MASTER EN SCIENCES INFORMATIQUES

Développement d'un simulateur de OSPF

Gaussin, Jean

Award date:
2000

FUNDP
Institut d'Informatique

Rue Grandgagnage,21
5000 - NAMUR
BELGIQUE

DEVELOPPEMENT D'UN SIMULATEUR DE OSPF

Mémoire présenté pour l'obtention
du grade de Licencié en Informatique
par

Promoteur : Olivier BONAVENTURE

Jean GAUSSIN

Année Académique 1999-2000

Table des matières

Table des figures

Introduction

L'Internet est un réseau global de communication qui permet de relier plusieurs centaines de millions de personnes dans une grande communauté. Celui-ci connaît aujourd'hui une croissance très importante qui laisse penser qu'il deviendra LE média de communication par excellence des décennies à venir. Beaucoup de personnes connaissent l'Internet et savent comment l'utiliser mais peu ont conscience de la technologie sous-jacente.

On peut simplifier l'Internet en le représentant comme un ensemble de machines spéciales, les routeurs, dont le but est d'interconnecter les différents réseaux qui ensemble forment l'Internet. L'information circule, comme le courrier postal, à travers le réseau dans de petites unités de données (les paquets) depuis un émetteur jusqu'à un destinataire. Le problème à résoudre est alors le suivant : comment envoyer une information d'un point A à un point B ? Ce problème est semblable à celui de la gestion du trafic routier : si un automobiliste veut se rendre d'un point A à un point B, il utilise une carte routière pour s'orienter. L'Internet fonctionne de manière semblable mais avec une différence notable : alors que l'automobiliste est responsable à chaque carrefour du chemin qu'il doit prendre pour rejoindre sa destination, l'information qui circule dans l'Internet ne possède aucune forme d'intelligence. C'est donc au routeur qu'incombe, à chaque *carrefour*, le choix du meilleur chemin en se basant sur la destination de cette information. Ce choix ne peut en aucun cas être aléatoire, il doit permettre à l'information de rejoindre le plus rapidement possible sa destination. Il est donc nécessaire qu'un routeur possède une connaissance plus ou moins importante de la topologie de l'Internet pour pouvoir orienter efficacement l'information qu'il reçoit. L'apprentissage et l'entretien de la *carte* de l'Internet sont réalisés par des protocoles particuliers qu'on appelle communément : **Protocole de routage.**

Plusieurs types de protocole de routage existent. On peut les classer en deux grandes catégories : d'une part les protocoles statiques et d'autre part les protocoles dynamiques.

Les protocoles statiques exigent une définition et une adaptation manuelle au niveau de chaque routeur de la topologie de l'Internet et des chemins disponibles. Cette approche, très défensive, va à l'encontre du principe même du Réseau qui ce veut quelque chose de souple et d'évolutif. Les protocoles dynamiques, plus complexes, répondent mieux à cette attente. Ils permettent à un routeur d'apprendre rapidement et automatiquement toutes modifications topologiques. Pour ce faire, tous les routeurs collaborent entre eux de manière à ce que chacun obtiennent finalement une image correcte du réseau. Ainsi, si tous les routeurs possèdent la même carte et implémentent le même processus de décision pour orienter l'information, on peut affirmer que, si un routeur quelconque *planifie* un chemin pour une information à travers l'Internet, alors, son suivant sur ce chemin planifiera le même chemin pour cette information. Beaucoup de protocoles de routage reposent sur l'approche dynamique ; OSPF (Open Shortest Path First) est l'un d'eux.

Connaître la carte du réseau est quelque chose d'absolument nécessaire pour véhiculer efficacement des paquets de données, mais d'autres paramètres entrent en jeu. En faisant abstraction des exigences économiques ou politiques, le plus important est sans aucun doute l'encombrement (nous utiliserons ici le terme de charge) du réseau. Un simple constat : il est parfois plus intéressant de prendre un *itinéraire bis* un peu plus long mais fluide plutôt qu'une autoroute plus rapide mais complètement bouchée. OMP (Optimized Multi Path) est un protocole complémentaire à OSPF dont le but principal est d'améliorer ce dernier en prenant en compte la quantité et la localisation du trafic dans le réseau. Comme OSPF, OMP est un protocole collaborant qui vise en premier lieu l'utilisation optimale de toutes les ressources disponibles (ce n'est pas parce qu'une voie secondaire est peu utilisée que tout le monde doit quitter l'autoroute pour l'emprunter).

Ce mémoire a pour but la réalisation d'un simulateur du comportement de OSPF en JAVA. Il comprend également une partie consacrée à OMP. Plusieurs objectifs ont été poursuivis durant son élaboration :

- premièrement, fournir un outil didactique qui puisse aider à la compréhension des mécanismes de base du protocole OSPF ;

- deuxièmement, montrer que la conception de ce simulateur permet l'ajout de protocoles complémentaires comme OMP sans remettre en cause les fondements de sa conception ;
- troisièmement, fournir un outil pour comprendre le fonctionnement de OMP et partant de là, l'utiliser comme banc de test.

Ce travail se divise en 4 chapitres et une annexe.

Le premier chapitre présente une synthèse théorique des différents mécanismes utilisés dans le protocole OSPFv2. Celui-ci a été décrit par John T. MOY dans le RFC 2328. Ce protocole est largement utilisé aujourd'hui dans l'Internet.

Le second chapitre décrit les mécanismes complémentaires propres à OMP. Ce protocole n'est pas à proprement parlé un standard de la communauté Internet ; il a été proposé par Curtis VILLAMIZAR dans « draft-ietf-ospf-omp-02 ». Des applications propriétaires y font aujourd'hui référence.

Le troisième chapitre présente l'architecture du simulateur OSPF en 2 étapes : premièrement, les spécifications du simulateur sont données ; deuxièmement, les grands concepts dégagés lors de la première étape et utilisés pour l'implémentation sont présentés de manière thématique.

Le quatrième et dernier chapitre utilise une démarche similaire à celle du chapitre 3. Il présente l'architecture de OMP et son intégration dans le simulateur OSPF.

L'annexe 1 est un manuel expliquant le fonctionnement du simulateur.

Avec ce travail est fourni sur support électronique, outre une version exécutable du simulateur, le code source commenté de toutes les classes JAVA.

L'élaboration de ce travail n'aurait pas été possible sans l'aide ni les conseils avisés de Monsieur le Professeur O. BONAVENTURE, promoteur de ce mémoire et de son assistant Monsieur A. KASSA MOLALA. Je tiens à les en remercier vivement. Ma gratitude va également à Monsieur le Colonel DEGHEZELLE, i.r. et Monsieur le Colonel PINCHEMAIL, i.r. qui m'ont permis, durant ces deux dernières années, de suivre cette formation. Je voudrais aussi remercier Monsieur C. DETHIER qui a relu ce travail pour en relever les imperfections. Finalement, je ne peux oublier toute l'aide que SANDRA, mon épouse, a pu m'apporter.

Chapitre 1 : Description du protocole OSPF

1. Introduction

a. *Les protocoles de routage*

Dans l'Internet, les machines communiquent entre elles en s'envoyant des messages. Ces messages contiennent entre autres l'adresse de l'émetteur et l'adresse du destinataire. Sur base de ces informations, les paquets[1] seront acheminés à travers tout le réseau. Deux méthodes peuvent être utilisées pour véhiculer l'information :

1) par circuit virtuel. Dans ce cas, on crée au travers du réseau un *chemin* entre l'émetteur et le destinataire et tous les paquets entre les deux stations ne passent que par ce chemin. Ce mécanisme n'est utilisé dans l'Internet que pour certains cas particuliers ; il ne sera pas traité ici.
2) par datagramme. Dans ce cas, aucun chemin n'est créé. Mais, au niveau de chaque routeur, pour chaque paquet entrant, une décision est prise en fonction de l'adresse de destination pour choisir la ligne de sortie vers laquelle le paquet doit être envoyé. Deux paquets concernant une même liaison entre deux machines peuvent donc cheminer de manière différente dans l'Internet en fonction des décisions prises par les routeurs. La plupart des protocoles de routage dans l'Internet sont basés sur cette approche.

Au niveau du routage par datagramme, deux options peuvent être choisies :

1) routage statique. Dans ce cas, l'information concernant le routage est définie manuellement au niveau du routeur. Cette méthode est facile à implémenter mais présente l'inconvénient de n'offrir aucune possibilité de réaction rapide et automatique en cas de modifications topologiques. Cette méthode n'est presque jamais utilisée sauf dans certains réseaux TCP/IP très stables.
2) routage dynamique. Dans ce cas, l'information concernant le routage est apprise de manière automatique en fonction de la topologie du réseau et de son évolution. Cette méthode permet une réaction rapide face à tout changement. Cette méthode, beaucoup plus souple, est utilisée par tous les protocoles de routage de TCP/IP.

Deux types de techniques sont utilisés par un routeur *dynamique* pour apprendre la topologie du réseau et partant de là, les différents chemins existants. Le principe général est la recherche du meilleur chemin (en terme de coût) pour joindre une destination.

1) vecteur de distance. Chaque routeur envoie périodiquement à ses voisins la liste des chemins qu'il utilise. Le routeur voisin qui la reçoit la compare à sa propre liste. Pour chaque destination pour laquelle le coût local est supérieur à celui reçu, l'entrée correspondante vers cette destination dans la table de routage locale est modifiée de façon à référencer le routeur qui propose un chemin plus court vers cette destination[2]. Cette méthode permet ainsi de converger vers un ensemble de chemins optimaux, mais elle présente plusieurs inconvénients notamment en cas d'instabilité du réseau. RIP (Routing Information Protocol) est un exemple de protocole de routage dynamique qui utilise cette technologie.

---

[1] Le paquet est l'unité de transmission au niveau du protocole IP.

[2] Supposons le routeur A qui, pour atteindre le routeur B, envoie ses paquets vers le routeur C et que le coût du chemin A-B est de 10. Supposons un routeur D, voisin du routeur A avec, entre les deux, une liaison de coût 2, et que ce routeur D annonce un coût de 5 pour le chemin D-B. Lorsque A reçoit cette information, il se rend compte que le chemin proposé par D augmenté du coût de la liaison A-D est moins coûteux (sa valeur est 7) que le chemin qu'il connaît actuellement pour joindre B(la valeur est 10). A va donc remplacer dans sa table de routage le routeur C par le routeur D pour joindre le routeur B et donner à ce chemin un coût de 7.

2) état de liaison. Chaque routeur possède une image de la topologie du réseau. Celle-ci est obtenue par l'envoi régulier par chaque routeur de sa structure locale (c'est-à-dire les liaisons qui vont et partent de lui) à tous les membres du réseau. Chaque routeur, ayant une image complète et identique du réseau et du coût de chacune des liaisons, peut dès lors calculer le chemin le plus court vers chacune des destinations. Des protocoles tels que OSPF et IS-IS (Integrated Intermediate System-to-Intermediate System) utilisent ce mécanisme.

Finalement, l'Internet a été découpé en différentes zones appelées *systèmes autonomes* pour des raisons de taille, de politique et d'économie. Les protocoles de routage qui gèrent le cheminement des paquets au sein d'un système autonome sont appelés IGP (Internal Gateway Protocol). RIP, IS-IS et OSPF font partie de ceux-là. Les protocoles qui gèrent les relations entre les systèmes autonomes s'appellent EGP (External Gateway Protocol).

En résumé, on peut situer OSPF dans la famille TCP/IP comme un protocole de routage dynamique IGP à état de liaison [MOY98a p. 6-7].

b. Les protocoles à état de liaison

Les protocoles à état de liaison sont fondés sur la distribution d'un graphe orienté représentant la structure du réseau. Les nœuds représentent des routeurs ou des sous-réseaux. Deux arcs en sens opposés entre deux routeurs représentent une liaison point-à-point. Deux arcs en sens opposés entre un routeur et un sous-réseau représentent l'interface de ce routeur avec le sous-réseau. Tous les nœuds d'un même réseau possèdent une copie complète à jour de la structure de ce réseau. Elle est contenue dans une base de données où chaque enregistrement représente un arc. En simplifiant quelque peu, on peut considérer qu'un enregistrement est composé de quatre champs :

1) émetteur (un nœud du réseau) ;
2) destination (un autre nœud du réseau qui est relié à cette origine) ;
3) identifiant d'interface valide pour le nœud émetteur (il faut pouvoir distinguer des arcs ayant la même origine et la même destination) ;
4) poids affecté à cette liaison, les valeurs les plus petites étant les meilleures.

L'origine, la destination et l'identifiant d'interface permettent de reconnaître de manière univoque une liaison orientée dans le réseau.

Possédant ces informations, chaque nœud peut facilement calculer le chemin le plus court entre lui-même et tous les autres nœuds de son réseau. Les protocoles à état de liaison utilisent habituellement pour ce calcul l'algorithme de Dijkstra : "Shortest Path First".[3] Etant donné que chaque nœud possède la même base de données, aucune boucle ne sera possible dans le routage.

Sur base de sa connaissance des meilleurs chemins pour rejoindre toute destination dans le réseau, le routeur peut dès lors construire sa table de routage en indiquant, pour chaque destination non directement joignable, l'adresse du prochain saut (qui correspondra à une de ses interfaces de sortie).

2. Le cœur de OSPF

a. Les états de liaison

Chaque routeur est responsable de la description de sa partie locale du domaine[4] via des messages d'état de liaison (LSA – Link State Advertisement). Ces messages sont régulièrement envoyés vers les autres

[3] Il n'y a pas d'obligation quant à l'usage d'un algorithme particulier. Cependant, l'algorithme de Dijkstra est le plus efficace avec une complexité de l'ordre de O (m * log m) où m représente le nombre de liaisons dans le réseau. Une description plus complète de l'algorithme de Dijkstra ainsi qu'un exemple sont donnés dans le chapitre traitant de l'architecture du simulateur.

[4] Par domaine, on entend l'ensemble des routeurs d'un système autonome qui partage le même protocole OSPF.

routeurs à l'aide d'un protocole particulier. Ensemble, tous les états de liaison forment la base de données d'état de liaison qui décrit le réseau et permet ainsi de calculer les chemins. Il faut noter qu'un routeur ne peut effacer ou modifier (sauf dans certains cas particuliers) que ses propres LSA.

OSPF distingue à l'origine cinq types de messages d'état de liaison [HUI94 p. 148] en fonction de l'information qui est communiquée. Au fur et à mesure que le protocole sera expliqué, les LSA correspondants seront décrits.

Tout LSA, quel que soit son type, possède un en-tête commun [MOY98b p.74-79] de 20 octets résumé dans la figure 1[5].

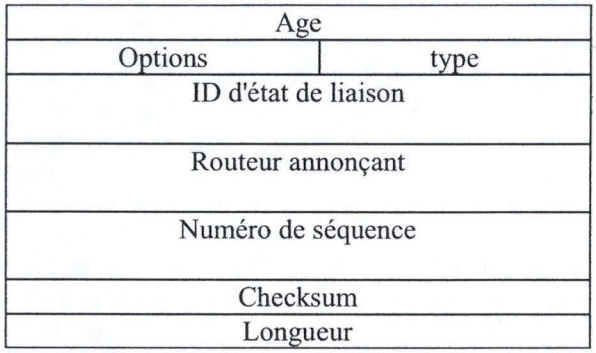


Figure 1 : En-tête d'un LSA

Chacun des champs de l'en-tête est responsable d'une fonction particulière. Dans ce qui suit, ceux-ci vont être détaillés de manière thématique.

1) Identifier un LSA

a) Le champ *LS Type*

Le champ *LS Type* indique le type du message LS. On distingue cinq types :

1. routeur ;
2. réseau (network – LSA) ;
3. récapitulatif réseau IP (network - summary – LSA) ;
4. récapitulatif routeur externe (ASBR – summary – LSA) ;
5. externe (AS-external LSA).

D'autres types ont été développés pour des extensions de OSPF et ne seront pas traités ici.

b) Le champ *ID d'état de liaison*

Le champ *ID d'état de liaison* [MOY98a p. 126-133] doit permettre la distinction d'un LSA parmi les autres LSA de même type émis par un même routeur. Dans la plupart des cas, l'information de distinction est une adresse IP.

[5] Dans cette figure et dans toutes celles qui représentent une structure de données, une ligne d'épaisseur simple représente 16 bits (exemple : champ *Age*). De la même manière, une ligne d'épaisseur double représente 32 bits (exemple : champ *Numéro de séquence*) et ainsi de suite. Une ligne séparée en deux indiquent que chacun des champs qui la compose à une taille de 8 bits (exemple : champ *type*).

c) Le champ *routeur annonçant*

Le champ *routeur annonçant* identifie le routeur qui crée le LSA. Sa valeur est une des adresses IP de ce routeur; elle est dès lors considérée comme l'identifiant de ce routeur dans le domaine OSPF.

La combinaison des trois champs précités permet d'identifier de manière unique un LSA parmi tous les autres [MOY98b p.74].

2) Identification les différentes instances d'un même LSA

a) Le champ *numéro de séquence*

OSPF distingue les différentes instances d'un même LSA à l'aide d'une séquence linéaire codée sur 32 bits signés. Cette valeur est placée dans le champ *numéro de séquence*. L'instance la plus récente est celle possédant la valeur la plus élevée pour ce champ.
La première instance créée reçoit le plus petit nombre négatif (InitialSequenceNumber) et chaque nouvelle instance voit la valeur de ce champ augmentée d'une unité par rapport à celle de l'instance précédente.
Lorsqu'on arrive au maximum (MaxSequenceNumber), on recommence avec la valeur minimale. Cependant, pour que cette nouvelle instance soit reconnue comme étant la plus récente, le routeur émetteur doit d'abord forcer l'effacement de l'instance précédente dans tous les routeurs du domaine avant de diffuser le nouveau LSA.
Il est à noter que, dans des circonstances normales, l'intervalle de temps entre la diffusion de deux instances doit être au minimum de cinq secondes... ce qui laisse une période de rotation de l'ordre de 600 ans.

3) Vérification du contenu du LSA

a) Le champ *checksum*

L'en-tête (à l'exception du champ *Age*) et le contenu d'un LSA sont sécurisés contre les erreurs de transmission à l'aide de l'algorithme de checksum de Fletcher. Sa valeur est déterminée lors de la création du LSA, est incorporée dans son en-tête et n'est jamais modifiée[6]. Le checksum est vérifié dans chaque routeur par lequel le LSA transite et une corruption entraîne son rejet. Au sein même d'un routeur, les checksum des différents LSA stockés sont vérifiés régulièrement et toute erreur découverte entraîne une réinitialisation du routeur.

4) Suppression des vieux LSA

a) Le champ *Age*

Le champ *Age* indique le temps qui s'est écoulé depuis la génération d'une instance particulière d'un LSA. Sa valeur est initialisée à zéro par le routeur émetteur. Elle est ensuite incrémentée dans chaque relais lors de sa diffusion et toutes les secondes dans les bases de données des routeurs. Normalement, quand sa valeur atteint 30 minutes (LSRefreshTime), une nouvelle instance de ce LSA est générée par son émetteur. Lorsqu'un routeur quelconque reçoit cette nouvelle instance, il la met à la place de l'ancienne.

Si, pour une raison quelconque, le LSA n'est pas rafraîchi, il est conservé dans la base de données des routeurs jusqu'au moment où son âge atteint 60 minutes (MaxAge). Passé ce délai, le LSA est considéré comme périmé et il est supprimé d'autorité de la base de données. Avant la suppression cependant, pour garantir une synchronisation de toutes les bases de données du domaine, le LSA

[6] C'est la raison pour laquelle le champ donnant l'âge du LSA n'est pas inclus dans le calcul du checksum ; celui-ci est sujet à modification et de plus, une erreur dans ce champ n'a pas de conséquences dramatiques.

avec la valeur MaxAge est diffusé[7] vers les autres routeurs (un routeur recevant un LSA avec une valeur MaxAge pour le champ *Age* interprétera cette information comme une demande de suppression de sa version locale de ce LSA car il est considéré comme obsolète dans le domaine). Cette propriété est aussi utilisée par un routeur émetteur pour forcer l'effacement d'un de ses LSA dans le domaine.

L'âge d'une instance d'un LSA permet d'implémenter d'autres fonctionnalités [MOY98b p. 80] :

- MinLsArrival : temps minimal entre deux mises à jour dans la base de données (une seconde) ;
- MinLsInterval : temps minimal entre deux mises à jour d'un même LSA (5 secondes) ;
- CheckAge : intervalle de temps entre deux vérifications du checksum d'un LSA (5 minutes) ;
- MaxAgeDiff : si deux instances d'un même LSA diffèrent de plus de MaxAgeDiff, elles sont considérées comme deux instances séparées et la plus jeune est traitée comme étant la plus récente.

5) Autres champs

a) Le champ *option*

Le champ *option* [MOY98a p. 187-188] décrit les capacités du routeur émetteur au regard de certaines fonctionnalités de OPSF.

b) Le champ *longueur*

Le champ *longueur* donne la taille en octet du LSA complet (en-tête et contenu). Des valeurs de l'ordre de quelques centaines d'octets ne sont pas rares.

b. La base de données d'état

La base de données d'état [MOY98b p.83-84] donne une description complète du domaine avec ces routeurs, ces réseaux et la manière dont ceux-ci sont interconnectés. Cette base de données doit être identique pour tous les routeurs du domaine OSPF. Deux bases de données sont synchronisées si elles possèdent le même nombre d'enregistrements et que la somme des checksum de tous leurs LSA respectifs est identique.[8]

Des enregistrements sont ajoutés à la base de données suite à la réception d'un nouvel LSA généré par un autre routeur ou par la création de nouveaux LSA au niveau du routeur local.
Des informations sont supprimées de la base de données suite à l'apparition d'une instance plus récente d'un LSA déjà présent (dans ce cas, il s'agit plutôt d'un remplacement) ou parce qu'un LSA a atteint la limite d'âge autorisée.

Lorsqu'un nouveau routeur apparaît ou que deux routeurs voisins se découvrent, un échange des bases de données est directement réalisé pour rejoindre l'état de stabilité du domaine OSPF.

Ces mécanismes (la découverte, la synchronisation et l'inondation) sont réalisés par trois protocoles de OSPF à savoir : le protocole Hello, le protocole d'échange et le protocole d'inondation.

La base de données d'état peut également être utilisée comme un outil d'observation et d'analyse du domaine étant donné qu'elle en décrit la topologie complète.

