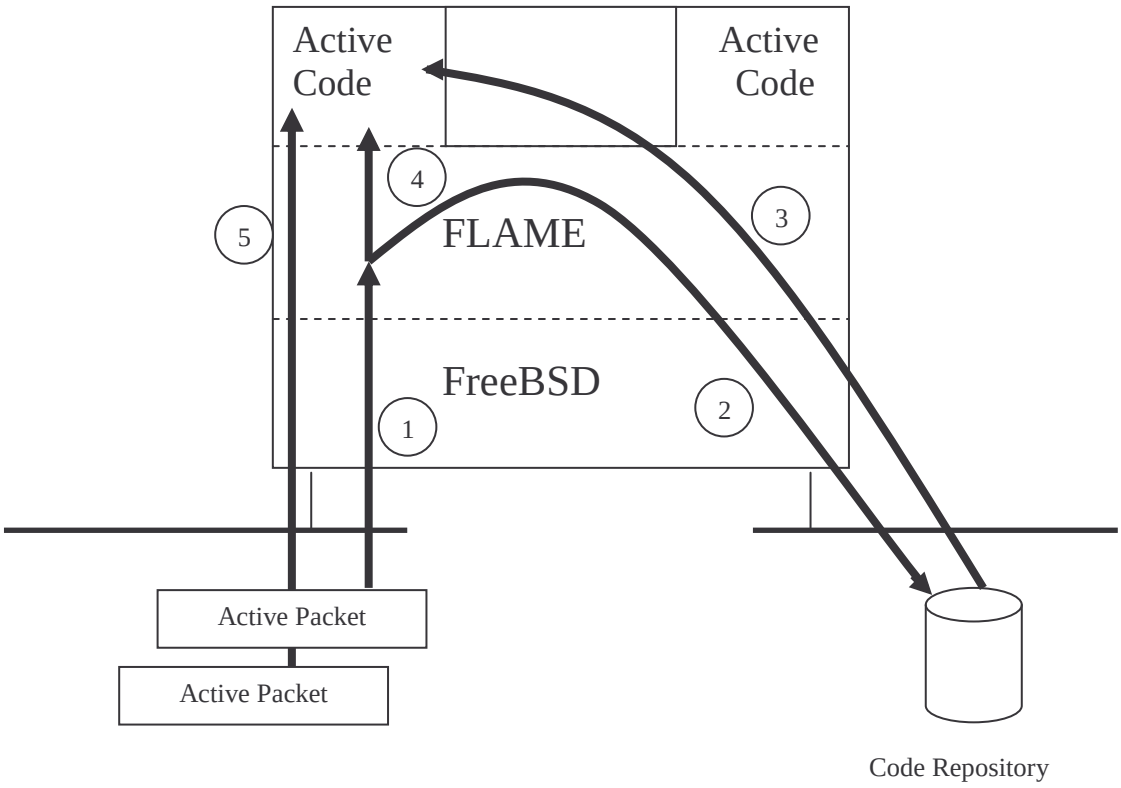


Fig 26- *Mode de fonctionnement de FLAME*

Lorsque l'environnement d'exécution actif FLAME reçoit un paquet actif (Transition 1), il va identifier l'application active appropriée qui va recevoir ce paquet, et va la charger depuis le serveur de code (Transitions 2 et 3). Une fois l'application active chargée, FLAME va lui acheminer le paquet actif correspondant (Transition 4). Si FLAME reçoit le même paquet actif après un bout de temps, il va l'acheminer directement à l'application active déjà chargée en mémoire (Transition 5).

L'environnement FLAME peut être décomposé en 6 unités [OLI 01]:

- Active Packets Receiver : dédié à la réception de paquets actifs.

- User Agent Server : peut être vu comme un serveur telnet, il permet des connexions à l'environnement d'exécution, il procède à l'exécution d'une commande directement ou l'envoi à l'application active correspondante.

- AAManager : crée et détruit des applications actives, il envoie les paquets et les requêtes à l'application active appropriée.

- CodeLoader : télécharge le code de l'application active et fait quelques vérifications.

- Code Server : un simple serveur http permettant la distribution du code à travers le réseau.

- AA : Active Application.

L'architecture de l'environnement FLAME est schématisée ci dessous :

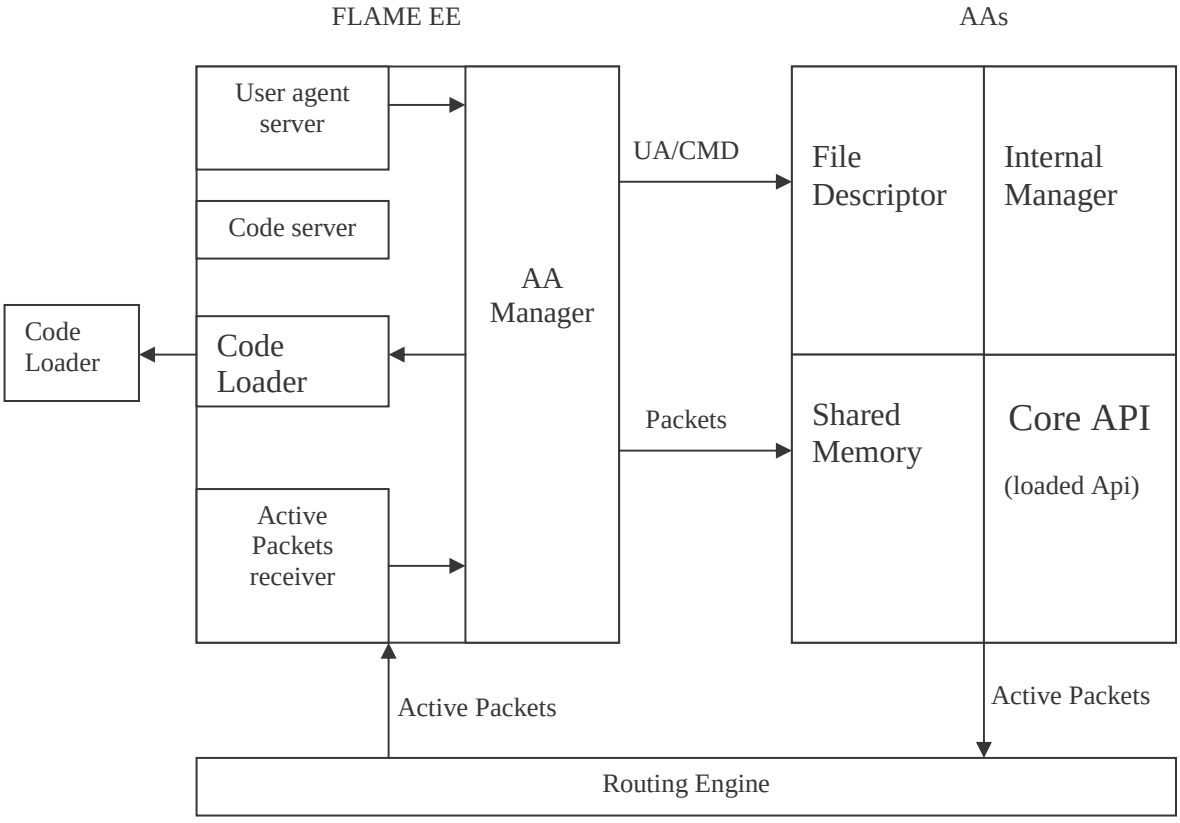


Fig 27 – *Architecture FLAME*

Un test du protocole MLD sera sous forme d'une application active. Le code de chaque test est téléchargé à la machine souhaitée depuis le serveur de code ; l'application active test étant chargée, elle peut s'exécuter sur cette machine et ainsi tester d'autres machines cibles.
Comme chaque application active, le processus d'un test du protocole MLD comprend les quatre points d'entrées suivants [Annexe A]:

- Initialisation : Cette fonction initialise les variables globales, les mutex et déclarent les API qui vont être utilisés par l'application active. L'API MLD réalisée sera chargée lors de l'initialisation de l'application test pour pouvoir être utilisée au cours des autres fonctions.