---

[7] Cette diffusion peut être débutée à partir de n'importe quel routeur du domaine.
[8] Cette règle n'a rien d'absolu mais elle est utilisée en pratique car la probabilité d'une erreur est infime.

c. *La communication entre les routeurs OSPF*

Les routeurs OSPF communiquent entre eux en utilisant des paquets OSPF qui reposent directement au-dessus de la couche IP. La figure 2 donne un aperçu des trois niveaux d'imbrication :

- l'en-tête du paquet IP ;
- le contenu du paquet IP qui se décompose en deux parties :
    - l'en-tête du paquet OSPF ;
    - le contenu du paquet OSPF.

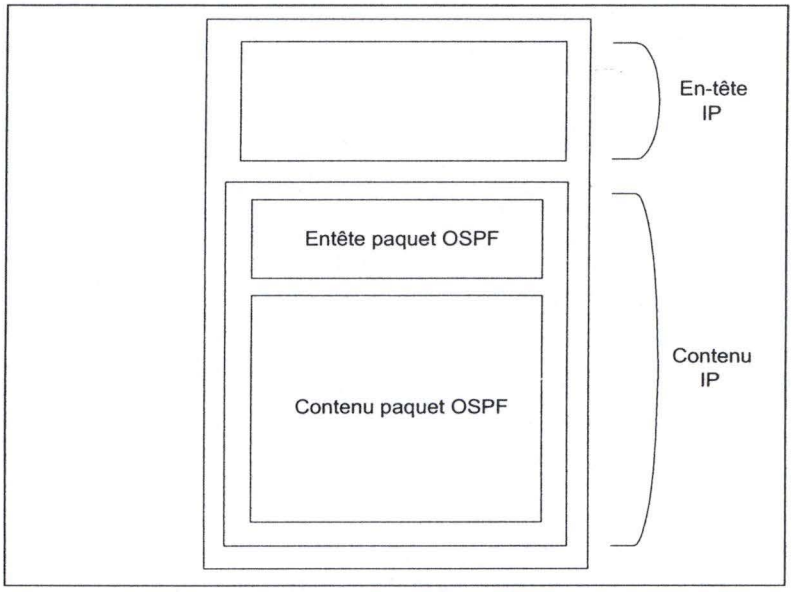


Figure 2 : IP et les paquets OSPF

1) L'en-tête IP

L'en-tête d'un paquet IP pour OPSF présente les particularités suivantes [MOY98b p.85-86] :

- la préséance est mise à la valeur 110 (Internetwork control) pour favoriser les informations de routage par rapport aux données ;
- la fragmentation des paquets IP est autorisée mais n'est pas recommandée ;
- le TTL (Time To Live) est mis à la valeur 1. Un paquet n'est destiné qu'à son voisin immédiat ;
- le numéro de protocole est 89 ;
- l'adresse IP de source est l'adresse de l'interface qui envoie le paquet. Dans le cas d'une liaison point-à-point non numérotée, on prend une autre adresse IP mais dépendante du routeur émetteur. De par ce fait, tout routeur **doit** posséder au moins une adresse IP ;
- l'adresse IP de destination est toujours, soit l'adresse IP du voisin, soit une des deux adresses OSPF multicast.

2) L'en-tête OSPF

On distingue cinq types de paquets OSPF, mais tous ont le même en-tête de 24 octets décrite dans la figure 3 [MOY98a p. 190-192] [HUI94 p .155].

Version	Type
Longueur	
Identifiant du routeur OSPF source	
Identifiant de la zone OSPF	
Checksum	
Type d'authentification	
Authentification	

Figure 3 : L'en-tête du paquet OSPF

Ce qui suit décrit les différents champs composant l'en-tête des paquets OSPF.

Le champ *version* indique la version courante du protocole OSPF à savoir la version 2 actuellement.

Le champ *type* indique le type du paquet OSPF. Il en existe cinq :

1) paquet Hello ;
2) paquet de description de la base de données ;
3) paquet de requête d'état de ligne;
4) paquet de mise à jour d'état de ligne ;
5) paquet d'acquittement d'état de ligne.

Le champ *longueur* exprime la taille totale du paquet OSPF en octets.

Le champ *identifiant du routeur OSPF source* permet d'identifier l'émetteur de manière non ambiguë dans le système autonome. Une solution possible est de prendre la plus petite de ses adresses IP.

Le champ *identifiant de la zone OSPF* associe le paquet avec son niveau hiérarchique[9].

Les champs *type d'authentification* et *authentification* sont utilisés pour implémenter des fonctions de sécurité dans la transmission des paquets.

Le champ *checksum* prend en compte le paquet OSPF complet à l'exception du champ d'authentification.

d. Les différents protocoles de OSPF

OSPF dispose de trois sous-protocoles pour assurer une synchronisation correcte et rapide entre les différentes bases de données. Il s'agit des protocoles suivants :

- *hello* pour la découverte du voisinage ;
- *échange* pour permettre à deux voisins de synchroniser totalement leurs bases de données ;
- *inondation* pour envoyer un message à tous les routeurs du domaine.

[9] Voir infra

1) Le protocole Hello

Les routeurs voisins d'un routeur donné sont ceux avec lequel il peut échanger **directement** des informations de routage. C'est le protocole Hello [MOY98a p. 77-79 et p. 96-99] [HUI94 p.156-158] qui est responsable dans OSPF de la découverte et de l'entretien des relations de *voisinage*.

Régulièrement (toutes les 10 secondes par défaut), un routeur envoie un message Hello sur toutes ses interfaces. Il apprend l'existence d'un routeur voisin en recevant, en retour, un autre message Hello.

La transmission et la réception régulières de paquets Hello permettent donc, pour un routeur, de détecter les éventuelles pannes chez un de ses voisins[10]. En effet, si une interface d'un routeur ne reçoit plus de messages Hello durant un intervalle de temps donné (typiquement 40 secs), ce routeur considère que cette liaison est défectueuse et la supprime de sa base de données.

Les messages Hello servent également à vérifier qu'une connexion entre deux voisins est bien bidirectionnelle. Si ce n'est pas le cas, alors, cette liaison ne peut être utilisée par des paquets de données car une possibilité de perte existe.
Tout routeur recevant un message Hello place son émetteur dans la liste des *routeurs voisins* qu'il connaît. Cette liste est toujours jointe à un message Hello envoyé par un routeur. On vérifie la bidirectionnalité en examinant la liste des voisins communiquée dans le message Hello émis par le voisin et reçu localement : si le routeur local n'est pas présent dedans, alors, le voisin n'a pas reçu le message Hello envoyé par le routeur local et on ne peut pas dire que la liaison est bidirectionnelle. Par contre, si le routeur local est annoncé dans le message Hello émis par le voisin, alors, on peut affirmer que celui-ci a bien reçu le message Hello du routeur local et donc que la liaison est bidirectionnelle.
Le protocole Hello doit aussi s'assurer que les routeurs voisins sont en accord sur les intervalles de retransmission (HelloInterval) et de mort (RouteurDeadInterval).

Finalement, ce protocole est utilisé dans la gestion des routeurs dans le cas de réseau avec ou sans diffusion.

Le format des paquets Hello (auquel on doit ajouter un en-tête OSPF – figure 3) est donné par la figure 4 :

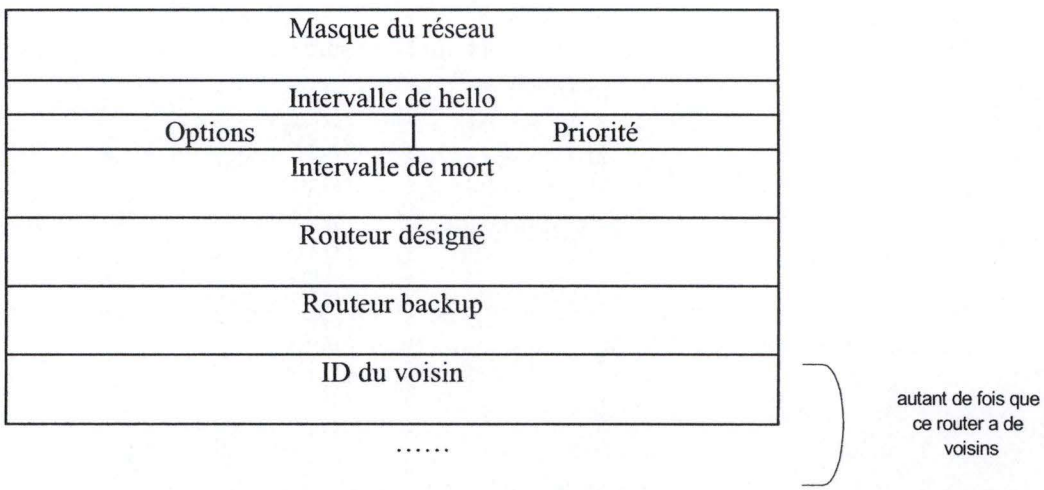


Figure 4 : format du paquet *hello* – type 1

Ce qui suit décrit les différents champs composant le paquet *hello*.

[10] Dans la pratique, une panne de liaison sera détectée beaucoup plus rapidement par les couches inférieures des protocoles de communication.

Le champ *Intervalle de hello* permet à un routeur d'annoncer sa fréquence de répétition des messages *hello*.

Le champ *Intervalle de mort* permet à un routeur d'annoncer la durée à partir de laquelle l'absence de réponse à ses propres messages *hello* est considérée comme une panne dans la liaison entre les deux routeurs.

Le champ *ID du voisin* est répété autant de fois qu'il n'y a de voisins connus par le routeur local. Chaque instance de ce champ contient l'adresse d'un des voisins connus.

Le champ *Options* sert à négocier certaines extensions possibles entre deux routeurs voisins.

Les champs *Priorité*, *Masque du réseau*, *Routeur désigné* et *Routeur backup* sont utilisés lorsqu'on travaille avec des segments de réseau.[11]

2) Le protocole d'échange

Lorsque deux routeurs ont établi une liaison bilatérale (avec le protocole Hello), ils doivent synchroniser leur base de données avant tout échange de paquets de données entre eux. Cette synchronisation est réalisée par le protocole d'échange [MOY98a p. 99-107] [MOY98b p.89-91] qui se déroule en trois phases :

1) choix d'un maître et d'un esclave dans la relation ;
2) échange des descriptions des bases de données respectives ;
3) mise à jour des bases de données.

Le protocole d'échange utilise le paquet de description de la base de données dont la structure est donnée dans la figure 5.

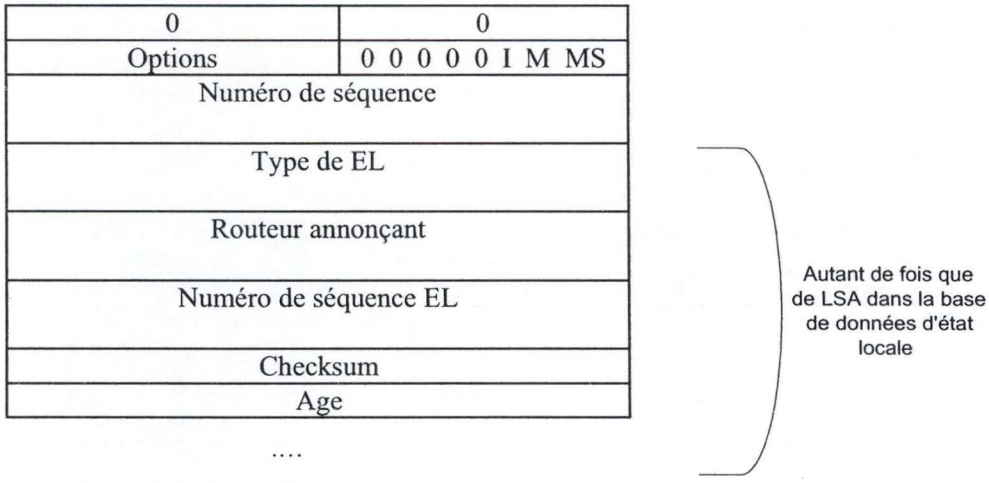


Figure 5 : format du paquet de description de la base de données - type 2

Ce qui suit décrit les différents champs composant le paquet de description de la base de données.

Les deux premiers champs ont une valeur nulle.

Le champ *options* sert à négocier certaines extensions possibles entre deux routeurs voisins.

[11] Voir infra

Le champ suivant n'utilise que ses trois derniers bits à savoir I pour Initialize, M pour Master et MS pour Master/Slave. Ce champ permet de négocier le statut des participants à l'échange et de déterminer l'étape actuelle du processus.

Le champ *numéro de séquence* donne le numéro de séquence du paquet de description courant.

Le paquet se poursuit, si nécessaire, par une liste de descriptions des LSA de la base de données de l'émetteur. Pour chaque LSA, on retrouve son type, le routeur qui en est à l'origine, son numéro d'instance courant, son checksum et son âge.

a) Choix du maître et de l'esclave

Le routeur qui souhaite commencer un échange envoie un paquet de description vide (sans aucune description de LSA) avec les bits I, M et MS mis à 1 indiquant de ce fait qu'il est le maître, et un numéro de séquence quelconque. L'autre routeur accepte d'être l'esclave en renvoyant un paquet de description avec les bits I et M à 1, le bit MS à 0 et le même numéro de séquence que celui choisi par le maître.

On se protège des pertes de paquets par des retransmissions : en l'absence d'acquittement après un certain délai, le paquet initial est retransmis. Les collisions (deux routeurs s'annoncent simultanément comme étant maître) sont réglées en comparant les adresses qui identifient les routeurs; le routeur qui a l'adresse la plus élevée devient alors l'esclave et en informe son vis-à-vis tandis que celui avec la plus petite adresse attend la réponse de l'autre routeur.

En cas d'erreur dans la procédure d'échange, à quelque niveau que ce soit, celle-ci est recommencée depuis le début.

b) Echange des descriptions

Le maître envoie une description des enregistrements de sa base de données dans une suite de paquets de description (type 2) dans lesquels le bit I est mis à 0, le bit MS est mis à 1 et le bit M est mis à 1 sauf pour le dernier paquet. Les paquets sont numérotés séquentiellement et envoyés l'un après l'autre : après chaque paquet de description reçu, l'esclave doit envoyer un acquittement, c'est à dire un paquet de description ayant le même numéro de séquence mais dont le bit MS est mis à 0. Les paquets d'acquittement transportent les descriptions des enregistrements de la base de données esclave. Si le maître ne reçoit pas d'acquittement après un certain intervalle de temps, il répète le dernier paquet. Si l'esclave reçoit un numéro de paquet identique au précédent, il répète son dernier acquittement. Si, suite à l'envoi du dernier paquet par le maître, l'esclave doit encore envoyer des descriptions, ce dernier renvoie un acquittement avec le bit M mis à 1; le maître continuera alors à envoyer des paquets de description vides avec le bit M à 0 et à accepter les acquittements jusqu'à ce qu'il finisse par recevoir un acquittement dont le bit M vaut 0. L'échange est alors terminé et les deux routeurs possèdent une description complète de la base de données de leur vis-à-vis.

Pour chaque description d'enregistrement reçue, le routeur récepteur vérifie s'il possède bien dans sa base de données un enregistrement correspondant et que cet enregistrement a un numéro de séquence supérieur ou égal à celui annoncé. Si ce n'est pas le cas, alors, il place cette description dans une liste d'enregistrements à demander.

c) Mise à jour des bases de données

Au plus tard quand l'échange est terminé, le maître et l'esclave demanderont, s'il y a lieu, à l'aide d'un ou plusieurs paquets de requête d'état de liaison, la description complète des enregistrements qu'ils ne possèdent pas ou qui sont trop vieux.

La structure du paquet de requête d'état de liaison est donnée dans la figure 6.

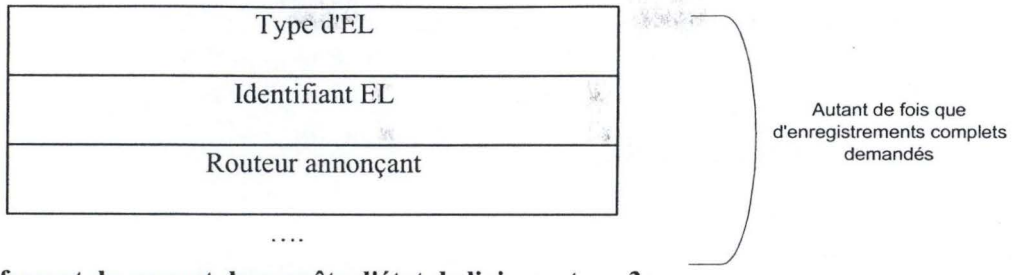


Figure 6 : format du paquet de requête d'état de liaison - type 3

Ce paquet a comme en-tête préalable l'en-tête commun des paquets OSPF. Il contient une liste d'éléments dont chacun décrit un enregistrement particulier que l'émetteur désire recevoir. Un enregistrement est décrit de manière unique par la combinaison de trois champs de 32 bits à savoir les champs *Type d'EL*, *Identifiant EL* et *Routeur annonçant*. Après la réception d'une requête de ce type, un routeur enverra une mise à jour à l'aide de la procédure d'inondation.

Si la liste demandée est trop longue, alors, plusieurs requêtes seront envoyées mais, à un moment donné, il ne peut y avoir qu'une seule requête active. Si, pour une raison quelconque, une requête n'aboutit pas, celle-ci est à nouveau émise avant de passer à la suivante. Lorsque le routeur reçoit une réponse, il efface les enregistrements reçus de la liste des enregistrements à demander et il met sa base de données à jour (aucun acquittement n'est envoyé à l'émetteur). Au moment où cette liste est vide des deux côtés, on considère que les deux bases de données sont synchronisées. On dit dans ce cas que les deux voisins sont *complètement adjacents*.

3) Le protocole d'inondation

Ce protocole [MOY98a p. 143-156] [MOY98b p. 92-95] gère la mise à jour des bases de données suite à des modifications structurelles du domaine OSPF. Il est utilisé dans deux cas :

1) lorsqu'un routeur veut modifier l'état d'une liaison ou supprimer une liaison dont il est à l'origine ;
2) lorsqu'un routeur reçoit une requête d'état de liaison issue d'un autre routeur.

Le routeur d'origine envoie les modifications dans un paquet de mise à jour d'état de liaison. Sa structure est donnée dans la figure 7.

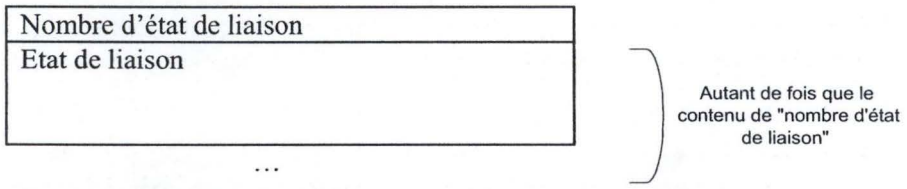


Figure 7 : format du paquet de mise à jour d'état de liaison - type 4

Ce qui suit décrit les différents champs composant le paquet de mise à jour d'état de liaison.

Ce paquet a comme en-tête préalable, l'en-tête commun des paquets OSPF.

Le champ *Nombre d'état de liaison* indique le nombre de LSA qui sont décrits dans le paquet courant.

Le champ *Etat de liaison* contient la description d'un enregistrement de la base de données.

La figure 8 [MOY98a p. 143-146] décrit l'algorithme réalisé par un routeur lorsqu'il reçoit un paquet de mise à jour (on ne tient pas compte ici de l'âge des LSA ni des particularités de la procédure d'échange).

```
Vérifier la non-corruption du message

Rechercher l'enregistrement dans la base de données

Si l'enregistrement n'est pas présent, alors

    L'ajouter à la base de données
    Diffuser le message sur toutes ses interfaces à l'exception de celle par laquelle est venu le message
    Envoyer un acquittement vers le routeur émetteur du message

Si l'enregistrement est présent, alors, comparer des numéros de séquence du LSA de la base de données
locale avec le LSA contenu dans le paquet

    Si le numéro de séquence du LSA de la base de données est plus petit que celui du paquet reçu, alors

        Remplacer le LSA de la base de données par celui contenu dans le paquet
        Diffuser le message sur toutes ses interfaces à l'exception de celle par laquelle il est venu
        Envoyer un acquittement vers le routeur émetteur du message

    Si les numéros de séquence sont égaux

        Envoyer un acquittement vers le routeur émetteur du message

    Si le numéro de séquence du LSA de la base de données est plus grand que celui du paquet reçu, alors

        Envoyer sa propre version du LSA uniquement au routeur qui a émis le LSA trop âgé
```

Figure 8 : algorithme de gestion des paquets de mise à jour

Cet algorithme est effectué par tout routeur qui reçoit un paquet de mise à jour. Ainsi, seul les messages **utiles** sont diffusés de proche en proche pour inonder tout le réseau et ce, jusqu'au moment où tous les routeurs possèdent le LSA à jour dans leur base de données.

Par soucis de fiabilité, un routeur retransmettra régulièrement le message de mise à jour tant qu'il n'aura pas reçu un message d'acquittement émis par le destinataire. La structure d'un paquet d'acquittement (type 5) est identique à celle utilisée pour le paquet de description de la base de données.

Chaque paquet d'acquittement peut contenir plusieurs acquittements. Il est habituel de retarder l'émission d'acquittements afin de regrouper ceux-ci dans un même paquet. Le retard doit être calculé de façon à éviter les retransmissions inutiles. Cependant, il ne faut pas retarder l'émission de paquets d'acquittement concernant des LSA qui ne modifient pas la base de données car ces LSA sont probablement le résultat de répétitions dues à des pertes d'acquittement.

Toutes les mises à jour ne requièrent cependant pas un acquittement. Ainsi, lorsque deux mises à jour identiques se croisent, la mise à jour reçue est considérée comme étant un acquittement implicite. La règle générale est : **in any link direction, one update or acknowledgment is sent but never both** [MOY98b p. 93].

Cette technique assure une bonne fiabilité du domaine et de son routage même en cas d'erreur de transmission ou de pannes de liaison car [MOY98b p. 94-95] :

- tant que l'arbre minimal qui relie tous les nœuds du réseau est opérationnel, les différentes bases de données resteront synchronisées ;

- toutes les 30 minutes par défaut, chaque routeur doit rafraîchir les LSA dont il est l'auteur après avoir incrémenté leur numéro de séquence. Ainsi, si un routeur voit sa base de données corrompue, dans les 30 minutes qui suivent, celle-ci sera automatiquement re-synchronisée ;
- les LSA contiennent un checksum qui, si sa vérification s'avère incorrecte, empêche l'envoi d'un acquittement, ce qui provoquera automatiquement un nouvel envoi rapide de cet LSA ;
- un routeur OSPF incrémente la valeur du champ *Age* d'un LSA quand il place celui-ci dans un paquet de mise à jour. Ce comportement permet de casser un effet de boucle étant donné qu'en cas de bouclage, le champ *Age* atteindra à un moment donné la valeur du MaxAge et le LSA sera alors automatiquement rejeté ;
- on évite les changements trop rapides en imposant un intervalle minimum entre deux mises à jour d'un même LSA. Cela permet d'éviter un trafic de maintenance excessif dans le domaine ;
- de la même manière, un routeur refusera de mettre à jour sa base de données si la dernière mise à jour a été réalisée moins d'une seconde avant la nouvelle demande de changement. Cette technique a pour but de limiter les perturbations dues au comportement défectueux d'un routeur à ses seuls voisins.

La figure 9 donne un exemple de propagation d'un message de mise à jour généré par la station *10.1.1.3* et de son acquittement dans un domaine OSPF. On supposera ici que l'acquittement est retardé.

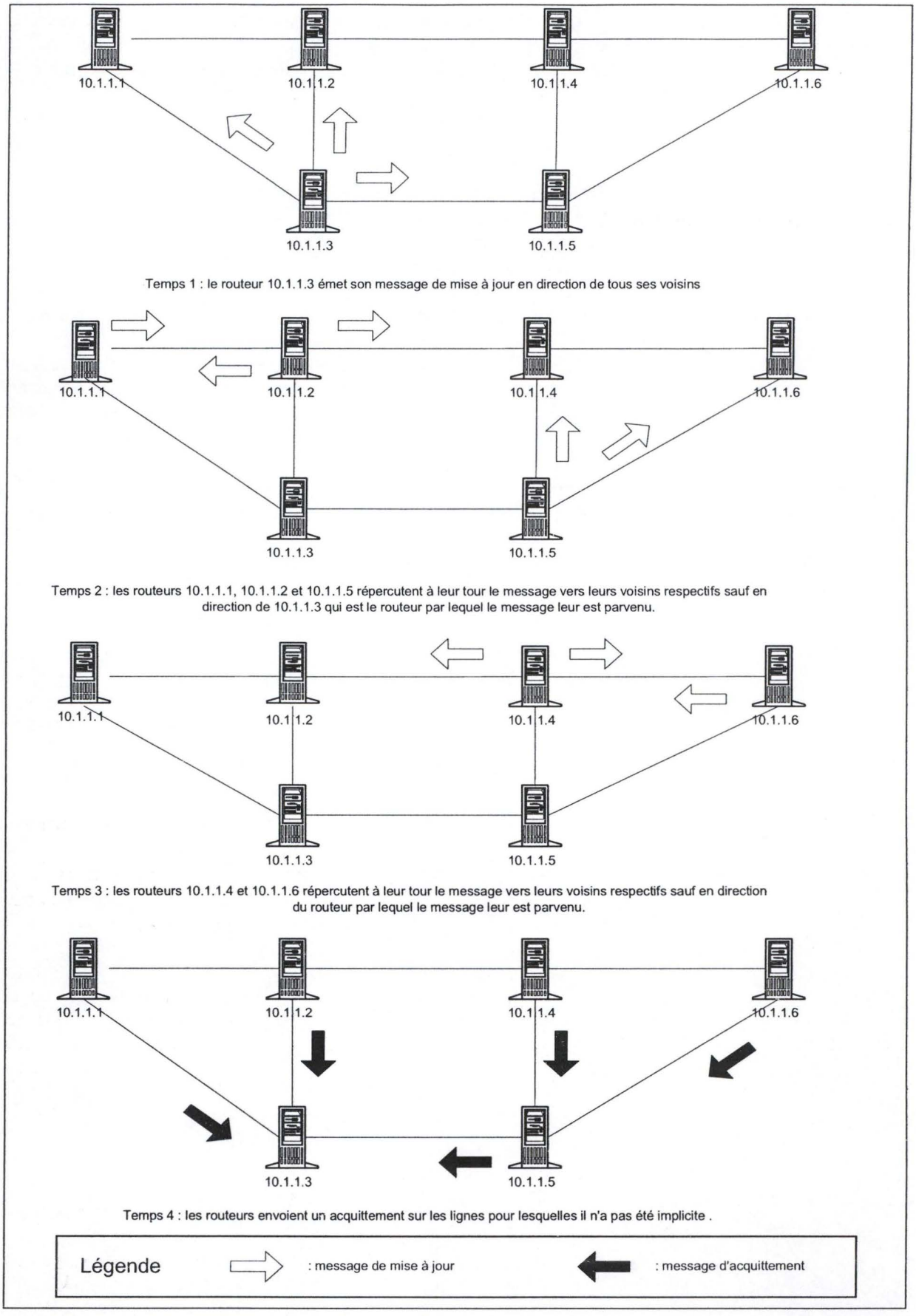


Figure 9 : exemple de propagation de mise à jour et d'acquittement

3. Autres concepts de OSPF

Suivant la technologie utilisée pour le domaine, les implémentations de OSPF sont aménagées. Le modèle simplifié utilisé jusqu'ici ne considérait que des liaisons point-à-point entre les différents routeurs OSPF. Cependant, dans la réalité, les routeurs sont généralement intégrés dans des réseaux à diffusion (exemple : Ethernet) ou des réseaux sans diffusion (exemple : X25). Les différences portent essentiellement sur trois points :

1) la découverte et la maintenance des relations de voisinage ;
2) la synchronisation des bases de données ;
3) la représentation abstraite de sous-partie du réseau.

La conception hiérarchique de OSPF lorsque son domaine est divisé en différentes zones et la manière dont un domaine OSPF collabore avec ses voisins seront ensuite abordés.

a. *OSPF dans les réseaux à diffusion*

Dans le modèle IP, un réseau à diffusion est souvent divisé en segments. Toutes les stations sur un même segment ont un même préfixe; on dit qu'elles appartiennent au même sous-réseau IP. La découverte du préfixe d'une adresse IP nécessite la connaissance du masque de ce sous-réseau afin de déterminer la portion de l'adresse qui identifie le sous-réseau. Ainsi, si le masque du sous-réseau est de 24 (on ne prend en compte que les 24 premiers bits de l'adresse IP qui elle, en compte 32), alors, par exemple, toutes les stations ayant une adresse comprise entre 10.3.2.0 et 10.3.2.255 sont dans le même sous-réseau IP tandis que la station 10.3.3.2 appartient à un autre sous-réseau.

IP fait du routage vers les sous-réseaux et non vers des adresses individuelles. Quand un paquet IP est redirigé par un routeur, il l'est toujours vers le sous-réseau qui est le plus spécifique dans sa table de routage. Par exemple, si la table de routage renseigne que le trafic à destination de 10.1.1.0/8 doit être envoyé par l'interface 1 et que le trafic à destination de 10.1.1.0/24 doit être envoyé par l'interface 2, le paquet ayant pour destination 10.1.1.20 sera dirigé vers l'interface 2 tandis que le paquet 10.2.2.20 sera lui dirigé vers l'interface 1.
Les règles suivantes sont d'application pour la gestion des sous-réseaux IP :

- deux hôtes sur deux sous-réseaux différents ne peuvent échanger de l'information que via un routeur ;
- deux hôtes sur un même sous-réseau peuvent échanger directement des informations ;
- deux routeurs ne peuvent échanger directement des informations de routage que s'ils ont un ou plusieurs sous-réseaux IP en commun (Il faut qu'il partage un préfixe commun).

C'est le protocole Hello qui garantit l'application du troisième point. En effet, un routeur qui envoie un message Hello sur une de ces interfaces inclut dans ce message sa propre adresse IP pour cette interface (champ *Identifiant du routeur OSPF source* de l'en-tête du paquet OSPF) et le masque de sous-réseau associé (champ *Masque du réseau* dans le corps du message Hello). Un routeur n'acceptera le message Hello envoyé que si les deux routeurs impliqués ont le même masque de sous-réseau et si les deux interfaces sont attachées au même sous-réseau (le préfixe des deux routeurs doit être identique).

Un réseau IP à diffusion offre deux services particuliers :

1) *broadcasting* qui permet à une station d'envoyer un message à toutes les autres stations de son sous-réseau en n'émettant qu'un seul exemplaire de ce message ;
2) *multicasting* qui permet à une station d'envoyer un message à un groupe de station de son sous-réseau en n'émettant qu'un seul exemplaire de son message et en étant sûr que les stations ne faisant pas partie du groupe ne recevront pas ce message.

1) Découverte et maintenance

Périodiquement, un routeur dans un réseau à diffusion [MOY98b p. 105-106] envoie un paquet Hello sur l'adresse multicast AllSPFRouteurs (224.0.0.5)[12]. Ce paquet contient entre autres l'adresse du routeur désigné et du routeur backup du sous-réseau tel que l'émetteur les perçoit ainsi que les adresses des routeurs desquels il a reçu un message Hello récemment.
La diffusion régulière des messages Hello via le multicasting permet d'assurer une découverte automatique des relations de voisinage avec un minimum d'échange de données et sans déranger les stations ne participant pas au protocole OSPF.

2) Synchronisation des bases de données

Pour éviter un échange trop grand de données de synchronisation [HUI94 p. 138-141] entre tous les routeurs appartenant à un même sous-réseau, un routeur désigné et un routeur backup sont choisis; tous les routeurs *normaux* de ce sous-réseau n'établiront de relation de synchronisation qu'avec ces deux routeurs.

Le routeur désigné remplit essentiellement deux fonctions :

1) jouer le rôle central dans la synchronisation des bases de données des routeurs de ce sous-réseau ;
2) créer les états de liaison réseau qui décrivent les routeurs attachés actuellement au sous-réseau.

La figure 10 montre les liaisons à réaliser sans l'utilisation de routeurs désignés et backup et puis avec leur utilisation.

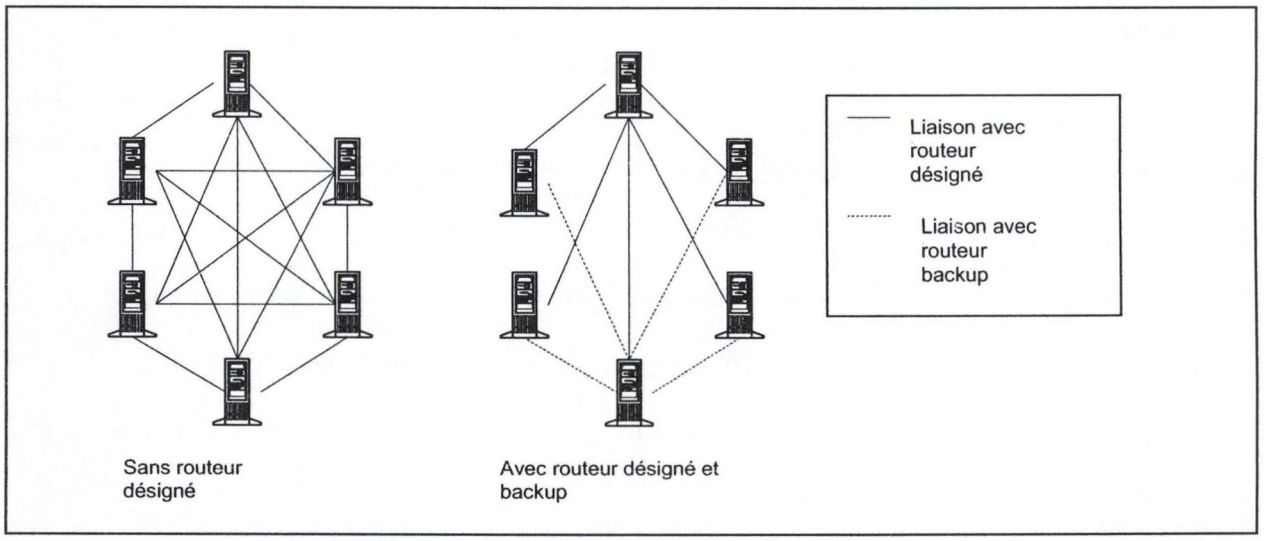


Figure 10 : gestion des états de liaison dans un réseau à diffusion

Le premier routeur à rejoindre un sous-réseau reçoit directement l'étiquette de routeur désigné. Le second routeur à joindre le sous-réseau devient automatiquement le routeur backup. A partir de ce moment, l'algorithme d'élection [HUI94 p. 158-159] peut fonctionner en régime. Il se base sur la valeur du champ *priorité* du message *hello*.

1) lorsqu'un routeur rejoint le sous-réseau, il envoie un message *hello* dans le sous-réseau en mettant les champs *routeur désigné* et *routeur backup* à la valeur nulle. En faisant cela, il se met dans une position d'écoute (durant un intervalle de 40 sec) pour

[12] Ce groupe contient tous les routeurs OSPF d'un même sous-réseau.

mémoriser la priorité de chaque voisin, l'état de leur relation (unilatérale ou bilatérale) ainsi que pour déterminer les routeurs désignés et backup actuels ;

2) sur base des informations recueillies, le routeur local détermine le nouveau routeur backup en tenant compte de la liste qu'il a établie et en s'incluant dedans. Si plusieurs routeurs se sont déjà déclarés comme backup, celui avec la plus grande priorité et l'identifiant le plus élevé est choisi. Si personne ne s'est déclaré comme backup, on choisit alors le routeur avec la priorité et l'identifiant les plus élevés et qui ne se déclare pas comme désigné ;
3) le routeur détermine alors le nouveau routeur désigné de la même manière que pour le second point. Si personne ne s'est déclaré comme désigné, le routeur choisi à la seconde étape comme backup devient le désigné et on recommence les étapes 2 et 3.

Cet algorithme (point 2 et 3) est exécuté par chaque routeur après la réception de chaque paquet *hello*, ce qui garantit tout à la fois une stabilité et une convergence rapide en cas de modification de la configuration et des priorités.

Lorsqu'un sous-réseau est partagé en deux morceaux, chacune des parties devenues *autonomes* se choisit un nouveau routeur désigné et un routeur backup. Lorsque deux sous-réseaux fusionnent, le routeur désigné qui a la plus haute priorité devient le nouveau routeur désigné.

Comme mentionné précédemment, le routeur désigné joue un rôle central dans la synchronisation des bases de données des routeurs du sous-réseau. Le protocole d'inondation est modifié. Ainsi, si un routeur quelconque du réseau à diffusion reçoit par une de ses interfaces un LSA qu'il faut modifier ou s'il décide d'en modifier un dont il est l'émetteur, la séquence suivante de mise à jour est exécutée :

1) le routeur *source* adapte sa base de données locale ;
2) ce routeur envoie un paquet de mise à jour d'état de liaison en utilisant l'adresse multicast AllDRouteur (224.0.0.6) qui n'est écoutée que par le routeur désigné et son backup dans le réseau à diffusion ;
3) le routeur désigné répercute ce message vers les autres routeurs de son réseau à diffusion en l'envoyant cette fois sur l'adresse AllSPFRouteur (224.0.0.5) qui est écoutée par tous les routeurs OSPF du réseau à diffusion ;
4) si le routeur backup ne voit pas venir de messages sur l'adresse AllSPFRouteur, il prend l'initiative, après un certain délai, d'envoyer le LSA qu'il avait reçu au second point car il considère alors que le routeur désigné ne l'a pas envoyé .

3) Abstraction

La présence d'un routeur désigné permet également de réduire le nombre d'état de liaison dans la base de données [MOY98b p. 108-110]. En effet, si on devait représenter toutes les relations entre les N routeurs participant à un sous-réseau, on obtiendrait N*(N-1) liaisons orientées différentes. Au lieu de cela, on va représenter le réseau à diffusion par un ensemble de liens entre les routeurs et un *routeur virtuel* qui représente le sous-réseau, de manière à former un réseau en étoile. On aura deux liens par routeur, de et vers le *routeur virtuel* (qui sera le centre du réseau en étoile). La liaison entre le routeur et le réseau sera annoncée par le routeur dans son propre état de liaison tandis que la liaison en sens inverse sera annoncée par le routeur désigné du sous-réseau avec un état de liaison dit de réseau (type 2 de LSA)[13]. Cet état de liaison réseau aura un poids de valeur nulle et contiendra la liste des routeurs qui ont leur base de données synchronisée avec celle du routeur désigné. Seul le routeur désigné émet un état de liaison réseau. De cette manière, la vision du sous-réseau reste conforme à la manière dont les informations sont échangées entre les routeurs de ce sous-réseau.

[13] Le routeur désigné utilise l'adresse de son interface avec le sous-réseau comme valeur pour le champ *ID d'état de liaison*. La combinaison de cette information avec le masque du sous-réseau permet de communiquer le préfixe de ce sous-réseau à tout le domaine OSPF grâce au protocole d'inondation.

b. OSPF dans les réseaux sans diffusions

Les protocoles IP peuvent fonctionner sur les réseaux à commutation de circuits virtuels comme X25 et ATM. Ce type de réseau fournit une connectivité complète à toutes les stations participantes (deux stations peuvent ainsi communiquer directement) mais la grande différence avec les réseaux à diffusion vient du fait qu'on ne peut faire ni du broadcasting ni du multicasting.

Il est important dans ce genre de réseau de limiter au maximum les communications entre les routeurs étant donné que le coût des communications est souvent fixé en fonction de la quantité d'informations échangées.

OSPF manipule les réseaux sans diffusion de la même manière que les réseaux à diffusion mais en tenant compte de leurs particularités. [MOY98b p.111-114]

1) Découverte et maintenance

Comme ce genre de réseau ne dispose pas de capacité de découverte, tous les routeurs OSPF voisins dans un même sous-réseau doivent être indiqués dans la configuration initiale du routeur ainsi que leur capacité à devenir désigné ou backup.

Pour implémenter l'élection du routeur désigné, tous les routeurs éligibles s'échangent des messages Hello à intervalle régulier (il faut, ici, envoyer une copie du message par routeur existant qu'on veut rejoindre; ainsi, si 10 routeurs sont éligibles, il faudra envoyer 90 messages différents alors que dans un réseau à diffusion, 10 messages auraient suffit).

Le fait d'avoir un routeur désigné et un routeur backup n'a pas d'effet sur les circuits qui devront être établi entre les routeurs. Ces derniers serviront à la transmission des paquets de données et pourront être établis à la demande tandis que les circuits liant un routeur à son routeur désigné et son routeur backup devront être quasi permanents étant donné la fréquence d'échange des informations de routage (Les messages HELLO sont envoyés toutes les 10 secondes par défaut).

2) Synchronisation des bases de données

La synchronisation est semblable à celle utilisée dans les réseaux à diffusion. L'inondation passe par le routeur désigné qui propage l'information vers les autres routeurs de son réseau. La seule différence vient du fait que le message de mise à jour des liaisons doit être répliqué et envoyé personnellement à tous les autres routeurs par le routeur désigné.

3) Abstraction

La représentation du sous-réseau dans la base de données est identique à celle des sous-réseaux à diffusion. Un LSA de réseau est créé pour le sous-réseau par le routeur désigné et tous les LSA créés par les routeurs de ce sous-réseau contiennent un lien vers le sous-réseau.

c. La conception hiérarchique de OSPF

Si la taille d'un réseau IP augmente, le nombre de routeurs augmentera dans une proportion équivalente ainsi que la taille de la base de données d'état de liaison. Le but du routage hiérarchique[HUI94 p. 142-147] est de réduire la taille de cette base de données et donc le trafic de routage associé en découpant le domaine en un ensemble de parties indépendantes appelées zones et connectées ensemble par un backbone qui sera responsable de la distribution des informations de routage entre les différentes zones.

Chaque zone est considérée comme étant indépendante. Un sous-réseau n'appartient qu'à une seule zone; une interface d'un routeur n'est connectée qu'à une seule zone. La base de données des routeurs

d'une zone contient toutes les informations concernant cette zone mais elle ne possède qu'une esquisse de la structure des autres zones. Le protocole d'inondation s'arrête à la frontière d'une zone cachant ainsi les détails de sa topologie à ses voisins. Quand il s'agit de naviguer hors de sa zone, un routeur s'appuie sur le niveau supérieur pour localiser la zone qui contient l'adresse de destination. Cette méthode a cependant un désavantage : comme un routeur ne dispose pas de la structure complète du réseau, il est possible que le chemin donné pour une destination ne soit le chemin optimal mais bien le meilleur chemin que l'on connaisse au regard des informations disponibles.