- Paket handler : Cette fonction permet la lecture des paquets actifs reçus par l'application active. Cette fonction n'est pas obligatoire pour les tests du protocole MLD puisque les paquets sont reçus par une Raw socket [Annexe B].

- User Agent : Cette fonction offre à l'utilisateur un moyen pour configurer et commencer le test souhaité.

- Aamain : Cette fonction permet d'avoir un thread permanent au sein de l'application active. Elle est souvent consacrée à la réception des messages MLD et à l'émission périodique des General Query par les routeurs MLD.

En plus des apports des réseaux actifs par rapport aux réseaux traditionnels, l'environnement d'exécution FLAME offre un autre avantage qui est la sécurité, en effet :

- FLAME permet un accès limité des ressources par l'application active en restreingnant la liste des API et en contrôlant la mémoire et la Cpu qu'elle utilise.

- FLAME est protégé aussi contre le code erroné, en effet l'environnement d'exécution est capable de détecter si une application active a réussi ou échoué. Ainsi, il pourrait marquer une application active qui échoue souvent.

Envoi et réception des messages MLD :

L'envoi des différents messages MLD se fait via des Raw socket[Annexe B]. Ce type de socket est le plus adapté [RIC 98]. En effet, les Raw sockets offre trois aspects qui ne sont pas fournies par des sockets TCP ou UDP.
- Raw socket nous permet de lire et écrire des paquets ICMPv4, IGMPv4 et ICMPv6. Elle permet aussi aux applications basées sur l'utilisation de ICMP ou IGMP de se construire entièrement par des processus utilisateurs.
- Avec une Raw socket, un processus peut lire ou écrire un datagramme IPv4 avec un champ protocole IPv4. Le RFC 1700 liste toutes les valeurs du champ protocole. Par exemple, le protocole de routage OSPF n'utilise pas TCP ou UDP mais utilise IP directement, le champ protocole dans le datagramme a dans ce cas comme valeur 89.
- Avec une Raw socket, un processus peut construire son propre IPv4 Header, en utilisant l'option socket IP_HDRINCL. Ceci peut être utilisé pour construire nos propres paquets UDP ou TCP.

4.5 Conclusion :

Deux approches pour la réalisation des suites de tests MLD sont possibles : approche non active et approche active.
Cependant, l'approche active présente beaucoup d'avantages et contribue au mieux au bon déroulement des tests MLD, c'est ainsi que la solution retenue serait d'utiliser une infrastructure active capable de gérer au mieux les tests MLD.
L'environnement d'exécution actif utilisé pour réaliser les tests est l'environnement FLAME réalisé dans RESEDAS. Cet environnement est adéquat pour mettre en œuvre les tests MLD, chaque test aura la forme d'une application active capable de s'exécuter sur n'importe quelle machine du réseau un fois chargée depuis le serveur de code.

Chapitre 5

Réalisation

5.1 Introduction :

Pour réaliser les tests MLD sur l'environnement d'exécution FLAME, nous avions à notre disposition une plate forme FreeBsd 4.5, composée d'un nombre de nœuds et de routeurs et supportant IPv6.
Les machines utilisées pour les tests sont DELL Pentium 2 et 3.
Le nombre de lignes de code est 1772.

La structure de la salle IPv6 est la suivante :

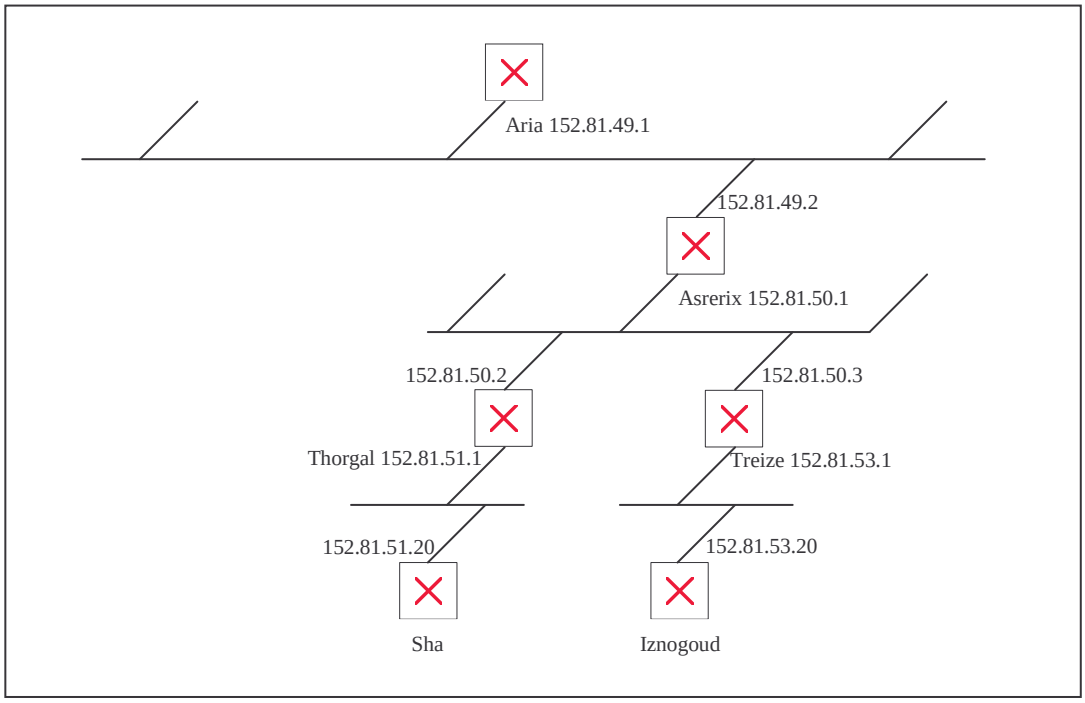


Fig 28 – *Architecture de la salle IPv6*

5.2 Implémentation des tests :

5.2.1 Implémentation Test 1 : Seul_Routeur_Querier_Par_Lien :

5.2.1.1 Objectif du test :

Vérifier qu'il ne doit y avoir qu'un seul routeur Querier par lien.

5.2.1.2 Approche FLAME :

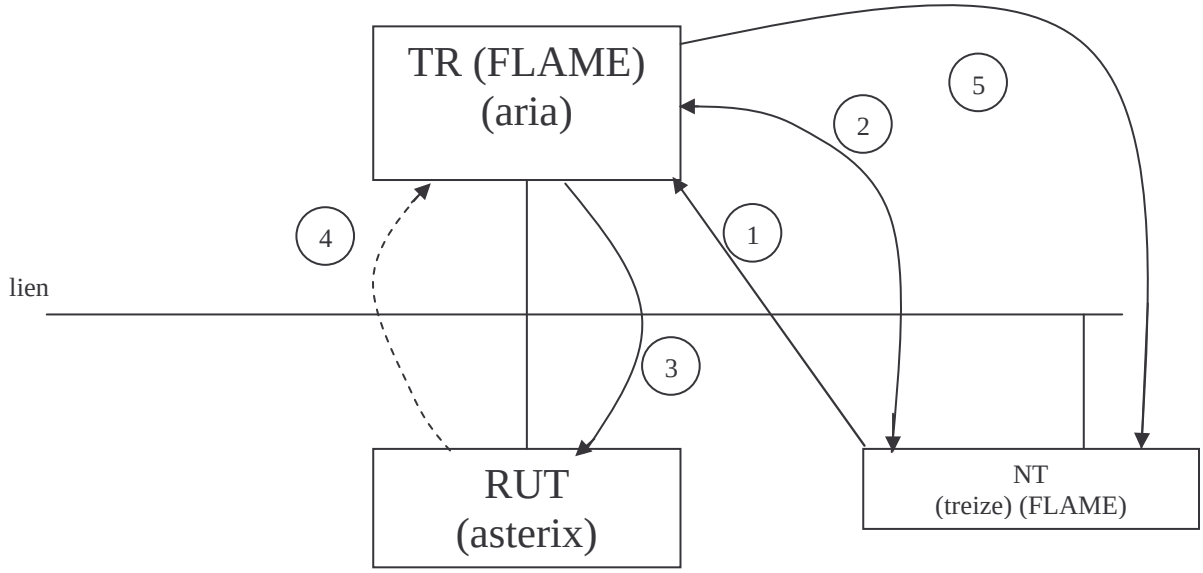


Fig 29 – *Approche FLAME TEST Seul Routeur Querier Par Lien*

5.2.1.3 Procédure du test :

1- Telnet sur le routeur testeur.
2- Le routeur testeur charge le code de l'application active correspondante depuis la machine treize.
3- Le routeur testeur envoie un General Query au routeur à tester et déclenche le timer [Other Querier Present Interval] , puis envoie un autre General Query après une période de [Query Interval].
4- Si le routeur testeur reçoit un General Query du routeur à tester après l'expiration du timer, c'est l'échec du test, sinon, c'est un succès.
5- Affichage du résultat du test.

5.2.1.4 Exemple d'exécution :

telnet aria 6666 :
connected to aria

cmd> uagent {a = test_mld_1 ; v = 1 ; p = http://treize :6667}
routeur à tester : asterix

résultat : Succès : un seul routeur est à l'état Querier.
Echec : il y a eu deux routeurs à l'état Querier.

5.2.2 Implémentation Test 10 : Mise_En_Ecoute_Noeud :

5.2.2.1 Objectif du test 2:

Ce test permet de vérifier qu'un nœud doit envoyer un Report une fois qu'il se met à l'écoute sur une adresse multicast.

5.2.2.2 Approche FLAME :

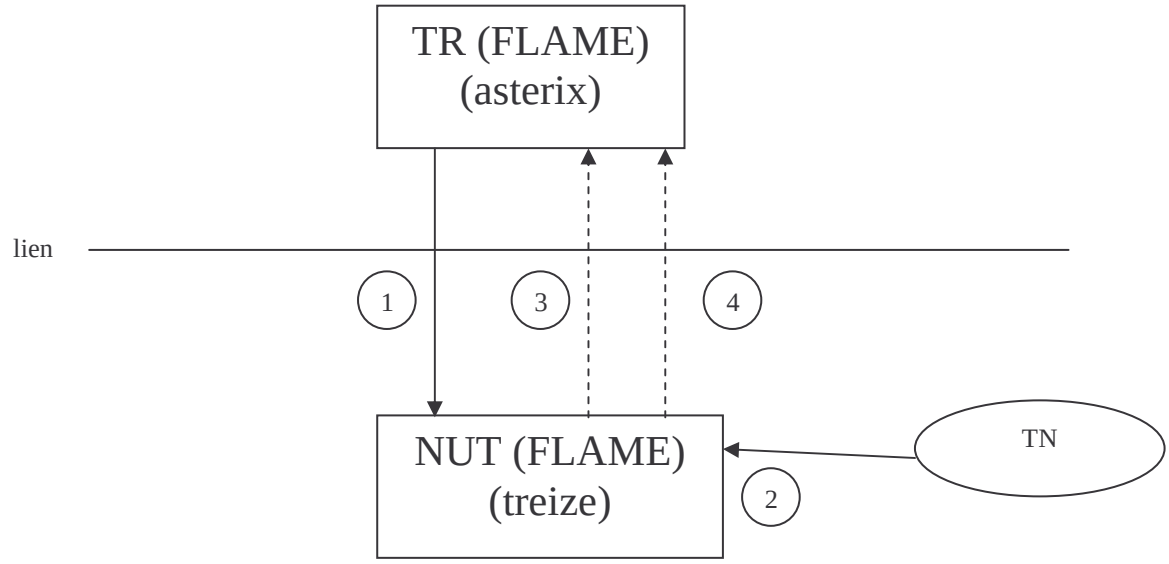


Fig 30 – *Approche FLAME Test Mise En Ecoute Nœud*

5.2.2.3 Procédure du test :

1- Le TR envoie régulièrement des General Query au nœud à tester.
2- Le TN met le nœud à tester sur écoute d'une adresse multicast M.
3- Le NUT doit envoyer un premier Report au TR.
4- Le NUT doit envoyer un deuxième Report au TR avant l'expiration du Maximum Response Delay.

5.2.2.4 Exemple d'exécution :

telnet asterix 6666 :
connected to asterix

cmd> uagent {a = test_mld_2 ; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

résultat : Succès : le comportement du nœud est correct.
Echec : le comportement du nœud est incorrect.

telnet treize 6666
connected to treize

cmd> uagent {a = mise_en_ecoute ; v = 1 ; p = http://treize:6667}

Adresse de diffusion à joindre : ffff::f0ff

5.2.3 Implémentation Test 11 : Cesser_Ecoute_Nœud_1 :

5.2.3.1 Objectif du test :

Ce test permet de vérifier qu'un nœud doit envoyer un Done une fois qu'il cesse d'écouter à une adresse multicast.