Une zone est identifiée par un ID (l'identifiant du backbone est 0.0.0.0) et se compose d'une collection de segments de réseaux connectés par des routeurs. Le routage à l'intérieur d'une zone est tel qu'il a été décrit dans les points précédents (on qualifie celui-ci de routage plat).
Pour coller ensemble les différentes zones au backbone, il est nécessaire que certains routeurs appartiennent à plusieurs zones. Ces derniers portent le nom de routeur interzone (area border routeur). Il y en a au moins un par zone et sa fonction est double :

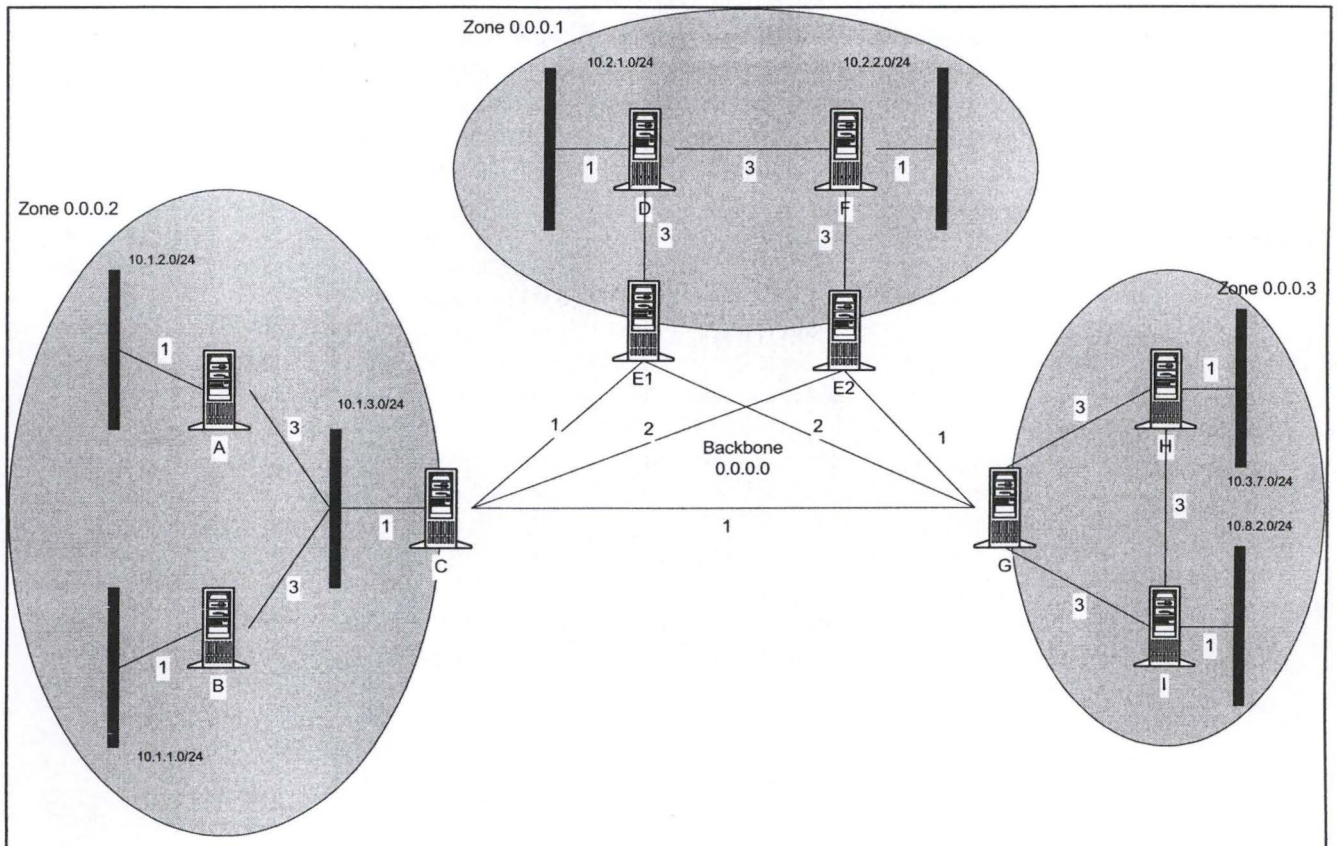
- communiquer à sa propre zone, les informations concernant les autres zones ;
- diffuser les informations d'adressage récapitulatives à l'intention des autres zones à l'aide d'état de liaison récapitulatif (LSA type 3). Cet état de liaison résume les adresses IP comprises dans sa zone sous la forme d'un préfixe (si plusieurs préfixes sont nécessaires pour décrire la zone, on enverra alors plusieurs états de liaison récapitulatifs). Cet état de liaison récapitulatif reçoit pour poids le coût de la liaison entre le routeur interzone émetteur et l'élément le plus éloigné dans la zone qu'il représente (l'élément le plus éloigné est fonction du métrique utilisé).

L'échange d'informations de routage entre les zones se déroule comme suit [MOY98b p. 125] :

1) chaque routeur interzone annonce un résumé des adresses de la zone à laquelle il est connecté par la génération d'un message d'état de liaison récapitulatif sur le backbone ;
2) chaque routeur interzone collecte alors, grâce à la procédure d'inondation, les états de liaison récapitulatifs de tous les autres routeurs interzones. Ces routeurs ont alors une image complète du backbone et un résumé de chacune des zones annoncées ;
3) pour chaque destination donnée, le routeur interzone examine tous les LSA annonçant cette destination et il utilise celui avec le coût minimal pour créer une entrée dans sa table de routage. Il annonce ensuite cette information dans sa propre zone d'attache à l'aide d'un LSA d'état de liaison récapitulatif local. Ainsi, les routeurs internes à une zone sauront comment atteindre une destination dans une autre zone et son coût d'accès. Le coût sera équivalent à la somme du coût annoncé dans le LSA récapitulatif émis par le routeur interzone distant, du coût du chemin entre le routeur local interzone et le routeur interzone distant et du coût de la liaison entre le routeur local et son routeur interzone local.

Si plusieurs routeurs interzones d'une même zone annoncent la même destination, le routeur local pourra choisir celui qui est le plus favorable pour lui-même. Cette remarque et la manière dont un routeur interzone choisit l'accès à une zone distante indiquent que le calcul du coût se base ici essentiellement sur un protocole à vecteur de distance. Cela explique le fait que toutes les zones doivent être connectées au backbone afin d'éviter la création de boucle. Cependant, la connexion ne doit pas être obligatoirement physique, elle peut aussi se faire au travers d'une autre zone à l'aide d'une liaison virtuelle.

La figure 11 [MOY98b p.123] donne une représentation d'un domaine en différentes zones. Elle donne également un exemple des tables de routage simplifiées des routeurs qui composent le domaine.



Destination	Coût	Suivant
10.1.2.0/24	1	-
10.1.3.0/24	3	-
10.1.1.0/24	7	B
10.2.0.0/16	12	C
10.3.0.0/16	9	C
10.8.0.0/16	9	C

Table de routage A

Destination	Coût	Suivant
10.2.1.0/24	1	-
10.2.2.0/24	4	F
10.1.0.0/16	9	E1
10.3.0.0/16	9	E1
10.8.0.0/16	9	E1

Table de routage D

Destination	Coût	Suivant
10.3.0.0/16	4	H
10.8.0.0/16	4	I
10.1.0.0/16	6	C
10.2.0.0/16	8	E2

Table de routage G

Destination	Coût	Suivant
10.1.2.0/24	7	A
10.1.3.0/24	3	-
10.1.1.0/24	1	-
10.2.0.0/16	12	C
10.3.0.0/16	9	C
10.8.0.0/16	9	C

Table de routage B

Destination	Coût	Suivant
10.2.1.0/24	4	D
10.2.2.0/24	7	D
10.1.0.0/16	6	C
10.3.0.0/16	6	G
10.8.0.0/16	6	G

Table de routage E1

Destination	Coût	Suivant
10.3.0.0/16	1	-
10.8.0.0/16	4	I
10.1.0.0/16	9	G
10.2.0.0/16	11	G

Table de routage H

Destination	Coût	Suivant
10.1.2.0/24	5	A
10.1.3.0/24	1	-
10.1.1.0/24	5	B
10.2.0.0/16	8	E1
10.3.0.0/16	5	G
10.8.0.0/16	5	G

Table de routage C

Destination	Coût	Suivant
10.2.1.0/24	7	F
10.2.2.0/24	4	F
10.1.0.0/16	7	C
10.3.0.0/16	5	G
10.8.0.0/16	5	G

Table de routage E2

Destination	Coût	Suivant
10.3.0.0/16	4	H
10.8.0.0/16	1	-
10.1.0.0/16	9	G
10.2.0.0/16	11	G

Table de routage I

Destination	Coût	Suivant
10.2.1.0/24	4	D
10.2.2.0/24	1	-
10.1.0.0/16	10	E2
10.3.0.0/16	8	E2
10.8.0.0/16	8	E2

Table de routage F

Figure 11 : Exemple de domaine OSPF hiérarchique

1) Les liaisons virtuelles

Le principe des liaisons virtuelles est d'autoriser les états de liaison récapitulatifs d'une zone non connectée physiquement au backbone à transiter de manière transparente au travers d'une ou plusieurs autres zones de telle façon à aboutir finalement sur le backbone. On maintient ainsi une structure en étoile avec le backbone au centre.

Une liaison virtuelle fait partie intégrante du backbone et se comporte comme une liaison point-à-point entre deux routeurs. En pratique, une grande partie du backbone est composée de la sorte.

Le coût de la liaison est calculé de la même manière qu'au point précédent. A ce résultat, il faut ajouter le coût associé à la liaison virtuelle.

Il est important de remarquer que les informations de routage doivent transiter obligatoirement par le backbone mais que les paquets de données ne sont pas obligés de le faire, ceux-ci devant toujours prendre le chemin le plus court qui est donné dans la table de routage créée à partir de l'analyse de la base de données d'état.

d. Le routage externe

Un domaine OSPF est un système autonome. A ses frontières, il possède des routeurs utilisant deux protocoles de routage différents. Le premier protocole est OSPF et le second est celui qui gère les relations entre les systèmes autonomes. Le rôle de ces routeurs frontières [MOY98a p. 127-132] ou ASBR (Autonomous System Boundary Routeur), est d'obtenir des informations issues de l'extérieur et de les importer dans le domaine OSPF. Ainsi, les routeurs internes au domaine pourront choisir le bon routeur frontière lorsqu'ils devront envoyer des paquets vers l'extérieur.

Les informations extérieures sont importées dans des états de liaison externes (type 5) créés par les ASBR. Un état de liaison externe annonce un seul préfixe. Cependant, comme la taille de l'Internet est importante, plusieurs milliers de routes peuvent être apprises par une liaison extérieure causant la création de milliers d'états de liaison externe risquant d'encombrer inutilement la base de données d'état. Un tri peut donc s'imposer dans le choix des routes externes importées, obligeant les routeurs à utiliser des valeurs par défaut pour certaines destinations.

On distingue deux types de niveaux dans le routage externe :

1) type 1 lorsque les métriques utilisés dans le domaine OSPF et à l'extérieur de ce domaine sont de même type (par exemple : le nombre de saut) ;
2) type 2 lorsque le métrique du coût externe du chemin fournit des valeurs plus importantes que son coût interne (par exemple : le nombre de systèmes autonomes traversés). Dans ce cas, seul le coût externe est pris en compte dans l'évaluation du chemin le plus court.

Il existe donc, en résumé, quatre niveaux différents de routage dans OSPF. Le protocole privilégiera toujours celui qui est le plus local. L'ordre décroissant de priorité est le suivant :

- routage interne à la zone ;
- routage entre zones ;
- routage extérieur de type 1 ;
- routage extérieur de type 2.

Les états de liaison externes sont diffusés tels qu'ils ont été appris et sont stockés dans la base de données de tous les routeurs du domaine OSPF. Aucune conversion n'est faite pour éviter une

redondance d'informations dans la base de données au cas où un système autonome serait connecté par plusieurs ASBR[14].

Mais, pour pouvoir utiliser cette information, tous les routeurs du domaine doivent pouvoir localiser le routeur frontière qui est à l'origine du message d'état de liaison externe. Pour ce faire, OSPF annonce l'emplacement de ses ASBR avec un état de liaison récapitulatif - ASBR (type 4). A l'instar de l'état de liaison externe, l'état de liaison récapitulatif – ASBR est diffusé dans tout le domaine OSPF[15].

4. Les différents types d'état de liaisons

OSPF possède dans sa version initiale cinq types différents de LSA. Tous ces LSA sont réunis dans la base de données qui définit l'état courant du réseau. Ce qui suit décrit la structure du corps des différents LSA.

a. *Les liaisons de routeur (type 1)*

Les enregistrements d'états de liaison routeur [MOY98a p. 206-209] [HUI94 p. 149-150] récapitulent toutes les liaisons (état et coût) qui partent des interfaces du routeur qui annonce cet état. Pour ce type de LSA, l'identifiant d'état de liaison est mis à la valeur de l'identifiant du routeur dans le domaine OSPF. La figure 12 en donne la description. Ce type de LSA peut regrouper si nécessaire la description de plusieurs liaisons.

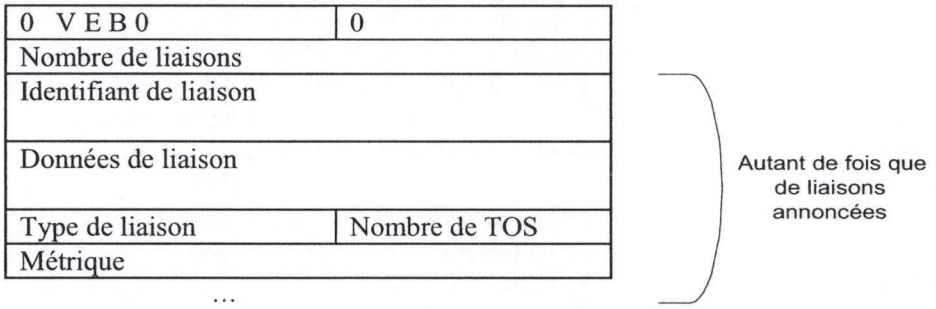


Figure 12 : LSA de liaison de routeur (type 1)

Ce qui suit décrit les différents champs composant le LSA de type 1.

Le bit *V* mis à 1 indique que le routeur est le point final d'une ou plusieurs liaisons virtuelles.
Le bit *E* mis à 1 indique que le routeur est un routeur externe.
Le bit *B* mis à 1 indique que le routeur est un routeur interzone.

Le champ *Nombre de liaisons* indique le nombre de liaisons qui sont décrites. Dans la suite on retrouvera autant de fois de description de liaison que le nombre donné.

Chaque liaison sera décrite par les champs qui suivent.

Les champs *Identifiant de liaison*, *Données de liaison* et *Type de liaison* prennent des valeurs en fonction du type de liaison décrite. Le tableau suivant en donne un récapitulatif simplifié.

[14] La solution de créer un nouvel état de liaison externe aux frontières d'une zone est plus coûteuse en terme de taille de base de données et de calcul que de laisser passer simplement l'état de liaison externe.
[15] On parle dans ce cas de *flooding scope* pour le distinguer du *area flooding scope*.

	Type	Identifiant de liaison	Valeur
1	Connexion point-à-point	ID du routeur voisin	IfIndex
2	Connexion à un réseau	Adresse IP du routeur désigné	Adresse IP de l'interface du routeur
3	Liaison virtuelle	ID du routeur voisin	Adresse IP de l'interface du routeur

Le champ *Nombre de TOS* indique le nombre de services différents gérés par cette liaison.

Le champ *Métrique* indique le coût relatif de l'envoi d'un paquet sur cette liaison.

b. Les liaisons de réseau (type 2)

Ce type de liaison [MOY98a p. 210-211] [HUI94 p. 150-151] est créé par le routeur désigné dans un sous-réseau pour décrire l'ensemble des routeurs actuellement connectés à ce sous-réseau. La valeur du champ *ID de liaison d'état* indique dans ce cas l'adresse du routeur désigné qui initie ce message. Cette liaison n'est créée que si au moins une relation avec un routeur du réseau est complètement adjacente. Sa structure est donné dans la figure 13.

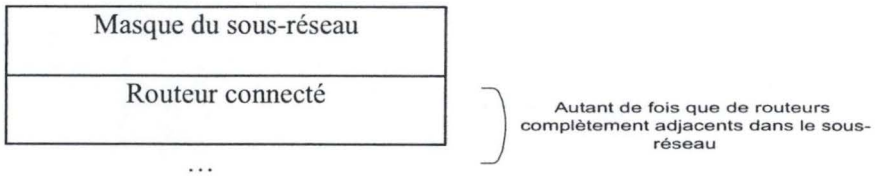


Figure 13 : LSA de liaison de réseau (type 2)

Ce qui suit décrit les différents champs composant le LSA de type 2.

Le champ *Masque du sous-réseau* donne le masque pour le sous-réseau considéré.

Le champ *Routeur connecté* donne l'identifiant du routeur OSPF qui est connecté et complètement adjacent à ce sous-réseau. Ce champ est répété autant de fois qu'il n'y a de routeurs connectés à ce sous-réseau. Le routeur désigné s'inclut lui-même dans la liste.

c. Les liaisons récapitulatives (type 3)

Ce type de liaison [MOY98a p. 212-213] [HUI94 p. 151] est créé par un routeur interzone. Si un tel routeur a besoin d'annoncer plusieurs liaisons récapitulatives, il devra envoyer un paquet par liaison. Pour ce genre de liaison, le champ *ID de l'état de liaison* de l'en-tête du LSA est mis à la valeur d'une adresse IP du réseau annoncé. La structure de ce LSA est donnée dans la figure 14.

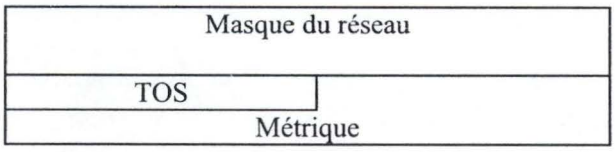


Figure 14 : LSA de liaison récapitulative (Type 3)

Ce qui suit décrit les différents champs composant le LSA de type 3.

Le champ *Masque du réseau* indique la valeur du masque pour le réseau annoncé dans le LSA. On doit le combiner avec le champ *ID de l'état de liaison* de l'en-tête du LSA.

Le champ *TOS* prend toujours la valeur 0.

Le champ *Métrique* indique le poids du chemin entre ce routeur et l'élément le plus éloigné dans la zone qu'il représente.

d. Les liaisons récapitulatives – ASBR (type 4)

Ce type de liaison répond aux même conditions que les liaisons de type 3. Cette fois cependant, le champ *ID de l'état de liaison* est mis à la valeur de l'identifiant du ASBR annoncé par cette liaison. Le champ *Masque du réseau* est présent mais sa valeur n'a pas d'objet. Le champ *Métrique* indique le poids du chemin entre le réseau annoncé et le ASBR qui diffuse le LSA.

e. Les liaisons externes (type 5)

Les liaisons externes [MOY98a p. 214-215] [HUI94 p. 152-153] sont annoncées par les routeurs externes. On ne peut annoncer qu'une seule destination par enregistrement. Le champ *ID de l'état de liaison* de l'en-tête du LSA spécifie l'adresse IP d'un réseau. La structure de ce LSA est donnée à la figure 15.

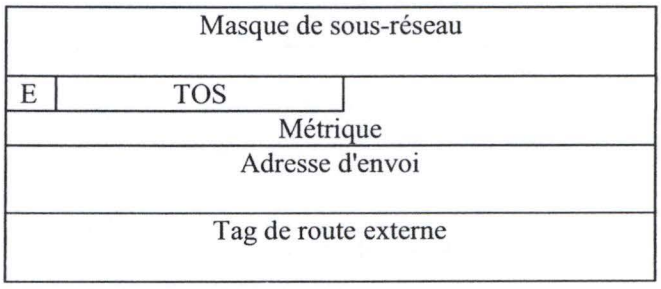


Figure 15 : LSA de liaison externe (type 5)

Ce qui suit décrit les différents champs composant le LSA de type 5.

Le champ *Masque du réseau* indique la valeur du masque d'adresse pour le réseau annoncé dans le LSA.

Le bit *E* indique le type de métrique utilisé pour le routage externe.

Le champ *TOS* prend toujours la valeur 0.

Le champ *Métrique* indique le coût de la liaison entre le routeur ASBR qui annonce cette liaison et la destination qui se trouve en dehors du domaine OSPF considéré

Le champ *Adresse d'envoi* indique l'adresse du routeur vers laquelle les paquets de données devront être envoyés lorsqu'ils auront la destination annoncée par ce LSA comme propre destination.

Le champ *Tag de route externe* n'est pas à proprement parlé utilisé par OSPF. Son but est le transport d'information de routage entre les protocoles s'exécutant aux limites du domaine OSPF.

Il est à remarquer qu'une liaison externe peut aussi servir à décrire une route par défaut. On l'utilise quand aucune route spécifique n'existe pour cette destination. Dans ce cas, l'identifiant de l'état de liaison vaut 0.0.0.0 ainsi que la valeur du masque de sous-réseau.

Chapitre 2 : Ospf Multi Path

1. Introduction

L'approche classique du routage défini dans OSPF permet de découvrir le meilleur chemin existant entre une source et une destination. Cette approche se base uniquement sur le coût affecté aux liaisons. Malheureusement, dans la pratique, d'autres paramètres influencent de manière importante le comportement d'un domaine. C'est le cas du trafic. Son comportement est totalement aléatoire et très variable ; une liaison dans l'Internet peut en quelques minutes se retrouver saturée suite à une annonce diffusée dans les médias.

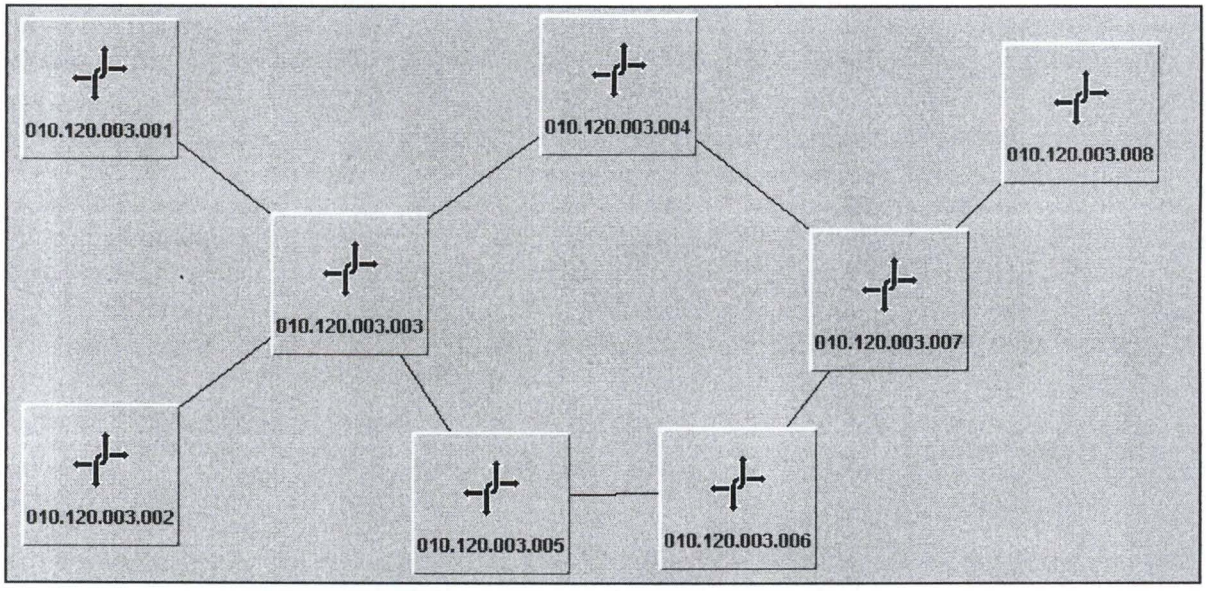


Figure 16 : Exemple de domaine OSPF

Prenons l'exemple du domaine représenté dans la figure 16 en admettant que toutes les liaisons ont un poids équivalent à 2. Le chemin entre les routeurs *010.120.003.001* et *010.120.003.008* passera respectivement par les routeurs *010.120.003.003*, *010.120.003.004* et *010.120.003.007*. Le chemin entre les routeurs *010.120.003.002* et *010.120.003.008* passera par les mêmes routeurs que ceux du premier chemin. En cas de forte charge du réseau, les liaisons entre *010.120.003.003* et *010.120.003.004* et entre *010.120.003.004* et *010.120.003.007* deviendront rapidement critiques. Elles seront à la base d'une congestion dans le domaine alors qu'il existe un autre chemin non saturé (en passant par le routeurs *010.120.003.005* et *010.120.003.006*) permettant de relier *010.120.003.001* et *010.120.003.002* à *010.120.003.008*. Cet autre chemin pourrait prendre en charge une partie du trafic de manière à mieux équilibrer la charge.