5.2.3.2 Approche FLAME :

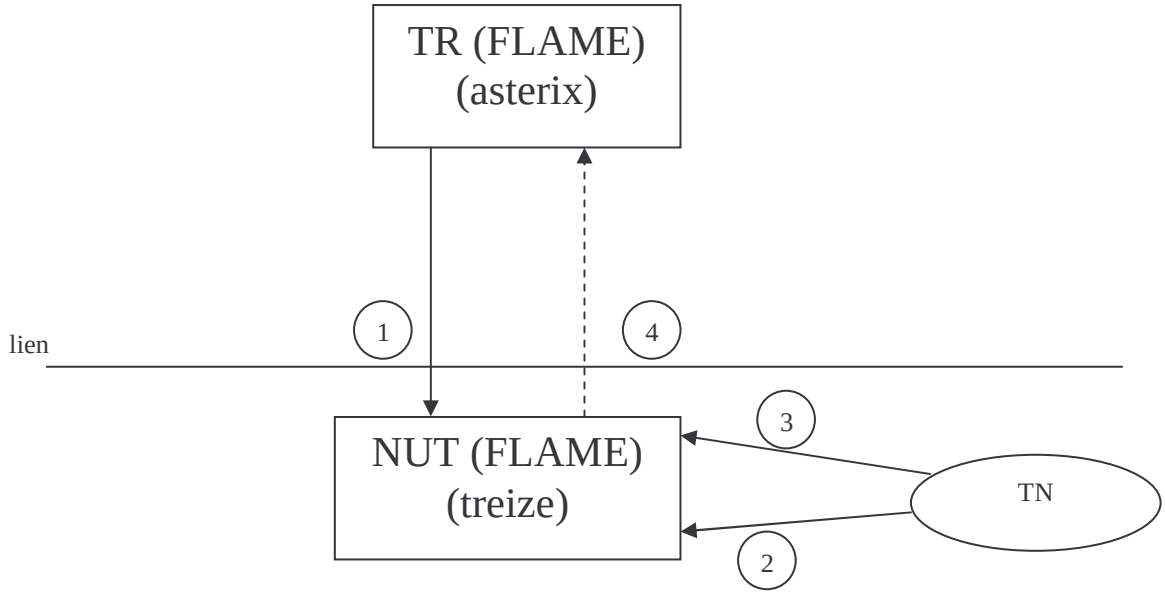


Fig 31 – *Approche FLAME Test Cesser Ecoute Nœud 1*

5.2.3.3 Procédure du test :

1- Le TR envoie régulièrement des General Query au nœud à tester.
2- Le TN met le nœud à tester sur écoute d'une adresse multicast M.
3- Le TN cesse l'écoute du nœud à tester sur la même adresse M.
4- Le NUT doit envoyer un Done (il n'a pas reçu un récent Report sur cette même adresse).

5.2.3.4 Exemple d'exécution :

telnet asterix 6666 :
connected to asterix

cmd> uagent {a = test_mld_3; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

résultat : Succès : le comportement du nœud est correct.
Echec : le comportement du nœud est incorrect.

telnet treize 6666
connected to treize

cmd> uagent {a = mise_en_ecoute ; v = 1 ; p = http://treize:6667}

Adresse de diffusion à joindre : ffff::f0ff

telnet treize 6666
connected to treize

cmd> uagent {a = arret_ecoute ; v = 1 ; p = http:// treize:6667}

Adresse de diffusion à quitter : ffff::f0ff

5.2.4 Implémentation du test 12 : Cesser_Ecoute_Nœud_2 :

5.2.4.1 Objectif du test :

Ce test permet de vérifier qu'un nœud ne doit pas émettre de Done s'il reçoit un Report pour l'adresse à laquelle il veut cesser d'écouter.

5.2.4.2 Approche FLAME :

Exemple d'architecture :

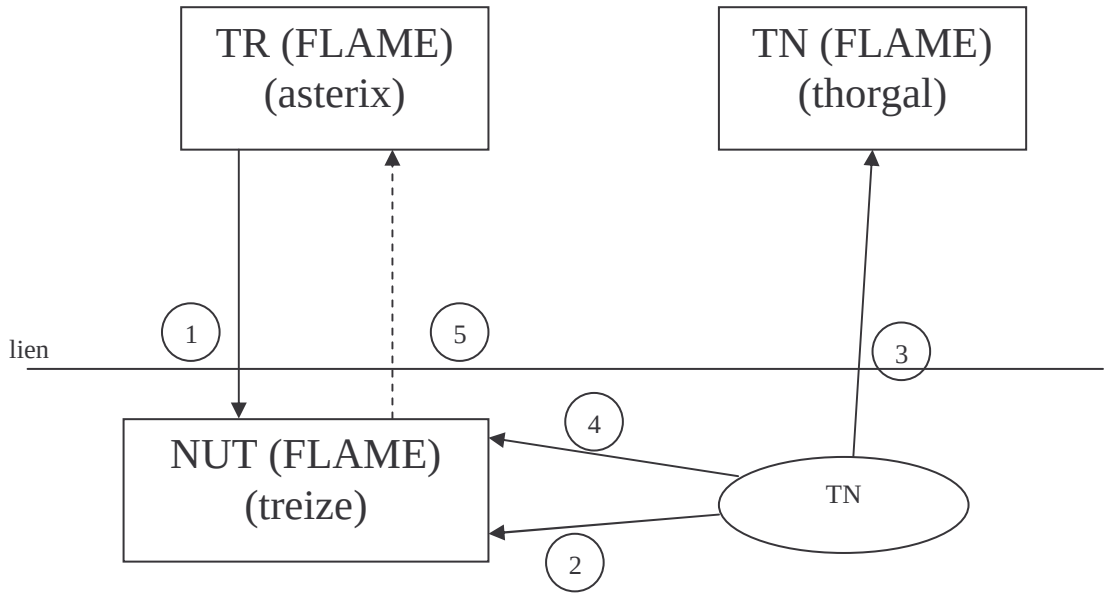


Fig 32 – *Approche FLAME Test Cesser Ecoute Nœud 2*

5.2.4.3 Procédure du test :

1- Le TR envoie régulièrement des General Query au nœud à tester.
2- Le TN met le nœud à tester à l'écoute d'une adresse multicast M.
3- Le TN met le nœud testeur à l'écoute à la même adresse multicast M
4- Le TN cesse l'écoute du nœud à tester à la même adresse M.
5- Le NUT ne doit pas envoyer un Done car le nœud testeur est à l'écoute sur l'adresse M.

5.2.4.4 Exemple d'exécution:

telnet asterix 6666 :
connected to asterix

cmd> uagent {a = test_mld_4; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

résultat : Succès : le comportement du nœud est correct.
Echec : le comportement du nœud est incorrect.

telnet treize 6666
connected to treize

cmd> uagent {a = mise_en_ecoute ; v = 1 ; p = http://treize:6667}

Adresse de diffusion à joindre : ffff::f0ff

telnet thorgal 6666
connected to thorgal

cmd> uagent {a = mise_en_ecoute ; v = 1 ; p = http://treize:6667}

Adresse de diffusion à joindre : ffff::f0ff

telnet treize 6666
connected to treize

cmd> uagent {a = arret_ecoute ; v = 1 ; p = http://treize:6667}

Adresse de diffusion à quitter : ffff::f0ff

5.2.5 Implémentation du test 14 : Réception_Massive_Query_1 :

5.2.5.1 Objectif du test :

Ce test permet d'observer le comportement d'un nœud non mis à l'écoute d'une adresse Multicast qui est bombardé par des messages de General Query du routeur Querier sur le même lien. Ce nœud doit ignorer le flux massif de General Query.

5.2.5.2 Approche FLAME :

Exemple d'architecture :

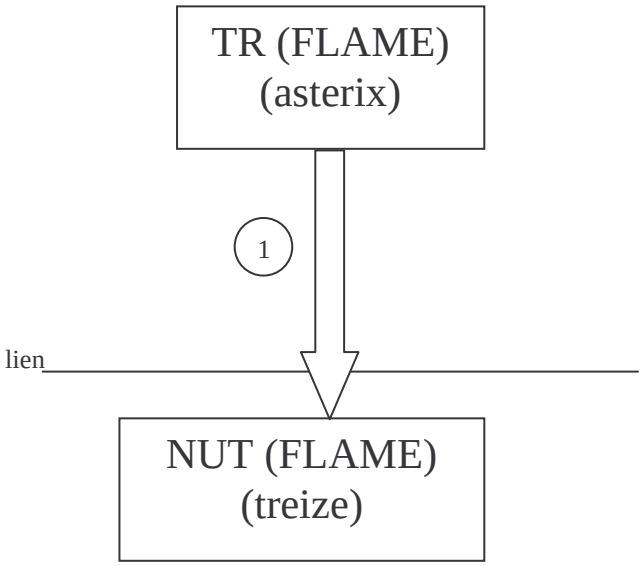


Fig 33 – *Approche FLAME Test Réception Massive Query 1*

5.2.5.3 Procédure du test :

Le TR bombarde le NUT avec des General Query pendant une période de temps.

5.2.5.4 Exemple d'exécution:

telnet asterix 6666 :
connected to asterix

cmd> uagent {a = bombarder ; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

5.2.6 Implémentation du test 15 : Réception_Massive_Query_2 :

5.2.6.1 Objectif du test :

Ce test permet d'observer le comportement d'un nœud mis à l'écoute d'une adresse Multicast qui est bombardé par des messages de General Query du routeur Querier du le même lien.
Ce nœud doit envoyer des Report périodiquement et ignorer le flux massif de General Query.

5.2.6.2 Approche FLAME :

Exemple d'architecture :

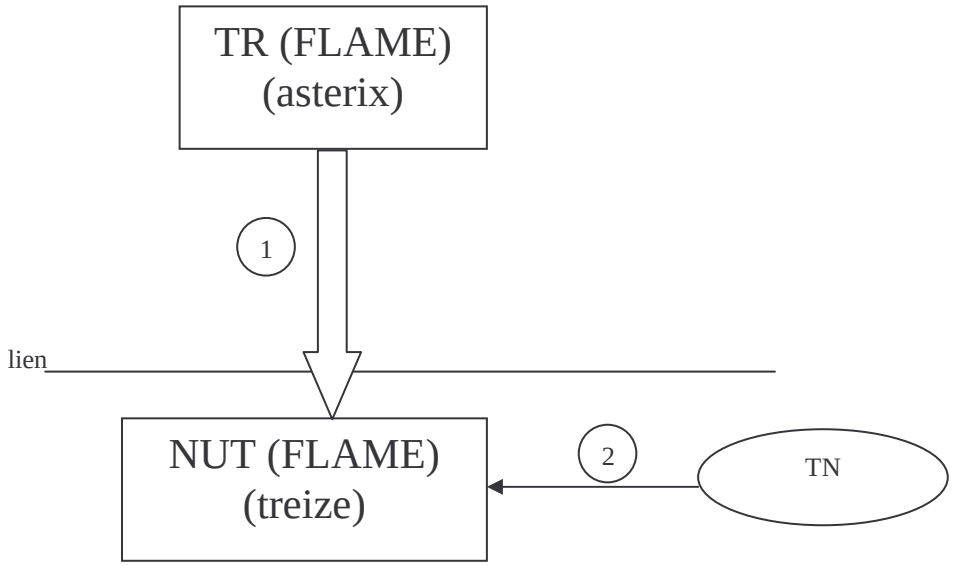


Fig 34 – *Approche FLAME Test Réception Massive Query 2*

5.2.6.3 Procédure du test :

Le TR bombarde le NUT avec des General Query pendant une période de temps, et en même temps le TN met le NUT à l'écoute d'une adresse multicast M.

5.2.6.4 Exemple d'exécution:

telnet asterix 6666 :
connected to asterix

cmd> uagent {a = bombarder ; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

telnet treize 6666 :
connected to asterix

cmd> uagent {a = mise_en_ecoute ; v = 1 ; p = http://treize:6667}

Nœud à tester : treize

5.3 Test générique :

L'implémentation des tests précédents concernant l'observation d'un nœud MLD est dépendante de la plate forme utilisée. En effet, le nœud MLD cible doit avoir le système d'exploitation FreeBSD et l'environnement d'exécution actif FLAME.
Pour migrer vers une approche plus générique, le nœud à tester ne doit pas être accessible et donc on pourra pas y installer FLAME pour le faire joindre ou le faire quitter un groupe.
D'où la nécessité d'implémenter une application complémentaire : un répondeur sur l'environnement cible [DAV 98]. Le répondeur devra être le plus portable possible pour faciliter son adaptation aux différents environnements de développements (FreeBSD, Windows, Linux,…) et aussi pour tester l'interopérabilité entre les différentes plates formes.
Le langage de programmation choisi pour implémenter le répondeur est java version jdk1.4. Cette version supporte IPv6 et est portable sur les différentes plates formes.
Le test qu'on va modifier pour insérer le répondeur est Test Mise_Ecoute_Nœud ; l'architecture de test serait la suivante :

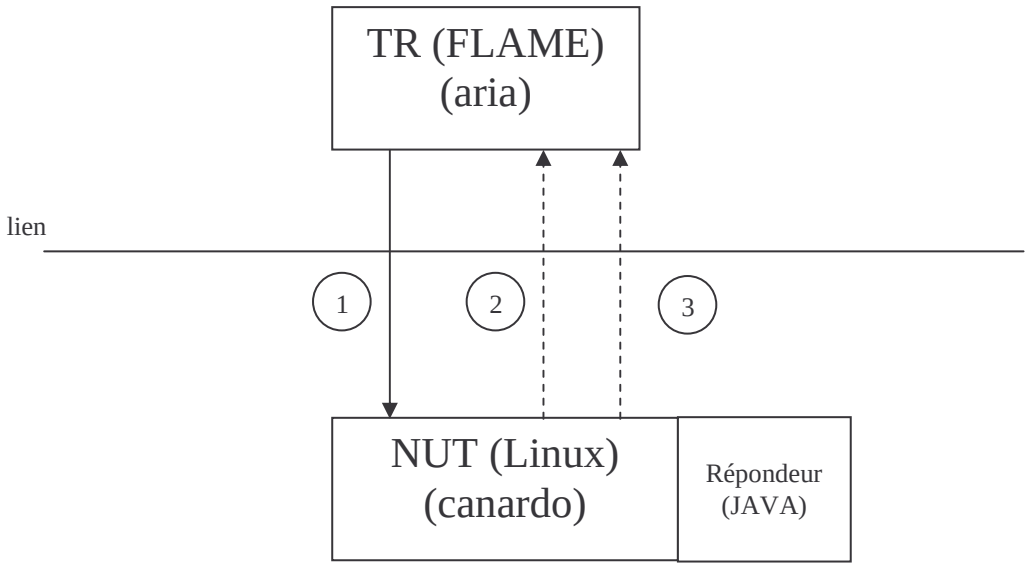


Fig 35 – *Architecture Test Générique*

Cette architecture nous permet de migrer vers une approche plus générique et plus portable. Elle nous offre aussi la possibilité de tester l'interopérabilité Linux-FreeBSD.