Actuellement les topologies de domaines offrent souvent plusieurs chemins de coût égaux pour relier deux points (sur la figure 16, il suffirait par exemple de mettre un poids égal à 0 pour la liaison unissant les routeurs *010.120.003.005* et *010.120.003.006*). Cela est dû d'une part, à la volonté d'avoir des chemins redondants offrant ainsi une plus grande sécurité d'utilisation et d'autre part, à l'accroissement constant du trafic qui nécessite l'ajout régulier de nouveaux circuits.

Partant de ce constat, et dans l'optique de réduire les congestions dans un domaine OSPF, il serait intéressant de distribuer la charge du trafic sur ces différents chemins.

La technique la plus souvent utilisée[16] pour gérer la répartition est connue sous le nom de *Equal Cost Multipath* (ECMP) [VIL99 p.2]. Celle-ci s'appuie sur une division égale du trafic entre les différents chemins de coûts égaux reliant deux points. Trois techniques ont été utilisées pour diviser le trafic de manière équitable :

1) en fonction de l'ordre d'arrivée. Ainsi, si 3 chemins pour une destination sont équivalents, les paquets 1,4,7,... seront envoyés sur le premier chemin ; les paquets 2,5,8 ... le seront sur le second chemin ; et les paquets 3,6,9,... sur le troisième chemin. Cette technique ne peut fonctionner que sur des chemins dont les délais de bout en bout sont plus ou moins égaux et stables. Dans le cas contraire, les performances au niveau TCP seront dramatiquement réduites ;
2) en fonction d'un préfixe basé sur l'adresse de destination du paquet. Cette technique offre des résultats peu prévisibles surtout au niveau des routeurs en périphérie du domaine car le trafic de ces derniers est souvent dirigé vers un seul préfixe ;
3) en fonction d'une technique de hachage basée d'une part sur l'adresse source et l'adresse destination du paquet et d'autre part sur le nombre de chemins équivalents pour la destination de ce paquet. Cette technique résout les problèmes des deux premières méthodes car le trafic entre une source et une destination prendra toujours le même chemin tandis que le trafic à destination d'un seul préfixe sera lui réparti de manière équitable.

ECMP fournit un effort pour répartir la charge dans le réseau, mais cela n'est pas suffisant. La répartition équitable telle que proposée n'est valable que localement. En effet, l'envoi d'une partie du trafic dans une direction donnée peut créer, en aval du point de décision, une congestion inattendue.
Il serait donc préférable d'avoir une répartition inéquitable du trafic sur les différents chemins possibles vus de manière locale dans le but d'optimiser le trafic global sur toutes les liaisons du domaine. Pour cela, les routeurs doivent être informés non seulement de la topologie du réseau mais également de la charge de celui-ci.
OMP (Optimized Multipath) propose une réponse à cette approche. Cette extension de OSPF fournit les moyens pour :

- informer les routeurs de la charge actuelle du domaine complet ;
- calculer de manière autonome la répartition des paquets sur plusieurs chemins afin d'optimiser la charge complète du réseau rapidement tout en évitant les effets d'oscillation.

OMP ne sera abordé ici que dans le contexte d'une seule zone. Les relations entre zones ne seront donc pas traitées.

2. Transmettre la charge du réseau

En résumé, on peut dire que la charge du réseau est évaluée localement au niveau de chacun des routeurs en se basant sur certaines informations fournies par la MIB II de SNMP. Cette information, synthétisée, est alors envoyée par inondation dans le réseau en utilisant des *Opaque LSA*.

a. *Information sur la charge locale du réseau*

Au sein de chaque routeur et pour chacune de ses interfaces, on mesure régulièrement (toutes les 15 secondes) grâce aux variables de SNMP [MKR91] :

- le nombre d'octets reçus et le nombre d'octets transmis (ifInOctets, ifOutOctets) ;
- le nombre de paquets reçus et le nombre de paquets transmis (ifInPacket, ifOutpacket) ;
- le nombre de paquets entrants rejetés et le nombre de paquets sortants rejetés (ifInDiscard, ifOutDiscard).

[16] Les premières techniques de prise en compte de la charge du réseau pour le routage utilisaient des méthodes de modifications de coût des lignes en fonction de leur charge respective. De telles méthodes d'ajustement entraînent des phénomènes d'oscillations dans la répartition de la charge.

Si on considère en simplifiant que la capacité de la ligne est une donnée constante (ifInSpeed et ifOutSpeed), on peut alors déterminer sans problème pour chaque direction, la charge de la ligne mesurée comme une fraction de sa capacité et, la proportion de paquets rejetés comme une fraction du nombre de paquets reçus.

Partant de là, on peut calculer pour chaque ligne une valeur particulière : sa charge équivalente. Cette valeur est égale à la charge actuelle de la ligne multipliée par un facteur. Elle fournit une estimation des pertes à partir desquelles TCP doit ralentir pour éviter une congestion. La charge équivalente ne doit en aucun cas être considérée comme une prédiction de la charge; elle sert plutôt à comparer différentes liaisons pour déterminer celle qui est la plus chargée en y incluant les effets des pertes. Si celles-ci sont nulles ou faibles, la valeur du facteur vaudra 1 (aucune estimation ne peut alors être faite sur l'évolution de la vitesse du trafic TCP en amont ou en aval). Si elles sont élevées et si la charge de la ligne avoisine les 100 %, alors, la valeur du facteur sera supérieure à un. Ce calcul est basé sur la constatation suivante : la charge utile véhiculée par TCP diminue de manière brutale en proportion à l'inverse de la racine carrée des pertes[17]. De plus, on estime généralement que TCP n'accélérera pas sensiblement si les pertes sont de l'ordre de 0,5 %. Si elles sont supérieures, alors, les performances de TCP se réduiront.

La valeur de la charge équivalente sera dès lors établie en fonction du tableau suivant :

Perte	Calcul	Commentaire
P <0.005	Equiv-load = load	Load = nombre d'octets en 1 sec/capacité de la ligne
0.005≤P≤0.09	Equiv-load = load * K * $\sqrt{P}$	K est un facteur fixé par défaut à 10
P > 0.09	Equiv-load = load * 3	Passée 9 %, la compensation reste fixe

Les différentes valeurs décrivant l'état de charge d'une ligne doivent être communiquées aux autres routeurs du domaine de manière à ce que tout le monde possède une image de la charge réelle du réseau. L'information est véhiculée grâce aux *opaque LSA*.

b. Opaque LSA

Le *opaque LSA*[VIL99 p.19-21] [COL98] est un nouveau type de LSA dont le but premier est de fournir un support aux extensions futures de OSPF. L'information contenue dans le *opaque LSA* peut être directement utilisée par OSPF ou indirectement par certaines applications qui veulent distribuer de l'information partout dans le domaine OSPF.
Sa structure comprend l'en-tête classique des LSA[18] suivie d'un champ de 32 bits contenant des informations spécifiques pour l'application utilisatrice, elle-même suivie par l'information proprement dite dont la taille peut être variable. Comme tout les autres LSA, le *opaque LSA* utilise les mécanismes classiques d'inondation de OSPF. On distingue 3 types de *opaque LSA* (pour des valeurs 9, 10 ou 11 du champ LSA type) suivant la partie du domaine qu'ils peuvent inonder.

---

[17] Cette affirmation se base sur l'équation suivante : $BP < \left(\frac{MSS}{RTT}\right) * \frac{1}{\sqrt{p}}$ dans laquelle :

- BP est le débit utile d'une connexion TCP ;
- MSS est la taille maximale d'un TPDU TCP ;
- RTT est le temps aller-retour entre l'émetteur et le destinataire ;
- P est le taux de perte (rapport entre les TPDU perdus et les TPDU effectivement transmis).

[MSM97]

[18] Pour être complet, il faut mentionner que le champ LinkStateID de l'en-tête est divisé en deux parties :

- *Opaque type* (8 premiers bits) permet de déterminer l'application utilisatrice du *opaque LSA* ;
- *type-specific ID* (24 bits restants) est une information dépendant du type de *opaque LSA*.

OMP utilise des *opaque LSA* de type 10. Dans ce contexte, le champ *Opaque Type* indique le type de liaison considérée et le champ *Opaque ID* contient un descripteur unique pour l'interface et le routeur émetteur.
Il faut noter qu'un *opaque LSA* ne décrit qu'une seule liaison. On aura donc autant de *opaque LSA* pour un routeur que celui-ci ne possède d'interfaces.

Le contenu du *opaque LSA* pour OMP est décrit dans la figure 17.

Version	Référence Type
Packing Method	BW scale
Reference LSA	
In Lin Capacity	
In Link Loading Fraction	
In Link Drop Fraction	
Out Link Capacity	
Out Link Loading Fraction	
Out Link Drop Fraction	

Figure 17 : format d'un *opaque LSA* pour OMP

Ce qui suit décrit les différents champs composant le *opaque LSA* de OMP.

Le champ *Version* prend la valeur 1.
Le champ *Reference Type* indique le type de LSA (type 1 - 5) qu'on référence dans le champ *Reference LSA*.
Le champ *Packing method* sert à indiquer le format de l'information véhiculée. La seule valeur possible actuellement est 1.
Le champ *BW scale* indique l'échelle de référence pour les champs *In Link Capacity* et *Out Link Capacity*.
Le champ *Référence LSA* reprend la valeur du champ *Link State ID* du LSA dans lequel la liaison est également décrite. Si la liaison est de type point-à-point, alors, on ajoute également dans ce champ un identifiant pour cette liaison.
Les champs *In Link Capacity* et *Out Link Capacity* donnent la capacité de la liaison en kiloBits par seconde.
Les champs *In Link Loading* et *Out Link Loading* donnent la charge effectivement utilisée sous forme fractionnaire.
Les champs *In Link Drop* et *Out Link Drop* donnent la proportion de paquets perdus.

c. *Envoi des Opaques LSA*

Régulièrement, les opaque LSA sont diffusés dans le réseau de la même manière que les LSA *classiques*. La fréquence de diffusion des opaque LSA est fonction de :

- la valeur actuelle de la charge équivalente de la ligne considérée ;
- la différence entre la charge équivalente du dernier envoi et la charge équivalente courante ;
- l'heure de l'envoi de la dernière description de cette liaison.

L'adaptation de la fréquence de diffusion permet de poursuivre un double objectif :

1. informer rapidement les membres du domaine de tout changement dans la charge pour arriver à une convergence plus rapide ;

2. tout en évitant une surcharge du domaine en messages de service lorsque ce dernier est dans un état stable.

3. Next Hop Structure

La *next Hop Structure* est présente dans tous les routeurs du domaine. Elle reprend, pour chaque destination (routeur ou réseau), un ensemble de chemins complets (on retient l'ensemble des tronçons à parcourir pour joindre une destination) dont certains peuvent avoir des parties en commun. Cette structure contient également l'information nécessaire pour répartir de manière optimale la charge vers une destination donnée. Il est important de noter qu'on retrouvera au sein d'un routeur une *next hop structure* pour chacune des destinations.

a. Abandonner le critère du meilleur chemin

Jusqu'ici, nous n'avons pris en compte que le meilleur chemin (en terme de coût) pour joindre deux points. Ce critère est beaucoup trop strict car il privilégie un sous-ensemble de liaisons dans le domaine (liaisons qui seront par ailleurs plus rapidement saturées) et délaisse tout une série de liaisons plus coûteuses mais peu ou pas chargées.

OMP propose un élargissement de cette règle : Admettre un plus grand nombre de chemins sachant que nombre d'entre eux seront plus coûteux que le meilleur, tout en évitant la création de boucle. Les boucles pourront être évitées en se basant sur le constat suivant : « Tout nœud adjacent qui est plus proche de la destination en terme de coût que le nœud courant et qui ne traverse pas un lien virtuel peut être considéré comme un nœud adjacent viable dans la détermination du chemin vers la destination »[19] [VIL99 p. 8].
L'utilisation de cette règle se limitera uniquement aux nœuds suivant directement le nœud courant car :

- il n'existe pas de moyens pour savoir si les nœuds en aval appliquent cette même règle ;
- si les nœuds suivants appliquent la même règle, alors l'ensemble des meilleurs chemins existe aussi dans leur *next hop structure*. OMP est comme OSPF un algorithme distribué où chaque nœud participe à l'élaboration d'un routage optimal.

b. Construire les chemins de la Next Hop Structure

La construction de cette structure [VIL99 p. 24] se base sur l'algorithme de Dijkstra comme dans le cas de OSPF auquel on apporte quelques modifications :

- premièrement, il est nécessaire de retenir le chemin complet entre la source et la destination (ce qui n'est pas nécessaire pour l'implémentation pure de OSPF) ;
- deuxièmement, il faut évaluer le coût des meilleurs chemins vers toutes les destinations à partir de tous les nœuds directement adjacents au nœud source[20]. Tous les chemins découverts dans les nœuds adjacents dont le coût est strictement inférieur au coût du meilleur chemin entre la source et la destination sont inclus dans la liste des chemins pour une destination donnée.

c. Le segment critique

Pour chaque ensemble de chemins concernant une destination particulière, un tronçon (pouvant contenir un ou plusieurs liens) est identifié comme étant le segment critique. Le segment critique est une partie d'un chemin ayant la valeur de charge équivalente la plus élevée. Cette valeur est calculable localement pour chacune des liaisons dans le domaine de par la diffusion des *Opaque LSA*.

[19] Si le coût au niveau du nœud suivant est égal ou supérieur, alors on peut assister à la formation éventuelle de boucles.
[20] Il est préférable de calculer les meilleurs chemins du point de vue de chacun des nœuds adjacents plutôt que d'implémenter un algorithme généralisé [VIL99 p. 31].

Une *next Hop Structure* ne peut contenir qu'un seul segment critique[21]. Ce segment peut être partagé par plusieurs chemins.

Dans un soucis d'optimisation, lorsque tous les chemins d'une *next Hop Structure* passent par une même liaison, il est possible de ne considérer, comme candidates à devenir segment critique, que les liaisons entre l'émetteur et la liaison commune à tous.

4. Ajuster le coût des chemins

La *next hop structure* décrite précédemment nous donne un ensemble de chemins complets et corrects vers une destination donnée pour un émetteur donné. Dans cet ensemble, on désigne un tronçon particulier comme étant le segment critique [VIL99 p. 11], c'est à dire celui qui a la charge équivalente la plus importante.
Comme précisé dans l'introduction, OMP va régulièrement adapter la proportion de trafic qui est envoyé sur chacun des chemins vers une destination en fonction de la charge globale du réseau. L'ajustement des proportions entre les différents chemins est réalisé lorsque [VIL99 p. 12] :

- soit, on reçoit un opaque LSA donnant des informations sur le segment critique de la *next hop structure* considérée ;
- soit, un critère liant les trois éléments suivants est réalisé :
    - la différence de charge équivalente entre le segment le plus chargé (segment critique) et le moins chargé ;
    - la valeur de la charge équivalente du segment critique ;
    - l'heure du dernier ajustement pour ces chemins.

a. *Calcul de l'ajustement*

Le calcul de l'ajustement [VIL99 p. 13-14] repose sur le principe suivant : la charge des chemins qui contiennent le segment critique doit être déplacée vers les chemins qui ne contiennent pas ce segment. Pour assurer une stabilité à l'algorithme et éviter des phénomènes d'oscillations, le taux d'ajustement doit rester limité tout en permettant cependant des adaptations relativement rapides. Ainsi, au fur et à mesure de l'évolution de la charge du réseau, ce taux sera revu à la hausse ou à la baisse.

Chaque chemin dans une *next hop structure* contient les variables suivantes :

- *traffic share* : la proportion du trafic que ce chemin reçoit (au départ, cette proportion est répartie équitablement entre tous) ;
- *move increment* : la valeur de l'ajustement, c'est à dire l'augmentation autorisée de trafic pour ce chemin lorsqu'on veut en déplacer vers lui. La valeur de l'ajustement augmente à chaque fois que ce chemin reçoit du trafic supplémentaire[22]. Cette valeur est généralement initialisée à 1 % du trafic global pour une destination donnée ;
- *move count* : le nombre de fois que ce chemin a vu sa proportion de trafic augmenter de manière consécutive. La valeur est initialisée à 0.

L'algorithme de modification des proportions se déroule alors en trois temps :

1) ajustement du *move increment* de chacun des chemins :

[21] Il faut remarquer que l'auteur ne mentionne pas le fait qu'on puisse avoir deux liaisons séparées ayant la même charge équivalente (par exemple, deux liaisons dont la charge réelle est maximale et dont le taux de perte dépasse 9 %). Dans le reste de ce travail, nous dérogerons à la règle d'unicité dans le cas de tronçons étant réputés critiques et ayant une charge équivalente égale. Une étude plus poussée dans ce domaine (hors du cadre de ce mémoire) permettrait probablement de reconnaître parmi des segments critiques identiques (c'est à dire ayant le même coût équivalent) celui qui a la plus grande influence sur la surcharge du réseau du point de vue du routeur local.

[22] L'accroissement à un comportement exponentiel ce qui permet une stabilisation assez rapide.

- si le chemin contient le segment critique, alors, le *move increment* reste inchangé ;
- si le chemin ne contient pas le segment critique mais qu'il contenait le précédent (c'est à dire qu'il possédait le segment critique avant que celui-ci ne soit modifié), alors, son *move increment* est remplacé par la valeur de la moitié du plus petit *move increment* des chemins contenant le segment critique. Cela permet de commencer avec de petites valeurs pour garantir une certaine stabilité ; ce chemin se trouvant à ce moment plus ou moins sur la frontière entre la possession et la non-possession du segment critique ;
- si le chemin ne contient pas le segment critique et ne le contenait pas au tour précédent, alors, son *move increment* est augmenté. L'augmentation est fonction du *move count* ce qui permet de donner à l'accroissement un caractère exponentiel.

2) si le chemin ne contient pas le segment critique, alors la valeur de son *traffic share* est augmentée de la valeur de son *move increment*. Par contre, tous les chemins possédant le segment critique voient leur *traffic share* diminué d'une valeur X. Cette valeur X est la même pour tous les chemins critiques et correspond au total de l'accroissement de trafic des chemins non critiques divisés par le nombre de chemins critiques. Ainsi, tous les chemins non-critiques vont prendre en charge une partie du surplus du ou des chemins critiques entraînant pour ceux-ci une diminution plus ou moins rapide de leur excédent.

3) lorsque tous les chemins ont leur nouvelle valeur pour *traffic share*, la proportion pour chacun des chemins qui partagent le même nœud suivant est regroupée et toute l'information est alors communiquée à la partie responsable du routage pour le routeur.

5. Adaptations aux modifications topologiques

Dans un domaine OSPF, des liaisons peuvent à tout moment apparaître ou disparaître [VIL99 p. 15-16]. Ces liaisons entraîneront une adaptation des chemins possibles dans le domaine brisant de ce fait l'équilibre dans la répartition de la charge. Un nouvel équilibre devra dès lors être trouvé. Cette recherche partira de l'état d'équilibre précédent de manière à ne pas provoquer de changements trop brusques.

Lorsqu'un chemin est perdu, la proportion du trafic qui lui était allouée est distribuée à tous les chemins restants proportionnellement à leur allocation antérieure. Si ce chemin était fort chargé, une adaptation sera alors très probablement enclenchée.

Lorsqu'une nouvelle liaison apparaît et que celle-ci entraîne la création d'un nouveau chemin possible pour une destination donnée, alors, ce chemin reçoit une proportion de la charge égale à 0. Au fur et à mesure des adaptations, sa proportion augmentera jusqu'à atteindre son point d'équilibre. Le fait de commencer avec une valeur nulle permet de limiter l'influence que peut avoir une liaison avec un comportement anormal (court délai entre son apparition et sa disparition) sur la répartition de la charge.

6. Exemple

Ce qui suit sert à montrer l'efficacité et l'utilité d'une répartition du trafic par OMP en comparaison avec une répartition de celui-ci suivant ECMP. Les représentations, les tableaux et les résultats ont été obtenus avec le simulateur OSPF, sujet de ce travail.

La figure 18 donne la représentation schématique du domaine.
La figure 19 est un tableau dans lequel on peut lire le poids associé à chacune des liaisons du domaine.
La figure 20 donne sous forme d'un tableau le trafic généré dans le domaine.

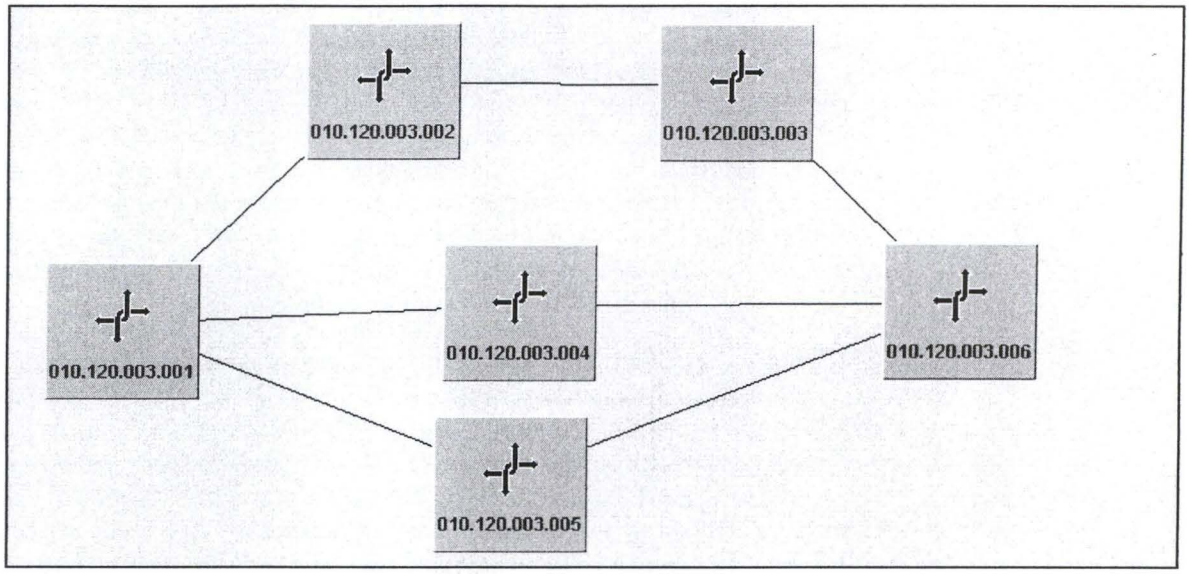


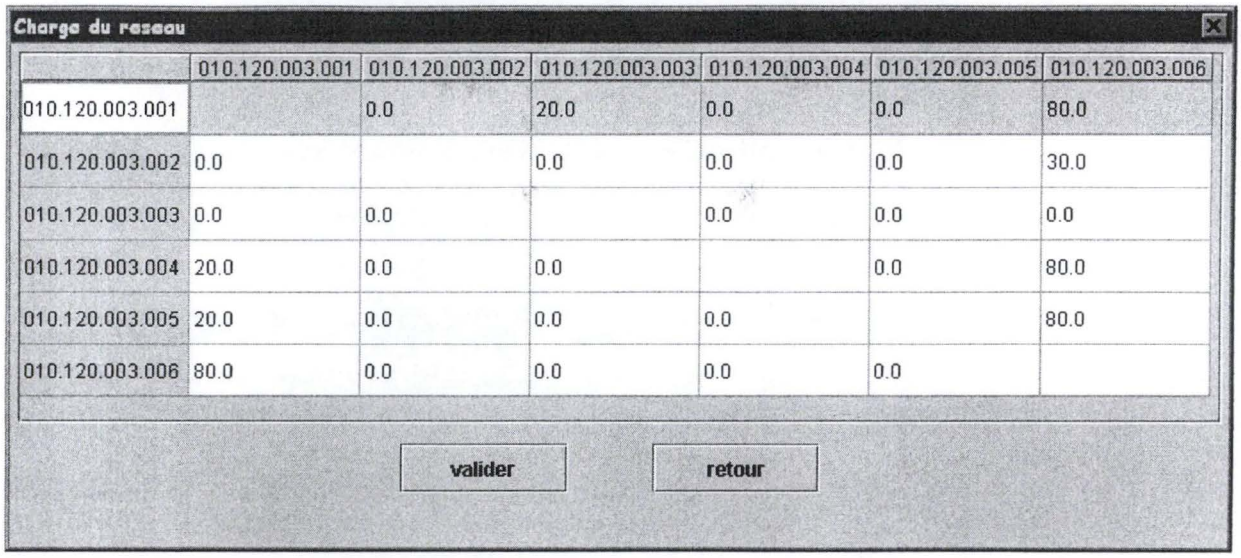
Figure 18 : Représentation du domaine

Tableau des couts de liaison

	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		5		5	5	
010.120.003.002	5		2			
010.120.003.003		2				5
010.120.003.004	5					5
010.120.003.005	5					5
010.120.003.006			5	5	5	

valider retour

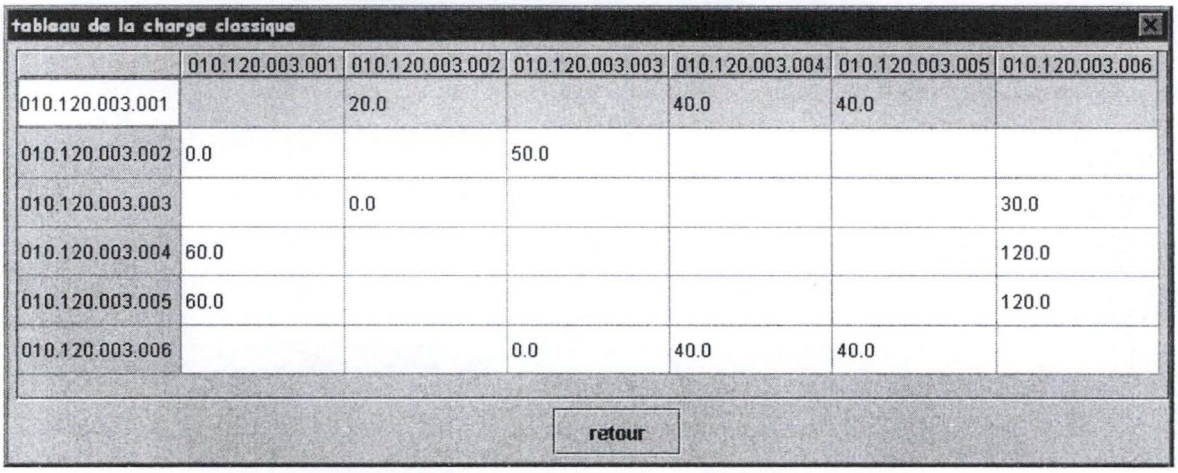
Figure 19 : coût des liaisons



	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		0.0	20.0	0.0	0.0	80.0
010.120.003.002	0.0		0.0	0.0	0.0	30.0
010.120.003.003	0.0	0.0		0.0	0.0	0.0
010.120.003.004	20.0	0.0	0.0		0.0	80.0
010.120.003.005	20.0	0.0	0.0	0.0		80.0
010.120.003.006	80.0	0.0	0.0	0.0	0.0	

Figure 20 : trafic dans le réseau

La figure 21 donne dans un tableau la charge des différentes liaisons lorsqu'on applique une répartition suivant ECMP. On peut remarquer que les liaisons entre *010.120.003.004* et *010.120.003.006* et entre *010.120.003.005* et *010.120.003.006* sont surchargées. Cette surcharge est causée par le fait que le routeur *010.120.003.001* envoie son trafic vers le routeur *010.120.003.006* uniquement par les routeurs *010.120.003.004* et *010.120.003.005*, ignorant ainsi complètement le chemin du dessus. On remarquera aussi que le trafic entre *010.120.003.006* et *010.120.003.001* n'exploite pas non plus le chemin du dessus.



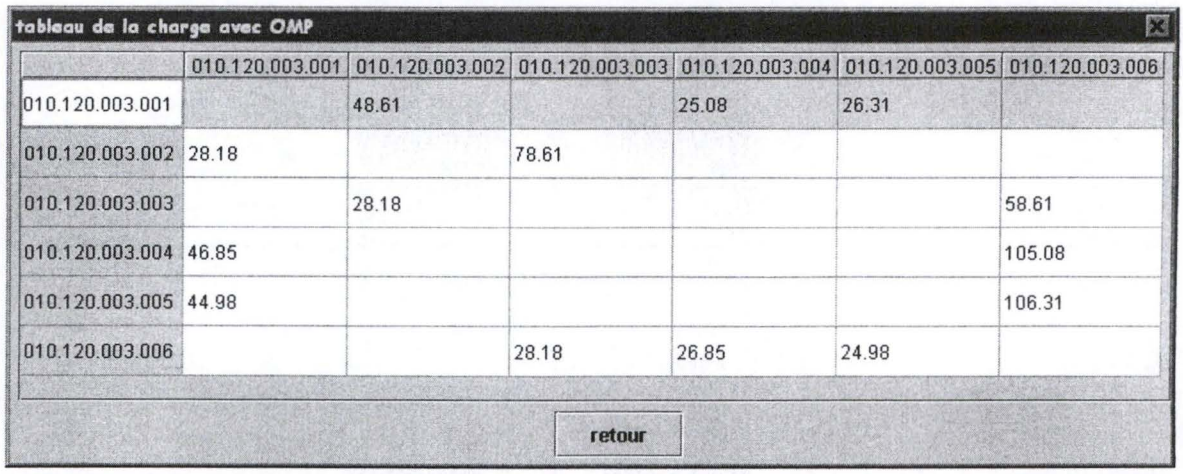
	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		20.0		40.0	40.0	
010.120.003.002	0.0		50.0			
010.120.003.003		0.0				30.0
010.120.003.004	60.0					120.0
010.120.003.005	60.0					120.0
010.120.003.006			0.0	40.0	40.0	

Figure 21 : répartition de la charge suivant ECMP

Les figures 22, 23, 24 et 25 donnent l'évolution de la répartition de la charge lorsque celle-ci est établie en tenant compte de OMP. La figure 22 est la charge obtenue au début de l'application de OMP. On peut remarquer qu'environ 1/3 du trafic entre *010.120.003.001* et *010.120.003.006* passe par le chemin du dessus ce qui permet de décharger quelque peu les deux autres chemins. Cette constatation est aussi vraie pour le trafic entre *010.120.003.006* et *010.120.003.001*. Les figures 23, 24 et 25 sont des vues de la charge prises à 10 minutes, 20 minutes et 40 minutes après le démarrage du domaine. On peut constater qu'une partie du trafic entre *010.120.003.001* et *010.120.003.006* s'est déplacé vers le chemin du dessus qui prend après 15 minutes 65 % de ce trafic et se stabilise autour de cette valeur. La charge s'est équilibrée et on n'a plus de pertes. Par contre, le trafic entre *010.120.003.006* et *010.120.003.001* est resté

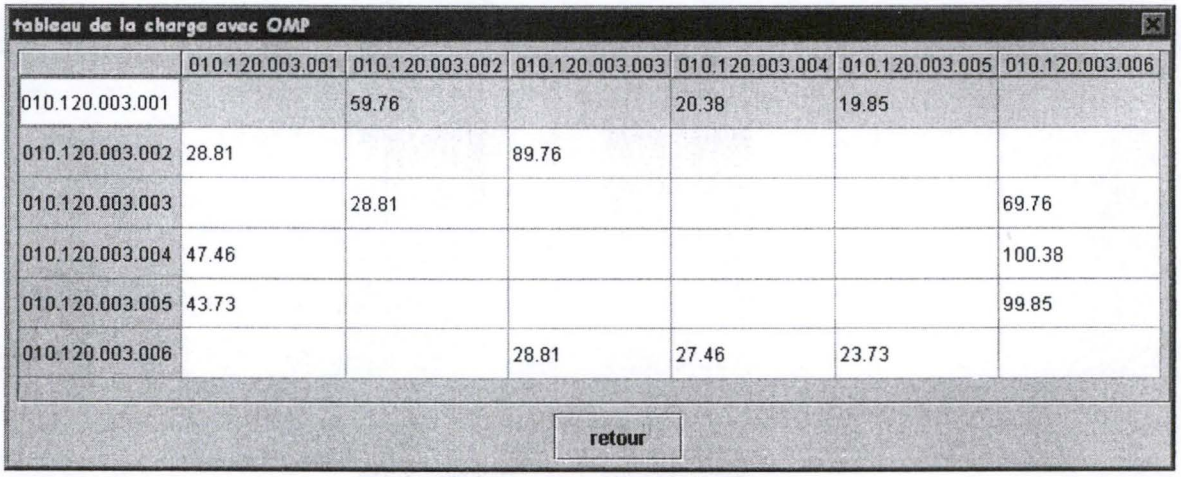
plus ou moins dans une répartition équitable et évolue très lentement. Cela est dû au fait que les liaisons impliquées dans ce trafic ne sont jamais surchargées.

OMP ne fournira jamais de résultats tranchés et oscillera toujours légèrement autour de la valeur moyenne idéale. Cela est dû au fait que tous les chemins entre une source et une destination peuvent avoir un segment critique et que le chemin possédant le segment critique perd une proportion de son trafic. Mais comme, dans une situation d'équilibre, tous les chemins posséderont régulièrement le segment critique, les oscillations resteront centrées autour de la moyenne idéale.



	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		48.61		25.08	26.31	
010.120.003.002	28.18		78.61			
010.120.003.003		28.18				58.61
010.120.003.004	46.85					105.08
010.120.003.005	44.98					106.31
010.120.003.006			28.18	26.85	24.98	

Figure 22 : répartition de la charge du trafic sur les liaisons au début de l'application de OMP



	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		59.76		20.38	19.85	
010.120.003.002	28.81		89.76			
010.120.003.003		28.81				69.76
010.120.003.004	47.46					100.38
010.120.003.005	43.73					99.85
010.120.003.006			28.81	27.46	23.73	

Figure 23 : répartition de la charge du trafic sur les liaisons 10 minutes après l'application de OMP

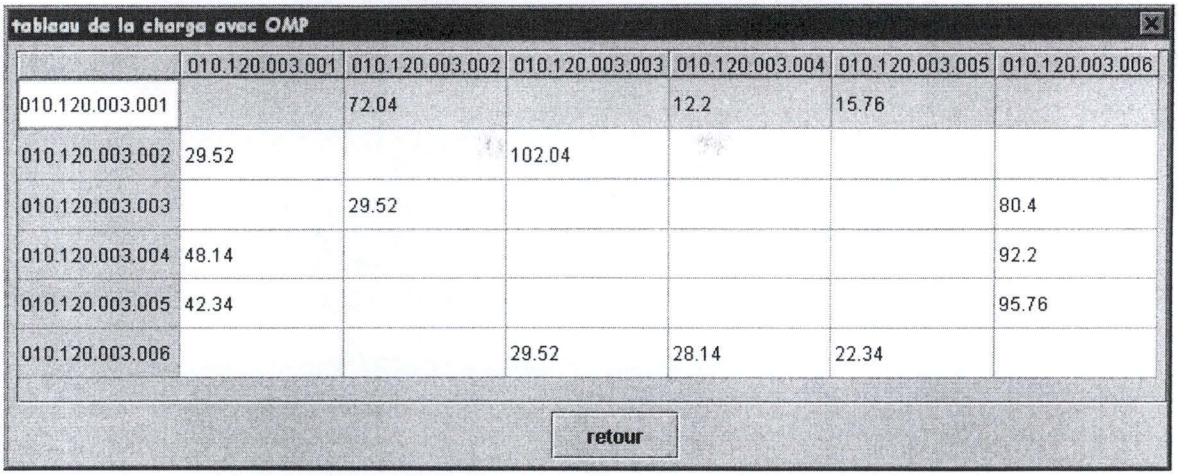


tableau de la charge avec OMP

	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		72.04		12.2	15.76	
010.120.003.002	29.52		102.04			
010.120.003.003		29.52				80.4
010.120.003.004	48.14					92.2
010.120.003.005	42.34					95.76
010.120.003.006			29.52	28.14	22.34	

retour

Figure 24 : répartition de la charge du trafic sur les liaisons 20 minutes après l'application de OMP

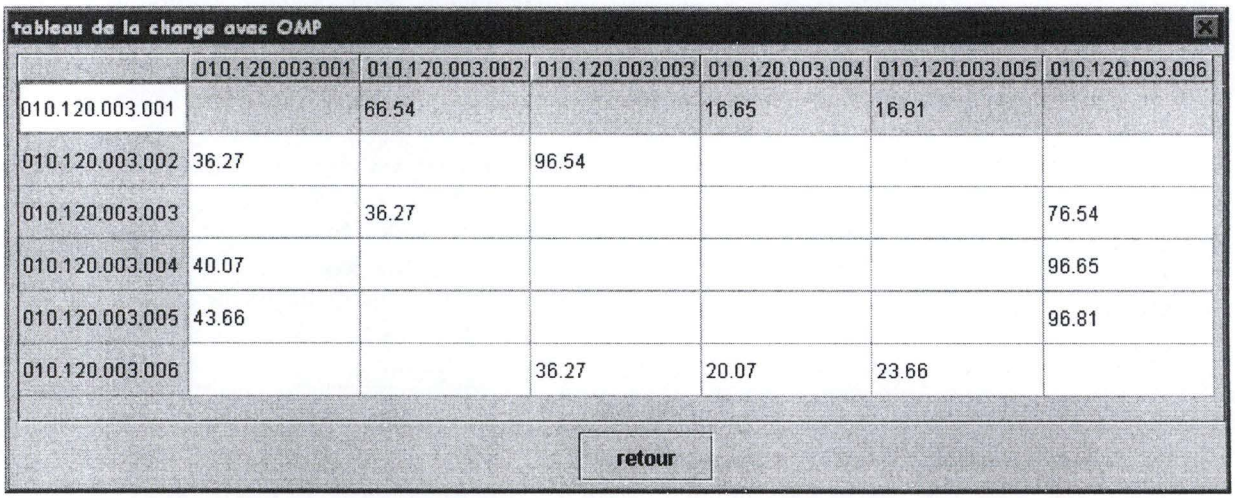


tableau de la charge avec OMP

	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006
010.120.003.001		66.54		16.65	16.81	
010.120.003.002	36.27		96.54			
010.120.003.003		36.27				76.54
010.120.003.004	40.07					96.65
010.120.003.005	43.66					96.81
010.120.003.006			36.27	20.07	23.66	

retour

Figure 25 : répartition de la charge du trafic sur les liaisons 40 minutes après l'application de OMP

Chapitre 3 : Architecture du simulateur OSPF

1. Spécification du simulateur OSPF

OSPF peut être divisé en deux parties. D'une part, on retrouve les éléments qui sont indispensables au fonctionnement du protocole et d'autre part, les éléments qui ne font qu'en augmenter les performances.

Dans un premier temps, nous allons nous baser sur la description théorique du protocole pour déterminer les fonctionnalités qui sont indispensables et celles qui ne sont que des améliorations. Dans cette démarche, il est important de souligner que le critère de performance n'est pas retenu. Nous considérerons également que le réseau est totalement fiable (il n'y a pas de pertes de paquets) et que les délais de transmission sont nuls.
Dans un deuxième temps, en se basant sur le premier point, nous décrirons le *cahier des charges* du simulateur.

a. *Déterminer l'essentiel*

OSPF est un protocole à état de liaison. Chaque routeur doit pouvoir décrire et diffuser sa configuration locale à travers tout le réseau à l'aide de LSA. Les éléments essentiels de l'en-tête sont :

- les informations qui permettent de distinguer deux LSA. Il s'agit des champs *type*, *ID d'état de liaison* et *routeur annonçant*;
- l'information qui permet de distinguer deux instances d'un même LSA à savoir le champ *numéro de séquence*. Cette information est nécessaire car elle permet de mettre la base de données à jour suite à l'application régulière du protocole d'inondation ;
- l'information qui permet de donner l'âge d'un LSA, à savoir le champ *âge*. Ce champ permet de décider le moment auquel une information doit être rafraîchie et le moment où elle doit être supprimée. Le rafraîchissement et la suppression sont essentiels pour conserver une synchronisation des différentes bases de données des routeurs. Sans cela, la disparition dans le réseau d'un routeur ne serait jamais perçue (du moins directement par le protocole), les enregistrements le concernant n'étant alors jamais supprimés.

Les autres champs ont trait à la fiabilité des moyens de transmission et aux extensions de OPSF. Ils ne sont donc, dans ce contexte, pas nécessaire.

Nous ne considérerons que des liaisons point-à-point entre tous les routeurs du domaine. En effet, les techniques expliquées dans le premier chapitre concernant la gestion des routeurs dans les réseaux avec ou sans diffusion ne sont que des optimisations destinées à limiter l'échange des messages de configuration entre les différents routeurs. Tous ces réseaux peuvent sans aucun problème être représentés par des liaisons point-à-point entre tous les routeurs participants. De la même manière, la division d'un domaine en différentes zones n'est qu'une optimisation dans le but de diminuer la taille de la base de données en n'y mettant que les informations considérées comme nécessaire (au détriment d'ailleurs d'une utilisation optimale). Nous considérerons également que le domaine OSPF est unique et qu'il ne doit pas communiquer avec d'autres systèmes autonomes. Toutes ses limitations nous amènent à n'utiliser que le LSA de type 1. Ce LSA décrit la configuration locale d'un routeur dans un réseau point-à-point. Dans ce LSA, nous retiendrons comme information nécessaire le nombre de liaisons qu'il contient et pour chacune d'elles, nous utiliserons comme élément descriptif :

- le champ *Identifiant de liaison* qui permet de connaître le routeur voisin du routeur émetteur ; les deux routeurs étant unis par la liaison décrite ;
- le champ *métrique* qui fournit le coût de la liaison décrite.

Les autres champs ont trait à des optimisations de OPSF ou contiennent des informations redondantes par rapport à ce que nous avons besoin ici. Ils ne sont donc pas nécessaire.

Chaque routeur doit disposer d'une base de données qui représente la topologie complète du domaine. Cette base de données contiendra la version la plus récente de toutes les informations topologiques envoyées par les autres routeurs du réseau ainsi que sa configuration locale.

Sur base de cette description de la topologie, le routeur doit pouvoir déterminer l'interface de sortie (comme il n'existe que des liaisons point-à-point, il suffit de donner le routeur suivant sur le chemin vers la destination) pour chacune des destinations dans le réseau. Cela revient à appliquer un algorithme de recherche du chemin le plus court à travers un graphe orienté donné par la base de données d'état.

Il est également nécessaire que les routeurs puissent s'échanger des messages de configuration. Cet échange est assuré par les sous-protocoles de OSPF. Dans ce travail, nous ne retiendrons que le protocole d'inondation et une sous-partie du protocole *hello*. Ce dernier sera utilisé par un routeur pour vérifier si un voisin situé à l'autre bout d'une liaison non encore utilisée pour le transfert des données est actif ou non[23] ; par extension, on l'utilisera aussi pour voir si une liaison est effectivement bi-directionnelle.
Le but du protocole d'échange des bases de données étant de rendre un nouveau routeur plus rapidement opérationnel dans un domaine OSPF (il reçoit une copie d'une base de données synchronisée), il ne sera pas utilisé; d'autant plus que le protocole d'inondation atteint le même but lors du rafraîchissement régulier des bases de données, mais, moins rapidement.

Les éléments essentiels de l'en-tête des paquets OSPF sont :

- le champ *Type* ;
- le champ *Identifiant du routeur OSPF source.*

Les autres champs concernent des optimisations qui ne sont pas prises en compte dans ce travail.

Le corps du paquet OSPF d'inondation contiendra :

- le champ *Nombre d'état de liaison* ;
- le contenu du LSA transporté par ce paquet.

Le corps du paquet OSPF *hello* contiendra le champ *neighbour* qui donne la liste des voisins connus par le routeur émetteur.

Il est à noter que la procédure d'acquittement prévue par le protocole d'inondation ne sera pas appliquée ici car le réseau est totalement fiable.

En partant des éléments que nous venons de dégager, nous pouvons donner les spécifications décrivant les fonctionnalités que le simulateur OSPF doit respecter.

b. Spécifications des fonctionnalités

Les spécifications du simulateur sont basées sur les normes édictées dans le protocole OSPF duquel on a retiré les éléments non vitaux.

Le simulateur doit pouvoir montrer l'évolution d'un domaine OSPF. Cette évolution se déroule en plusieurs phases :

[23] Il n'y aura donc pas de messages *hello* périodiques pour vérifier si une liaison existe toujours. En effet, on considère le réseau comme totalement fiable et toute suppression de liaisons doit être un acte explicite. De plus, l'envoi et la réception régulières de messages d'inondation permettront de voir l'état des voisins.