5.4 Implémentation de l'API :

API : Application Programing Interface : ensemble d'outils et de leurs règles d'utilisation qu'un logiciel met à la disposition des programmeurs pour réaliser leurs applications.
L'API regroupe les routines utilisées par un programme pour demander et exploiter les services d'un autre programme ou d'une bibliothèque de fonctions.

API MLD:

L'API MLD regroupe les deux fonctions suivantes :

- Envoyer à une machine destinataire un message, le message peut être de type Query, Report ou Done :

  *int (*send_Message) (struct flame *api, int type, int code, struct sockaddr_in6 *src, struct sockaddr_in6 *dst, struct in6_addr *group, int index, int delay, int datalen, int alert)*

- Initialiser la Raw socket qui va émettre et recevoir les messages MLD :

  *int (*init) (struct flame *api, int filtre)*

Cette fonction définit le filtre de la Raw socket pour ne recevoir que les messages dont on a besoin, les filtres possibles sont : PASS_ALL, BLOCK_ALL, PASS_QUERY, PASS_REPORT, PASS_DONE, PASS_QUERY_REPORT, PASS_QUERY_DONE, PASS_REPORT_DONE. Cette fonction retourne le descripteur de la Raw socket.

Le chargement de l'API MLD à partir de FLAME se fait par la commande :

FLAME_GETAPI(mld,1,0) ;

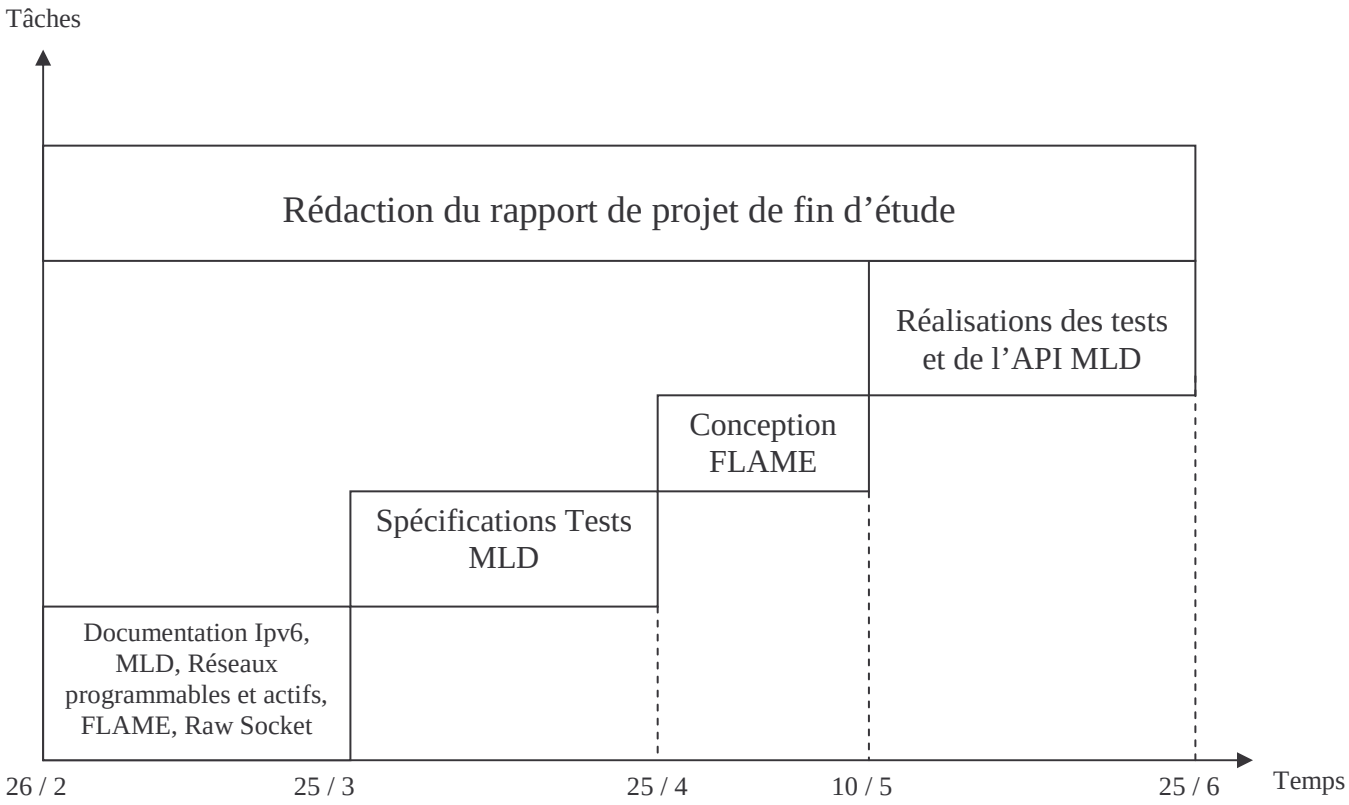
L'utilisation de l'API MLD à partir de FLAME se fait par la commande :

APICALL (mld, Send_Message, …) ;
APICALL (mld, Init, …) ;

5.5 Ce qui reste à faire :

Pour pouvoir assurer une bonne couverture de tests, il faut réaliser le plus grand nombre de tests ; ainsi, ce qui reste à faire dans ce projet est d'implémenter tous les tests spécifiés en utilisant l'environnement d'exécution actif FLAME. L'API MLD contribuera au mieux à l'implémentation de ces tests.
Une fois que tous les tests auront été réalisés, il faudrait calculer la couverture de ces tests et déduire ainsi que le protocole testé est conforme à sa spécification ou non.

5.6 Chronogramme :



5.7 Conclusion :

La réalisation des tests MLD a été effectuée en utilisant l'environnement d'exécution actif FLAME, développé au sein de l'équipe de recherche RESEDAS à LORIA. L'architecture de FLAME et son mode de fonctionnement ont contribué à la bonne mise en place de ces tests.
Les résultats des tests étaient conformes à la spécification formelle de MLD v1, mais ceci ne montre pas encore que le protocole testé est conforme. Il faudrait à ce stade réaliser une suite de tests assurant une bonne couverture des états et des transitions du protocole et pouvoir ainsi déduire la conformité de MLD v1 par rapport à sa spécification.

Chapitre 6

Conclusion Générale

Les protocoles IPv6, comme tout logiciel, passent par l'étape de tests qui vérifient leur conformité par rapport à leurs spécifications formelles. Ces tests sont un moyen sur et efficace pour aboutir à un protocole conforme.
Dans ce cadre a été mené ce projet intitulé « Utilisation de la technologie active pour tester les Protocoles Multicast Ipv6 ».

Le protocole choisi pour les tests est le MLD : Multicast Listeners Discovery, c'est un protocole de gestion de groupes multicast, de niveau lien local qui permet à un routeur de détecter la présence de nœuds sur l'ensemble de ses liens désirant recevoir des paquets relatifs à un groupe multicast. Le protocole MLD permet ainsi de tenir à jour, continuellement, la liste des adresses multicast disposant de listeners. Cette information est utilisée par les protocoles de routage multicast pour pouvoir acheminer les paquets vers les nœuds intéressés.

La génération des suites de tests a été modélisée par OSI qui a distingué plusieurs types de tests de conformité : test local, test distribué, test coordonné et test à distance. La structure d'une suite de tests selon OSI comprend un groupe de tests de capacité, un groupe de test de comportement valide, un groupe de test de comportement invalide et un groupe de test d'interconnexion.
On pourrait aussi générer des suites de tests en se basant sur la modélisation du protocole en un automate à états finis. Les tests auront ainsi pour but de détecter la présence ou l'absence des différents types de fautes : faute d'output, faute de transfert, faute de transfert avec états supplémentaires et faute de perte ou d'ajout de transitions.

Pour réaliser les tests MLD, deux approches étaient possibles : approche non active et approche active. L'approche non active consiste à mener les tests dans un réseau traditionnel tandis que l'approche active profite de tous les avantages des réseaux actifs et contribue au mieux déroulement des tests. En effet, l'approche active offre un déploiement dynamique du code, une capacité à télécharger les scénarios de tests à n'importe quel endroit du réseau, et ainsi une souplesse et facilité de réalisation des tests.

C'est ainsi qu'on a choisi de réaliser les tests sous une infrastructure active offrant au testeur souplesse et facilité d'utilisation des tests à travers un hôte de contrôle. L'environnement d'exécution actif qui a été utilisé est le FLAME développé par l'équipe de recherche RESEDAS à LORIA.

Les résultats obtenus des tests sont conformes aux spécifications, mais cela ne suffit pas pour s'assurer que le protocole MLD v1 est conforme à ses spécifications, et donc il faudrait réaliser le plus grand nombre de tests possibles et avoir ainsi une bonne couverture de tests.

D'un point de vue plus personnel, ce projet de fin d'étude a été enrichissant puisqu'il m'a permis d'aborder simultanément trois sujets : le protocole IPv6, les tests de protocoles et les réseaux actifs. En plus, j'ai pu me rendre compte du savoir faire et de l'état d'esprit d'une équipe de recherche dont la vocation est de mener des projets de recherche de grande qualité.

Comme perspectives de ce travail, on pourrait envisager de réaliser des tests sur la deuxième version de MLD en se basant sur une approche active, et ainsi pouvoir vérifier les modifications apportées à la première version. On pourrait envisager aussi de tester les protocoles de routage multicast comme le PIM en se basant également sur une approche active.

Glossaire

Adresse Anycast : désigne un groupe d'interfaces, lorsqu'un paquet a pour destination une telle adresse, il est acheminé à un des éléments du groupe et non pas à tous.

Adresse Multicast : désigne un groupe d'interfaces qui en général appartiennent à des nœuds différents pouvant être situés n'importe où dans l'Internet. Lorsqu'un paquet a pour destination une adresse de type multicast, il est acheminé par le réseau à toutes les interfaces membres de ce groupe.

Adresse Unicast : désigne une unique interface. Un paquet envoyé à une telle adresse sera donc remis à l'interface ainsi identifiée.

API : Application Programming Interface : ensemble d'outils et de leurs règles d'utilisation qu'un logiciel met à la disposition des programmeurs pour réaliser leurs applications.

Approche active : approche basée sur les réseaux actifs.

Done : message MLD de valeur 132, un nœud envoie le message Done au routeur dès qu'il cesse d'écouter sur une adresse multicast.

FLAME : Environnement d'exécution actif développé par RESEDAS.

IPv6 : Nouvelle version d'IPv4, l'avènement d'IPv6 a remédié aux différents inconvénients d'IPv4 mais aussi a apporté beaucoup de nouvelles fonctionnalités.

MLD : Multicast Listener Discovery : protocole permettant à un routeur de maintenir tous les listeners d'une adresse multicast. Cette information sera utile pour le protocole de routage.

NUT : Nœud sous test.

Query : message MLD de valeur 130, un routeur Querier envoie aux nœuds du même lien des Query à un intervalle de temps de Interval Querier afin de se rendre compte des nouveaux listeners.

Raw Socket : type de socket qui permet de lire et écrire des paquets ICMPv4, IGMPv4 et ICMPv6.

Report : message MLD de valeur 131, quand un nœud se met à l'écoute d'une adresse multicast, il envoie un Report au routeur pour lui informer.

Réseau actif : réseau dans lequel tout ou une partie de ses composants dans les différents plans sont programmables dynamiquement par des entités tierces.

Réseau programmable : réseau de transmission de données ouvert et extensible disposant d'une infrastructure dédiée à l'intégration et à la mise en œuvre rapide de nouveaux services sur l'ensemble de ses composants.

RFC : Request for comments.

RUT : Routeur sous test.

Test : Phase de développement d'un logiciel, elle permet de vérifier la conformité de ce logiciel par rapport à sa spécification.

TN : Nœud testeur.

TR : Routeur testeur.

Index

Bibliographie

[DAV 98] David Mercier, « Plan de test Internet Nouvelle Génération », Mémoire de DEA Informatique Août 1998, 78 pages

[DRAFT] Jun-ichiro Itojun Hagino, IIJ Research Laboratory, « Socket API for IPv6 Traffic Class Field », Internet Draft Septembre 2000, 5 pages

[FRA 01] François Bourdoncle, « Cours Systèmes et Réseaux », Septembre 2001, 15 pages

[GIS 97] Gisèle Cizault, « IPv6 », Editions Truc Décembre 1997, 295 pages

[ISO-9646] ISO, Information processing System, Open System Interconnexion, OSI Conformance Testing and Framework ISO-9646, 1991

[LOU 00] Mohamed Louzif, « Tests des protocoles », Septembre 2000, 34 pages

[OLI 00] Olivier Festor, Isabelle Chrisment, Eric Fleury, « Les réseaux programmables 1.0 », INRIA Mars 2000, 73 pages

[OLI 01] Olivier Festor, Abdelkader Lahmadi, Stephan Dalu, « FLAME », LORIA INRIA Avril 2001, 41 pages

[RFC 1883] S. Deering, R. Hinden, « Internet Protocol, Version 6 Specification », Décembre 1995

[RFC 1884] S. Deering, R. Hinden, « IP Version 6 Addressing Architecture », Décembre 1995

[RFC 1885] A. Conta, S. Deering, « Internet Control Mesage Protocol ICMPv6 for the Internet Protocol Version 6 », Décembre 1995

[RFC 2710] S. Deering, Cisco Systems, W. Fenner, AT&T Research, B. Haberman, IBM, « Multicast Listener Discovery for IPv6 », Octobre 1999, 18 pages

[RIC 98] W. Richard, « Unix Network Programing, networking APIs, sockets and XTI », Prentice-Hall inc. 1998