- la création de la topologie du domaine ;
- le démarrage du domaine ;
- l'application dynamique des protocoles de OSPF ;
- la terminaison du domaine.

Le simulateur doit pouvoir être volontairement et temporairement figé durant son exécution. Il doit également être possible d'en faire une exécution pas-à-pas. Par pas-à-pas, il faut entendre, au niveau de chaque routeur, le traitement d'un seul message en entrée au cours d'un cycle et le vieillissement d'une seconde des données de la base de données.

Le simulateur doit permettre la création et la suppression de routeurs avant le démarrage du domaine ainsi que pendant son déroulement dynamique. Un routeur peut être créé sans pour autant être démarré dans le domaine.

Le simulateur doit permettre de créer des liaisons point-à-point entre tous les routeurs du domaine. Une liaison ne peut être active et autoriser le transfert de données que si elle connecte deux routeurs existants et démarrés et que cette liaison est bidirectionnelle. Ce dernier aspect est découvert par un routeur uniquement en utilisant le protocole *hello*. On n'autorise qu'une seule liaison orientée entre deux routeurs. Un routeur ne peut pas avoir de liaison vers lui-même. Chaque liaison doit recevoir un poids de valeur positive ou nulle. Ce poids peut être consultable de manière globale pour toutes les liaisons et pourra être modifié. Toute liaison doit pouvoir être supprimée quelle que soit son état.

Tout routeur dans le domaine OSPF doit être identifié de manière unique par une adresse IP. Cette adresse ne peut être modifiée. Tout routeur doit pouvoir décrire son environnement local avec un LSA. Ce LSA reprendra :

- les champs qui permettent de l'identifier de manière unique ;
- le champ qui permet d'en distinguer deux instances ;
- le champ qui permet de donner son âge ;
- les champs qui permettent de décrire chacune des liaisons partant de ce routeur.

Tout LSA doit pouvoir être diffusé dans le domaine OSPF de manière régulière en veillant à différencier chacune des nouvelles instances.

Tout routeur doit disposer d'une base de données d'état qui lui est propre. Cette base de données devra contenir l'instance la plus récente du LSA local ainsi que la copie la plus récente de tous les autres LSA reçus par ce routeur. Le routeur doit pouvoir modifier l'âge des LSA de sa base de données et prendre des décisions de suppression en fonction de l'âge de chacun. Toute modification du coût d'une liaison ainsi que toute modification topologique doit entraîner la génération immédiate d'un nouvel LSA descriptif pour le routeur concerné.

Tout routeur démarré doit pouvoir à tout moment afficher le contenu de sa base de données d'état.

Un routeur recevant un LSA doit, si ce LSA répond aux critères de propagation (le LSA doit être strictement plus jeune en terme d'instance que le même LSA déjà présent dans la base de données locale), envoyer ce LSA à tous ses voisins directement connectés à l'exception de celui par lequel le message lui est parvenu. Si le critère n'est pas respecté, alors le LSA n'est pas propagé. Si le LSA reçu est plus vieux que celui présent dans la base de données, alors la version locale de ce LSA doit être envoyée au routeur émetteur.

Toute diffusion de LSA ne peut se faire que contenue dans un paquet OSPF. La durée de vie d'un paquet OSPF est limitée à la liaison directe entre deux routeurs ; lorsqu'il a atteint son destinataire direct, il est considéré comme mort. Chaque paquet OSPF contient les champs qui permettent d'identifier son origine ainsi que les LSA qu'il doit transporter. Les paquets OSPF sont également

utilisés pour le transport des informations du protocole *hello*. Tous les paquets OSPF circulant dans le réseau doivent pouvoir être conservés de manière à permettre leur analyse ultérieure.

Chaque routeur doit pouvoir, indépendamment des autres, calculer, pour chacune des destinations, le chemin le plus court pour pouvoir la rejoindre. Plusieurs chemins peuvent coexister s'ils ont le même coût minimal. Le calcul doit se baser sur l'algorithme de Dijsktra (recherche du meilleur chemin dans un graphe orienté). Les données topographiques doivent exclusivement être retirées de la base de données d'état du routeur.

Tout routeur X démarré doit pouvoir à tout moment afficher sa table de routage. Celle-ci doit indiquer, pour chaque routeur démarré dans le domaine s'il est possible de l'atteindre à partir de X et, dans ce cas, de donner le ou les identifiants du ou des premiers routeurs voisins de X sur le ou les chemins vers cette destination.

Le simulateur doit offrir une vision graphique du domaine OSPF. Chaque routeur doit pouvoir être déplacé et les liaisons entre routeurs doivent s'adapter à ces déplacements (principe du drag&drop), cela, dans le but d'adapter la représentation graphique d'un domaine aux desiderata de l'utilisateur. Le trafic circulant entre les routeurs via les différentes liaisons doit être représenté sur le schéma. Il faut également pouvoir distinguer les routeurs qui sont démarrés de ceux qui ne le sont pas. Ceci est aussi vrai pour les liaisons, entre celles qui sont activées et celles qui ne le sont pas (par exemple, il doit être possible de créer une liaison entre un routeur démarré et un routeur non démarré ; cette liaison n'étant alors représentée que pour des raisons de facilité ; en réalité, elle doit être ignorée par le routeur démarré).

2. Architecture du simulateur OSPF

Partant des spécifications données au point précédent, nous allons maintenant nous attacher à décrire les aspects principaux de l'architecture du simulateur OSPF. L'approche sera thématique. Cette description suppose que le lecteur possède déjà une connaissance du langage de programmation Java. Il ne sera pas traité des aspects graphiques du simulateur, celui-ci utilisant uniquement des éléments standards de la programmation graphique sous JAVA.

Le plan de description de l'architecture est le suivant :

- définition de la manière dont un routeur est perçu dans l'application, de la manière dont les routeurs communiquent entre eux et finalement description de la structure *domaine* qui regroupe tous les routeurs créés et utilisés dans l'application ;
- description de la structure d'un routeur et des méthodes qui permettent de le créer, de le démarrer et de l'arrêter ;
- description du protocole d'inondation qui est responsable de la diffusion des messages de configuration entre les différents routeurs et description de la structure de ces messages ;
- description des liaisons actives et inactives au sein d'un routeur, description du protocole *hello* qui permet d'activer les liaisons et finalement description de l'ajout, de la modification et de la suppression des liaisons ;
- description de la gestion du temps dans le simulateur principalement en ce qui concerne le champ *Age* des LSA et la génération des messages *hello* ;
- description et exemple de l'algorithme de Dijkstra.

a. *Relation entre les routeurs et structure du domaine*

Le domaine représente l'ensemble des routeurs qui interagissent entre eux. Nous allons, dans un premier temps, aborder la problématique des relations entre les routeurs puis nous donnerons la structure du domaine.

1) Définition et relation entre routeurs

Chaque routeur créé dans le cadre du simulateur est un processus autonome mais tous les routeurs dépendent du même programme. La classe *Router* est donc logiquement une extension de la classe Java *Thread*. Le comportement d'un routeur dans le simulateur est asynchrone par rapport aux autres routeurs. Cette approche a permis de modéliser le comportement du routeur de manière fort proche à la réalité.

Mais, qui parle de processus autonomes doit aborder la question des relations entre ces différents processus. Le simulateur n'échappe pas à cette règle étant donné qu'un routeur doit régulièrement échanger des messages de configuration avec ses voisins. Le fait qu'un processus puisse écrire des informations dans la structure de données d'un autre processus et que cet autre processus puisse les lire nous ramène à un cas typique de problème *producteur-consommateur* entre processus concurrents. La solution à ce problème consiste à créer une structure de données intermédiaire **critique** accessible par les deux processus mais de telle manière à ce qu'il ne puisse y avoir aucun conflits ni pertes de données. Plusieurs solutions ont été proposées pour résoudre ce problème. Java propose une technique basée sur les moniteurs. Chaque objet possède un verrou. Ce verrou est activé lorsqu'on applique une méthode déclarée comme *synchronized* sur cet objet. A partir du moment où cette méthode accède à l'objet, toute autre méthode issue d'un autre processus et voulant accéder à ce même objet se verra bloquée tant que la méthode qui accède à l'objet n'a pas libéré le verrou.
La structure d'échange d'informations entre routeurs est une instance de la classe *Buffer*, descendant direct de la classe *Linked List*. Le fait d'utiliser une structure dédiée uniquement à l'échange des messages permet de protéger les autres composants du routeur qui ne pourront et ne devront pas être accédés par un autre processus.
Cette structure implémente une file d'attente dont les seuls moyens d'accès sont quatre méthodes déclarées comme *synchronized* :

- *PushBuffer* qui permet d'ajouter un élément au bout de la file ;
- *PopBuffer* qui permet de retirer un élément en tête de la file ;
- *IsBufferEmpty* qui vérifie si la file en vide ;
- *ClearBuffer* qui vide la file d'attente.

La figure 26 donne un exemple des relations entre deux processus concurrents que l'on peut assimiler à deux routeurs. Elle montre la manière dont l'accès à une ressource critique est géré. Le premier cas montre les relations sans gestion de la concurrence et le second cas avec une gestion de la concurrence.

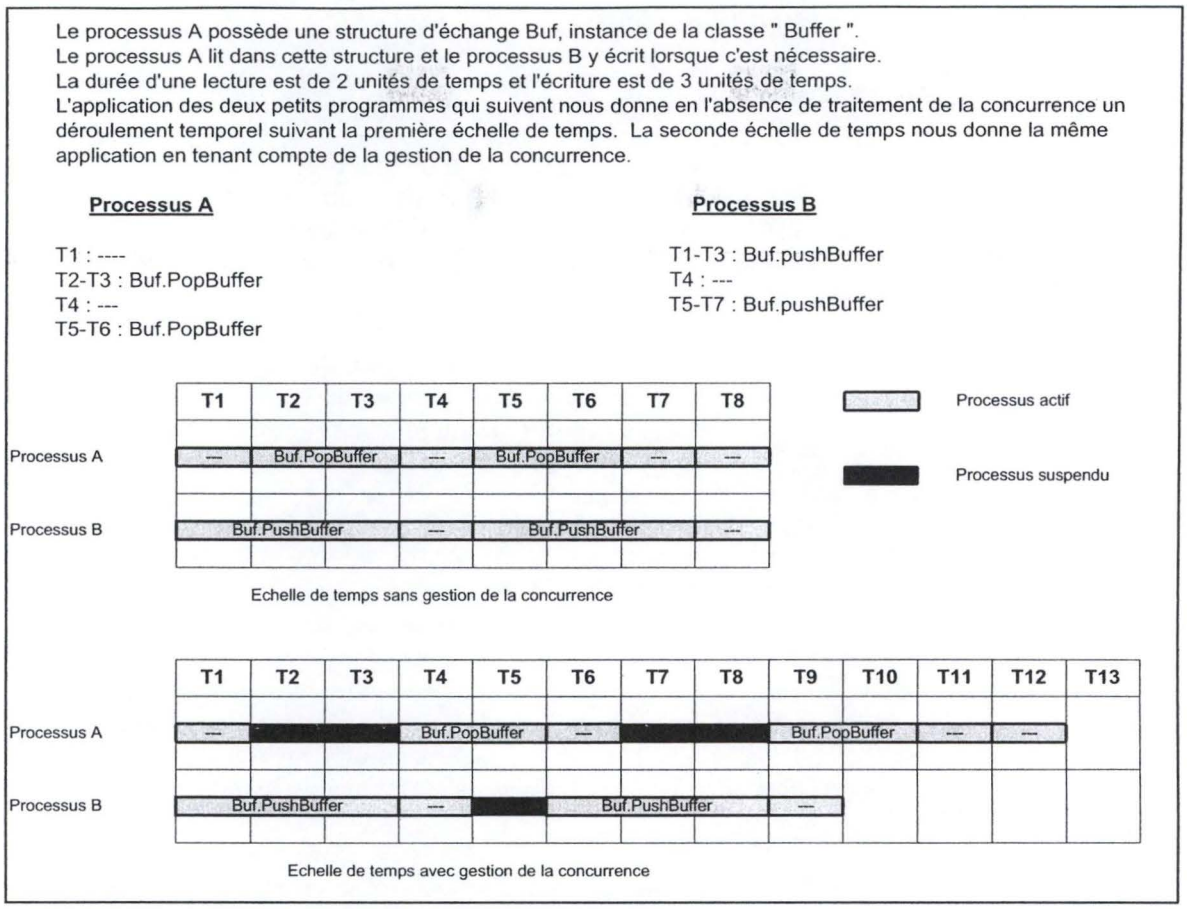


Figure 26 : Exemple de gestion des processus concurrents

Dans le premier cas, on peut se rendre compte qu'il y a un conflit quant à la lecture et à l'écriture dans le buffer A aux temps T2-T3 et T5-T6. Dans le second cas, les conflits sont résolus par l'introduction de suspensions temporaires d'exécution d'un des deux processus à la base du conflit. Cette méthode est semblable à celle utilisée par *synchronized*.

2) Structure du domaine

La classe *Domain* remplit deux fonctions principales :

- d'une part, elle gère l'état global du domaine représenté. L'appel de sa méthode *startDomain* à partir de l'interface utilisateur permet de démarrer individuellement, dans un processus séparé, chacun des routeurs créés. A partir de ce moment, on peut déclarer le domaine comme étant démarré (champ booléen *start*). Cette classe conserve en permanence, via le vecteur *listOfRouterVector* une trace de tous les *processus-routeur* existants dans le domaine ;
- d'autre part, elle conserve un ensemble de paramètres concernant principalement les éléments de l'interface utilisateur, et qui sont utilisés dans le cadre de la sauvegarde de l'état du domaine.

b. Structure et utilisation d'un routeur

1) Structure d'un routeur

Un routeur doit contenir un ensemble d'informations qui lui sont propres. L'analyse, le traitement et l'évolution de ces données lui permettent de se situer dans le domaine et d'interagir avec les autres routeurs au sein du simulateur. Nous allons, dans ce qui suit, décrire chacune des structures de données importantes qui constituent les différents champs d'une instance de la classe *Router*. Entre parenthèses, nous donnerons le type :

- *routerID* (IP[24]) : un identifiant unique du routeur et qui doit être une adresse IP. Dans la pratique, cette adresse sera généralement la plus petite de ses adresses IP. Dans le cas du simulateur, cette adresse doit être explicitement fournie lors de sa création.
- *routerInterfaceVector* (Vector) : liste de toutes les liaisons utilisables pour la diffusion de données et partant de ce routeur. Une liaison est une instance de la classe *PointToPointLiaison*. Chaque liaison correspond à une interface de sortie pour le routeur. Une instance de la classe *PointToPointLiaison* peut être identifiée de manière unique dans tout le simulateur si on connaît la valeur de son champ *origine* (origine) et la valeur de son champ *linkID* (destination). Il faut aussi préciser qu'une seule liaison orientée est autorisée entre deux routeurs.
- *waitLink* (WaitLink) : liste de toutes les liaisons non utilisables pour le transfert de données et partant de ce routeur. Si le routeur n'est pas encore démarré, ces liaisons correspondront à toutes les liaisons dont il est à l'origine. Si le routeur est démarré, elles correspondront aux liaisons unidirectionnelles ou aux liaisons pour lesquelles le protocole *hello* n'est pas terminé.
- *routerDb* (LinkStateDb) : base de données d'état de ce routeur. Son contenu donne la topologie du réseau telle qu'elle est vue localement par le routeur. Cette base de données est remplie par les informations topologiques locales et par les messages envoyés par les autres routeurs du domaine. La classe *LinkStateDb* de laquelle dépend le champ *routerDb* propose une structure de données pour stocker les LSA de description ainsi qu'un ensemble de méthodes pour ajouter, supprimer, modifier et rechercher des informations. La structure complète de la base de données peut être visualisée sur l'interface graphique.
- *memberOf* (Domain) : référence vers la structure dans laquelle évolue le routeur. On peut considérer que *Domain* représente la structure de données principale du simulateur.
- *bufferIn* (Buffer) : file des messages entrants. Tout routeur qui envoie un message (un message est en fait un paquet OSPF) à un autre routeur le fait en insérant ce message dans son champ *bufferIn*. L'accès à ce champ doit se faire de manière exclusive à l'aide des méthodes fournies par la classe *Buffer*.
- *routingTable* (Vector) : table de routage du routeur. Chaque ligne de cette table décrit un chemin vers une destination en donnant l'adresse du prochain routeur sur ce chemin ainsi que le coût du chemin global.
- *runningRouter* (boolean) : indication de l'état de démarrage du routeur. Sa transition de faux à vrai indique son démarrage.
- *exit* (boolean) : indication de l'état d'arrêt du routeur. Sa transition de vrai à faux indique la demande d'arrêt du routeur.
- *screenX*, *screenY* (int) : indication de la position du routeur sur l'écran d'interface.
- *myLocalRouterTimer* (Timer) : indication du temps actuel pour un routeur. A intervalles réguliers, les méthodes liées à cette variable permettront d'incrémenter l'âge des LSA dans la base de données d'état et de prendre des décisions face à certaines limites autorisées.
- *myLocalFlag* (Flag) : variable permettant de gérer l'arrêt et le redémarrage du routeur lorsqu'on travaille en pas-à-pas.

[24] IP est une classe définie dans le simulateur.

2) Création d'un routeur

La création d'un routeur est initiée dans l'interface utilisateur. Cette création peut avoir lieu avant le démarrage du domaine ou durant son déroulement. Créer un routeur revient à :

- initialiser tous ses champs et en particulier son identifiant et ses différentes bases de données ;
- le positionner sur l'interface utilisateur ;
- l'ajouter à la liste des routeurs tenues par la classe domaine (champ *listOfRouterVector*)[25].

3) Démarrage d'un routeur

Le démarrage d'un routeur se fait de deux manières différentes suivant l'état du domaine :

- lorsque le domaine est démarré, tous les routeurs qui sont déjà créés sont également démarrés;
- lorsque le domaine est déjà démarré et qu'un nouveau router est créé, alors, il faut explicitement le démarrer via l'interface utilisateur.

Le démarrage d'un routeur se compose de plusieurs activités à réaliser dans un ordre précis :

- Le champ *BufferIn* est vidé. Cela est nécessaire car un routeur créé et non encore démarré peut recevoir des messages *hello* en provenance de ses voisins (un routeur envoie un message à son voisin en écrivant dans son buffer d'entrée). Comme il n'est pas démarré, il doit être considéré comme étant non existant dans le domaine. Le fait de vider le buffer au démarrage permet de simuler cette non-existence dans le domaine malgré le fait qu'il a déjà été créé ;
- le routeur est déclaré comme étant démarré (champ *start*) et il change sa couleur sur l'interface ;
- le routeur envoie alors un message *hello* à destination de tous ses voisins directs par chacune de ses liaisons non encore activées ;
- le router crée son LSA qui décrit sa configuration locale en appelant sa méthode *createLocalLSA*. L'âge du LSA est initialisée à zéro. Le routeur ajoute le LSA dans sa base de données d'état. Ce premier LSA ne peut pas être diffusé car, dans cet état, le routeur n'a pas encore pu recevoir de message de confirmation[26] *hello* lui indiquant qu'une de ses liaisons est valide. Au fur et à mesure qu'il recevra des messages de confirmation, il pourra adapter son LSA local et dès lors le diffuser sur les liaisons valides ;
- si le déroulement du simulateur est prévu en pas-à-pas, alors, celui-ci est bloqué à ce niveau et il attend un déblocage explicite ;
- finalement, le routeur appelle sa méthode *checkBuffer* qui le met dans la boucle qui gère le protocole d'inondation.

4) Arrêt d'un routeur

L'arrêt d'un routeur doit être explicitement demandé par l'utilisateur (cela revient à terminer son processus). Il est possible d'arrêter un seul routeur ou de les arrêter tous lorsqu'on arrête le domaine. L'arrêt d'un routeur se déroule en plusieurs étapes :

- les routeurs du domaine sont suspendus. Cette suspension est nécessaire car la suppression d'un routeur implique des modifications structurelles chez ses voisins ;

---

[25] Un processus peut être initialisé sans pour autant être en exécution. Il ne le sera qu'au moment où on lui appliquera sa méthode start ().

[26] Il n'a, en fait, pas encore commencé à lire dans son champ *bufferIn*. Cela ne se fera que lorsqu'on appellera sa méthode *checkBuffer*.

- le champ *exit* du routeur est mis à la valeur *vraie* ce qui implique que le routeur quitte la boucle de la méthode *checkBuffer* qui gère le protocole d'inondation ;
- les liaisons allant et venant de ce routeur sont supprimées. On retire également le routeur de la liste des routeurs (champ *listOfRouterVector*) ;
- le processus de ce routeur arrive alors à son terme et il disparaît automatiquement.

c. La diffusion des LSA

1) La structure du routerLSA

Un routerLSA a pour but de décrire la topologie locale du routeur. Le simulateur décline les LSA en deux classes :

- d'une part, la classe *LSA* qui reprend les éléments communs à tous les types de LSA. Il s'agit en fait des champs qui constituent l'en-tête du LSA. Ces champs sont :
  - *lsaAge* (int) qui représente l'âge du LSA;
  - *lsaType* (int) qui représente le type du LSA;
  - *lsaStateLinkID* (IP) qui représente l'identifiant du LSA à savoir dans ce cas l'adresse du routeur source;
  - *lsaAdvertisingRouter* (IP) qui représente l'identifiant du routeur à l'origine du LSA;
  - *lsaSequenceNumber* (int) qui représente le numéro de séquence du LSA.
- d'autre part, la classe *RouterLSA* qui descend directement de la classe *LSA* et qui reprend l'ensemble de la description des liaisons point-à-point (dans le vecteur *routerLsaLinkVector*) dont le routeur local est à l'origine. Une liaison est décrite par l'ensemble des champs qui la constitue.

2) La structure des paquets OSPF

Les LSA de description doivent pouvoir être diffusés dans tout le domaine. Cette diffusion est réalisée en les encapsulant dans un paquet de type OPSF.