[SITE 1] http://www.ensem.fr

[SITE 2] http://www.kame.net

[SITE 3] http://www.freebsd.org

[SITE 4] http://www.java.sun.com

[SITE 5] http://www.tahi.org

[VOL 01] Volodymyr Nemchenko, « Génération de suites de test pour les protocoles IPv6 », INRIA Décembre 2001, 14 pages

Annexe

Annexe A : FLAME :

FLAME Packet :
La structure d'un paquet FLAME est comme suit :

```
Struct fpacket{
        int             tid ;       /* transport id*/
        char            fpid ;      /* packet id*/
        char            data ;      /* data buffer*/
        int             datalen ;   /* data length*/
        struct in6_addr *addr ;     /* src/dst addresse*/
        struct in6_addr *faddr ;    /* final address*/
        int             mif ;       /* multicast interface*/
        int             flags ;     /* packet options */
} ;
```

- tid : transport identifier, sa valeur est donnée par la fonction get_transport_id,
- fpid : FLAME packet identifier,
- datalen : longueur du paquet actif,
- data : paquet actif,
- addrlen : longueur de l'adresse de l'émetteur du paquet,
- addr : adresse de l'émetteur du paquet,
- faddr : adresse destination du paquet,
- mif : utilisé seulement en cas d'émission de paquet, il permet de choisir en cas de multicast l'interface sur laquelle le paquet devrait s'envoyer,
- flags : spécifie le comportement d'un paquet pour un routeur.

Mise en place de l'environnement d'exécution FLAME :

La structure du code FLAME est comme suit :

```
FLAME
|--------doc
|--------patch
|--------src
|        |--------aadir
|        |--------api
|        |--------transp
|        |--------flame
|        |        |--------aa
|        |        |--------ee
|        |        |--------flame
|        |--------misc
```

```
|       |--------easyc
|--------include
|--------lib
|--------test
```

L'installation de FLAME se fait comme suit :
./configure
gmake
#gmake install

La structure de FLAME install est la suivante :

```
PREFIX
|--------bin
|--------libexec
|       |--------flame
|       |       |--------lib
|       |       |--------api
|       |       |--------transp
|       |       |--------aadir
|       |--------httpdir
|--------include
```

Exemple d'exécution d'une application active :

On lance FLAME par la commande :
./flame –a –h : :
On ouvre une session telnet sur la machine où s'exécute le flame, exemple :
telnet treize 6666
Connected to treize
cmd> uagent {a=ping, v=1, p=http://127.0.0.1 :6667}
[Failed] : AA is loading

Le premier paquet transmis est toujours perdu du fait que l'application active n'est pas encore chargée, dans notre exemple c'est le ping, et dès qu'une requête est lancée sur une application active donnée, le code est directement chargé dans l'environnement d'exécution.
Ainsi, si on répète la même commande, le paquet transmis sera traité par l'application active ping :
cmd> uagent {a=ping, v=1, p=http://127.0.0.1 :6667}
Hostname : treize
Ping request for :
- host : treize
- IP : 2001:660:301:34:201:2ff:fee3:6013
- FPID : 20010660030100340201 02fffee3601368f7312cd4686ff281c56fa714da9205
Ping RTT = 0ms
Connection Closed by the foreign host.

La solution à la perte du premier paquet actif pourrait par exemple être l'utilisation d'un buffer ou on stocke tous las paquets arrivés. Un paquet ne sera détruit que s'il a été traité par l'application active correspondante.

Ecrire une application active :

L'application active a la forme suivante :

```
#include<flame/flame.h>
DEFINE_VESION (1,0,0) ;
int init (void) {
        …
        return 1 ;
}

int pkthdlr (fpacket_t *fp) {
        …
        return 1 ;
}

int uagent (int fd, int argc, char **argc) {
        …
        UA_RETURN_OK() ;
}
int aamain (void) {
        while (1) {
                …
        }
}
```

1/ Initialisation :

int init (void) : cette fonction est appelée pour réaliser l'initialisation d'une application active : elle initialise les variables globales, les mutex et déclarent les API qui vont être utilisées par l'application active.

2/ Paket handler :

int pkthdlr (fpacket_t *fp) : cette fonction permet la lecture des paquets actifs reçus par l'application active.

3/ User Agent :

int uagent (int fd, int arg c, char ** argv) : cette fonction doit être généralement implémentée parce qu'elle offre à l'utilisateur un moyen pour configurer et commencer une application active.

4/ Aamain :

int aamain (void) : cette fonction permet d'avoir un thread permanent au sein de l'application active (c'est un thread démon). Si cette fonction est implémentée, l'application active ne sera pas détruite même si elle ne reçoit pas des paquets actifs.

Annexe B : RAW Socket :

Création d'une Raw socket :

Quatre étapes pour la création d'un Raw socket :

- La fonction socket crée une Raw socket quand le deuxième argument est SOCK_RAW. Le troisième argument est normalement différent de zéro. Par exemple, pour créer une Raw socket IPv4 on doit écrire :

  Int sock ;
  Sock = socket (AF_INET, SOCK_RAW, protocol) ;

  Seulement le super utilisateur (root) peut créer des Raw sockets. Ceci empêche les utilisateurs d'écrire leurs propres datagrammes IP sur le réseau.

- L'option IP_HDRINCL peut être mise comme suit :

  Const int on = 1 ;
  If (setsockopt (sock, IPPROTO_IP, IP_HDRINCL, &on, sizeof (on)) < 0)
  Error

- bind peut être utilisé pour les Raw sockets mais c'est très rare. Le concept de nombre de port pour les Raw sockets n'existe pas.

- connect peut être utilisé pour les Raw sockets mais c'est très rare. Cette fonction spécifie l'adresse destination mais une autre fois le concept de nombre de port pour les Raw sockets n'existe pas.

Raw socket Output :

L'output de Raw socket est basé sur un ensemble de règles :
- L'output normal est appelé par *sendto* ou *sendmsg* et spécifie l'adresse IP destination. *write, writev, send peuvent aussi être appelées lorsque la socket est connectée.*
- Le noyau fragmente les Raw packets dont la longueur est supérieure à la MTU.

Raw socket Input :

- Des paquets UDP ou TCP reçus ne sont jamais passés à la Raw socket.
- La plupart des paquets ICMP sont transmis à la Raw socket.
- Tous les paquets IGMP sont transmis à la Raw socket.
- Tous les datagrammes IP que le noyau ne comprend pas la nature du champ protocole sont transmis à la Raw socket.
- Si un datagramme arrive en fragments, rien n'est transmis à la Raw socket jusqu'à ce que tous les fragments soient rassemblées.
- Pour réduire le nombre de paquets transmis à une application donnée utilisant les Raw sockets, une application filtre est mise au point, elle permet à chaque application de spécifier la nature des paquets qu'elle veut recevoir.

Exemple :

Considérons une application qui veut recevoir seulement les ICMP router advertissements :

```
struct imcp_filter myfilt ;
fd = socket ( AF_INET6, SOCK_RAW, IPPROTO_ICMPv6) ;
ICMP_FILTER_SETBLOCKALL ( &myfilt ) ;
ICMP_FILTER_SETPASS ( ND_ROUTER_ADVERT, &myfilt ) ;
Setsockopt ( fd, IPPROTO_ICMPv6, ICMP_FILTER, &myfilt, sizeof (myfilt )) ;
```