Le simulateur décline les paquets OSPF en trois classes :

- premièrement, la classe *OSPFPaquet* qui reprend les éléments communs à tous les paquets OSPF. Il s'agit ici des champs qui en constituent l'en-tête. Ces champs sont :
  - *ospfSender* (IP) qui représente le routeur à l'origine de l'émission du paquet OSPF;
  - *ospfType* (int) qui représente le type de paquet OSPF (1 pour Hello et 4 pour Update);
- deuxièmement, la classe *OSPFPaquetUpdate* descendant direct de la classe *OSPFPaquet*. Cette classe est responsable de la structure de données utilisée lors de la diffusion des messages par le protocole d'inondation. Une instance de cette classe possède un seul champ *paquetInformation* pour contenir un LSA ;
- troisièmement, la classe *OSPFPaquetHello* descendant direct de la classe *OSPFPaquet*. Cette classe est utilisée pour la transmission de messages par le protocole *hello*. Elle possède un champ *neighbour* qui contient la liste des voisins connus par le routeur émetteur.

3) Le protocole d'inondation

Le protocole d'inondation peut se résumer de la manière suivante : « **un routeur doit propager un LSA qu'il a reçu vers tous ses voisins à l'exception de celui par qui le message est arrivé, si et seulement si ce LSA est intégré dans sa propre base de données**».

La figure 27 donne un aperçu schématique du traitement de la diffusion.

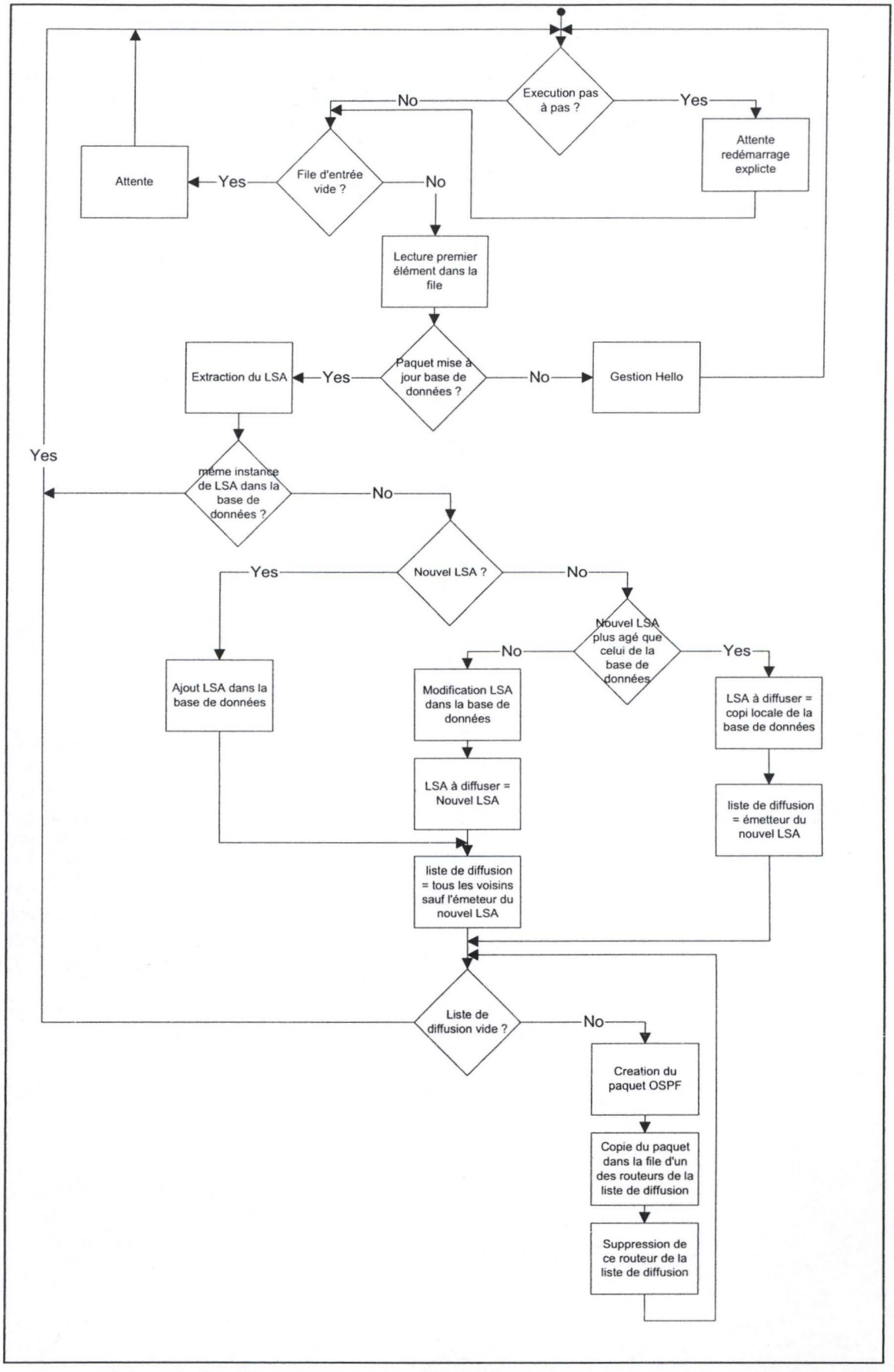


Figure 27 : gestion du protocole d'inondation pour un routeur

Pour le protocole d'inondation, on peut considérer chaque routeur comme étant un processus itératif initié lors du démarrage du routeur et qui s'exécute en permanence jusqu'à son arrêt. A chaque rotation, un ensemble d'opérations est exécuté pour traiter un message de configuration que le routeur aurait reçu dans son buffer d'entrée et réaliser si nécessaire la diffusion de ce message à ses voisins. Chacune des opérations est traitée par une méthode particulière de la classe *Router*. La figure 27 donne une vue globale du processus.

1. Vérifier si des informations n'ont pas encore été traitées (méthode *checkBuffer* de la classe *Router*). La file d'entrée (champ *bufferIn* de la classe *Router*) qui contient les messages adressés au routeur est analysée :

   - Si elle est vide, alors, le routeur n'a pas reçu de nouveaux messages ou tous les messages qu'il avait précédemment reçus sont traités ; dans ce cas, le routeur se met en attente durant 100 msec puis il analyse à nouveau la file d'entrée.
   - Si la file n'est pas vide, le routeur détermine le type du paquet lu. Si le paquet a trait au protocole *hello*[27], alors il est transmis à la méthode responsable de sa gestion. S'il s'agit d'un paquet contenant un LSA, alors, il appelle la méthode *forwardPaquet* de la classe *Router* pour initier le traitement et la propagation du message en tête de file. Le message qui est traité est alors retiré de la file d'entrée.

   Si l'exécution du simulateur se déroule en pas-à-pas, c'est durant cette étape que le routeur est bloqué. Sa libération doit être demandée de manière explicite par l'utilisateur.

2. Initier le traitement et la diffusion d'une information (méthode *forwardPaquet* de la classe *Router*). Le LSA du message est extrait. Dans un premier temps, celui-ci va être traité (méthode *updateDb* de la classe *Router*) puis, si nécessaire, il va être diffusé (méthode *diffuseLSA* de la classe *Router*) à tous les destinataires autorisés.

   2.a. Traitement local du LSA (méthode *updateDb* de la classe *Router*). Cette étape vérifie si une même instance du LSA reçu existe déjà dans la base de données, auquel cas, aucune modification ni propagation ne peuvent être entreprises ; on se trouve dans le cas où un message de mise à jour est parvenu par une interface et que ce même message arrive également via une autre interface du routeur de par le processus d'inondation.
   Si l'instance du LSA reçue est plus grande (en terme de numéro de séquence) qu'une autre instance du même LSA stockée localement, alors, on remplace le LSA local par la nouvelle instance et on autorise la diffusion (vers tous ses voisins à l'exception de celui par qui le message est arrivé). Si cette instance est totalement nouvelle[28], alors, on l'ajoute à la base de données et on autorise également la propagation.
   Si l'instance reçue est plus petite qu'une autre instance du même LSA stockée localement dans la base de données, alors, c'est le routeur émetteur du LSA reçu qui n'a pas son information à jour. Il faut alors envoyer à cet émetteur, et à lui seul, l'instance du LSA correspondant au sien et qui est stockée localement dans la base de données.

   2.b. Diffusion d'un LSA (méthode *diffuseLSA* de la classe *Router*). Dans cette étape, le routeur crée et envoie un nouveau paquet OSPF contenant le LSA qu'il doit diffuser. Si le paquet doit être diffusé globalement, alors, en se basant sur la liste de ses voisins (qu'il connaît en analysant ses interfaces de sortie), le routeur va copier dans la file d'entrée de chacun d'eux (à l'exception de celui par lequel le LSA lui est parvenu) ce paquet OSPF. Si le paquet ne doit être diffusé qu'à l'émetteur, alors, il n'est copié que dans la file d'entrée de celui-ci. A partir de ce moment, le ou les

[27] Voir infra.
[28] Il n'existe aucun LSA dans la base de données locales qui soit identique (en faisant abstraction du numéro de séquence qui permet de distinguer deux instances d'un même LSA) au LSA reçu.

voisins sont à même d'analyser le message suivant la procédure que nous venons de décrire. La transition d'un message sur une liaison est marquée à ce niveau par un changement temporaire de couleur du segment qui relie le routeur émetteur au destinataire du message.

Il est rappelé que le champ *bufferIn* de chacun des routeurs est une ressource partagée entre plusieurs processus. Il est donc nécessaire de le protéger tant en lecture qu'en écriture.

d. Gestion des liaisons

Toutes les liaisons point-à-point connues par un routeur sont des instances de la classe *PointToPointLiaison*. Cette classe est elle-même un descendant de la classe *Liaison* qui regroupe les éléments communs à toutes les liaisons. Il est à noter qu'une liaison est unidirectionnelle ; aussi, une liaison bi-directionnelle sera représentée par deux liaisons unidirectionnelles. Une liaison est décrite par un ensemble de paramètres :

- *origine* (Router) donne le routeur qui est à la source de la liaison point-à-point décrite ;
- *linkID* (IP) donne l'identifiant du routeur qui constitue la destination de la liaison décrite (une liaison est toujours orientée) ;
- *linkType* (int) décrit le type de la liaison. Dans l'état actuel du simulateur, seul le type 1 (liaison point-à-point) existe ;
- *linkMetric* (int) donne le coût affecté à la liaison décrite ;
- *uniquePictureID* (int) donne l'identifiant unique affecté à la liaison. Ce champ ne fait pas à proprement parlé de la description d'une liaison ; il est utilisé dans le cadre de la représentation de la liaison sur l'interface utilisateur.

On peut considérer qu'il existe deux types de liaison point-à-point dans le simulateur :

- les liaisons actives qui relient deux routeurs actifs et qui permettent l'échange de données. Ces liaisons sont gérées au niveau de l'instance de chaque routeur dans la structure de données *routerInterfaceVector*;
- les liaisons inactives[29] qui relient deux routeurs dans des états quelconques et qui ne permettent aucun échange de données. Ces liaisons existent mais elles ne sont pas encore reprises dans la structure de données *routerInterfaceVector* qui en est à l'origine. Le protocole *hello* permet de transformer une liaison inactive en une liaison active. Ces liaisons en attente sont décrites au niveau de l'instance de chaque routeur dans la structure de données *waitingLink*. Chacune de ces liaisons inactives possèdent un timer qui permet au routeur d'envoyer périodiquement des messages *hello* les concernant.

1) Le protocole hello

Comme mentionné dans la description, le protocole *hello* ne sera utilisé que pour rendre actif et ce de manière automatique, une liaison bidirectionnelle créée entre deux routeurs. La figure 28 donne la machine à états finis montrant la communication entre deux routeurs entre lesquels une nouvelle liaison doit être activée. Dans cette figure, un cercle correspond à un état et une liaison orientée indique le message permettant la transition entre deux états. Cette machine à états finis correspond à l'activation d'une seule liaison bidirectionnelle. Au sein d'un routeur, plusieurs instances de cette machine peuvent coexister, chacune traitant une liaison bien précise.

[29] Il s'agit principalement des liaisons qui relient un routeur démarré à un routeur non démarré et des liaisons entre deux routeurs démarrés pour lesquelles le protocole *hello* n'est pas terminé.

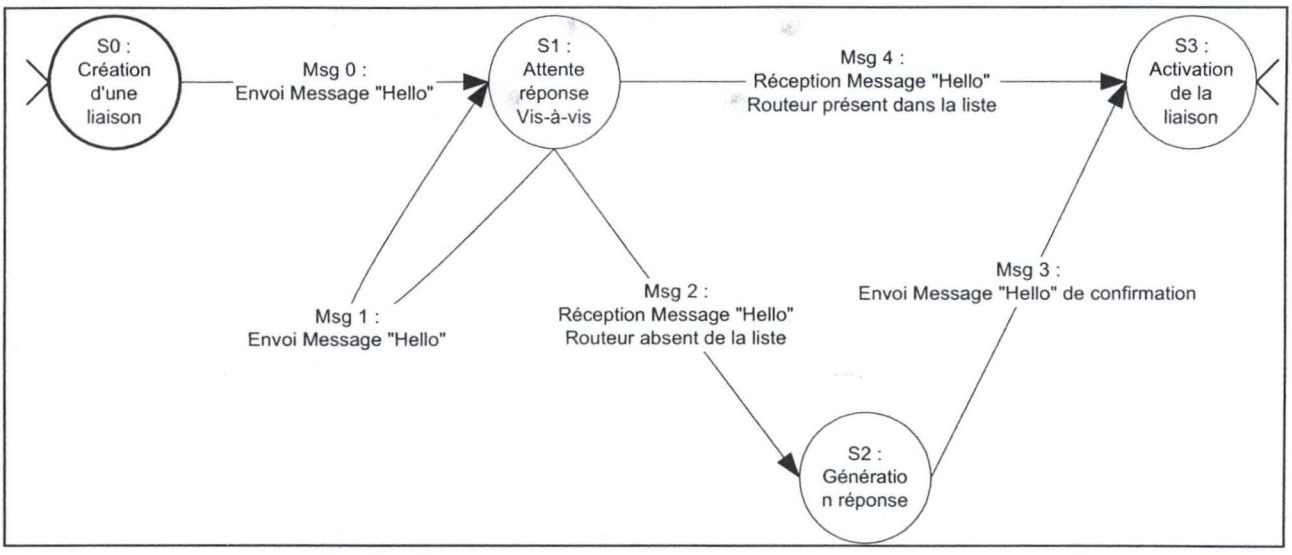


Figure 28 : Machine à états finis du protocole hello

S0 : une nouvelle liaison est créée, ayant le routeur courant comme origine. Si ce routeur est démarré, alors, il génère directement un message OPSF *hello* (msg 0) à destination du vis-à-vis. Ce message reprend la liste des voisins que ce routeur connaît à ce moment. Le routeur à l'autre bout de la liaison ne peut jamais se retrouver dans la liste car, de par les spécifications, il ne peut exister qu'une seule liaison entre deux routeurs et parce que , même si la liaison est bidirectionnelle, le routeur courant ne lira le message OPSF *hello* envoyé par l'autre routeur que lorsqu'il aura envoyé son propre message (la méthode *addNewInterface* qui ajoute une interface envoie le message OPSF *hello* avant d'autoriser toute autre lecture dans la file d'entrée) . Le routeur se trouve dès lors dans l'état S1. L'envoi de ce message est réalisé par la méthode *sendOnehello* de la classe *Router*.

S1 : le routeur courant attend maintenant une réaction de son vis-à-vis sur la liaison considérée. Plusieurs cas peuvent se présenter suivant l'état des routeurs concernés, l'ensemble étant traité par la méthode *helloRespond* de la classe *Router* :

- le routeur vis-à-vis n'est pas encore démarré ; il est dans l'incapacité de répondre. Dans ce cas, le message OPSF *hello* (msg 1) est répété toutes les dix secondes jusqu'au moment où le vis-à-vis pourra répondre (le msg 1 est fort semblable au msg 0; seule la liste des voisins connus peut évoluer). Dans cette situation, la liaison ne peut pas être utilisée car elle est unidirectionnelle ;
- le routeur courant reçoit un message OPSF *hello* venant de son vis-à-vis et ce message ne contient pas le routeur courant dans la liste des routeurs connus par le vis-à-vis (msg 2). Ce cas est celui de la réception par le routeur courant d'un message de type msg 0 ou msg 1. De ce message, le routeur courant peut tirer la conclusion qu'il peut voir un nouveau voisin. Il transite dans l'état **S2**. S'il peut répondre, alors il envoie un message OPSF *hello* à son vis-à-vis en le mentionnant cette fois dans la liste de ses voisins (msg 3). Il peut déduire que la liaison est bidirectionnelle et il transite alors vers l'état S3 ;
- le routeur courant reçoit un message *hello* venant de son vis-à-vis et ce message contient dans sa liste l'identifiant du routeur courant (msg 4). Ce message peut être par exemple la réponse générée au point précédent. De cela, le routeur courant peut en déduire qu'il est vu par son vis-à-vis et que la liaison est bidirectionnelle. Il peut donc directement transiter vers l'état S3.

S3 : le routeur active la liaison pour la rendre opérationnelle. L'activation est réalisée par la méthode *activateLink* de la classe *Router*. Cette méthode va déplacer la description de la liaison de la structure *waitingLink* vers la structure *routerInterfaceVector*. L'ajout d'une nouvelle liaison entraîne aussi la modification de la configuration locale. De ce fait, un nouvel LSA descriptif va être généré et diffusé dans le domaine.

Il est à noter que l'échange des messages *hello* entre les routeurs se fait en utilisant des paquets OSPF. Au niveau de la méthode *CheckBuffer*, un tri est réalisé[30] entre les messages OSPF de mise à jour (type 4) et les messages OSPF *hello* (type 1). Le traitement d'un message de type 1 implique l'appel de la méthode *helloRespond*.

2) Création, suppression et modification d'une liaison

La méthode *addNewInterface* de la classe *Router* est responsable de la création d'une nouvelle liaison entre deux routeurs. La nouvelle liaison créée est placée dans la liste des liaisons en attente et, si le routeur est actif, un message *hello* est directement envoyé au vis-à-vis. Cette méthode est invoquée à partir de l'interface utilisateur. Si l'utilisateur veut créer directement une liaison bidirectionnelle, alors, la méthode *addNewInterface* est invoquée une fois pour chacun des deux routeurs concernés (on crée en fait deux liaisons unidirectionnelles en sens opposé).

La méthode *removeInterface* de la classe *Router* gère la suppression d'une liaison unidirectionnelle entre deux routeurs. Cette méthode est invoquée à partir de l'interface utilisateur. La liaison concernée est supprimée de la liste des liaisons actives. Si la liaison entre les deux routeurs était bidirectionnelle, alors, la liaison restante est transférée de la liste des liaisons actives *routerInterfaceVector* vers la liste des liaisons inactives *waitingLink*.
La suppression d'une liaison va entraîner la modification de la configuration locale. De ce fait, un nouvel LSA descriptif va être généré et diffusé dans le domaine.

L'ajout ou la suppression d'une liaison entraîne aussi respectivement la création/suppression de sa représentation sur l'interface utilisateur. Cet ajout/suppression est géré dans la classe *ChoiceLiaison*.

La méthode *modifyInterface* de la classe *Router* est responsable de la modification d'un coût affecté à une liaison. Comme précédemment, une modification entraîne la génération d'un nouvel LSA descriptif.

e. La gestion du temps

Le temps est nécessaire au niveau des routeurs afin de procéder à l'incrémentation de l'âge des LSA dans la base de données d'état et à prendre des décisions quant à leur suppression ou à l'envoi d'un rafraîchissement à travers le domaine. Le temps est aussi utilisé pour l'envoi des messages *hello* lorsque le vis-à-vis dans une liaison ne répond pas.

Chaque processus routeur démarré possède une horloge matérialisée par le champ *myLocalRouterTimer*. A ce champ est liée une action qui se déclenche toutes les secondes (l'action est implémentée dans la classe *ActionOnTimer*). Cette action réalise plusieurs opérations :

- incrémentation de la valeur du champ *Age* de tous les LSA présents dans la base de données d'état du routeur ainsi que de l'âge de toutes les liaisons non encore activées présentes dans *waitingLink* ;
- si la valeur du champ *Age* pour un LSA quelconque dépasse 3.600.000 msec, alors, ce LSA est obsolète et il est supprimé de la base de données d'état ;
- si la valeur du champ *Age* du LSA local (c'est à dire le LSA qui décrit la configuration locale du routeur) dépasse 1.800.000 msec, alors, ce LSA doit être rafraîchi dans tout le

[30] Voir supra

domaine. Un nouvel LSA est généré, identique au précédent sauf pour son champ *lsaSequenceNumber* qui est incrémenté de 1 et son champ *Age* qui prend la valeur 0. Ce nouvel LSA remplace l'ancien dans la base de données d'état local et il est propagé vers les voisins directs du routeur via la méthode *diffuseLSA* ;

- si l'âge d'une liaison en attente pour laquelle le vis-à-vis n'a envoyé aucune réponse suite à l'envoi d'un message *hello* atteint 10.000 msec, alors, un nouveau message *hello* à l'intention de ce vis-à-vis est généré.

f. Elaboration de la table de routage

1) Algorithme de Dijsktra – SPF

Un réseau peut sans aucun problème être représenté sous la forme d'un graphe orienté G=(S,A) où S est l'ensemble des nœuds (dans notre cas, ces nœuds seront assimilés aux routeurs) et A représente l'ensemble des arcs orientés (un arc sera équivalent à une liaison point-à-point unidirectionnelle). Chaque arc possède un poids dont la valeur est positive ou nulle. Un des sommets (dans le cas du simulateur, ce sommet correspondra au routeur dont on veut déterminer la table de routage) est désigné comme étant la source.

Le problème consiste à déterminer le chemin ayant le coût le plus faible entre la source et chacun des autres sommets de S. Le coût d'un chemin est donné par la somme des poids des arcs qui constituent ce chemin. L'algorithme SPF - Shortest Path First – permet de résoudre ce problème [AHU87 p 206]. Nous présupposons ici que le graphe est fortement connexe c'est à dire qu'il existe toujours un chemin pour relier deux sommets distants.

Cet algorithme tient à jour un ensemble E de sommets dont les distances les plus courtes par rapport à la source (le routeur qui effectue l'algorithme) sont connues. Au départ, E ne contient que la source. A chaque étape, on ajoute à E l'un des sommets restants r ($r \in S$ et $r \notin E$) dont la distance à la source est la plus courte possible (cette distance ne peut être obtenue qu'en prenant des nœuds déjà présents dans E, à l'exception de la destination). Si tous les arcs ont des poids supérieurs ou égaux à 0, alors, il est toujours possible de trouver un chemin minimal depuis la source vers ce sommet en ne passant que par des sommets déjà présents dans E. A chaque itération, on adapte si nécessaire la valeur des distances des sommets non encore intégrés dans E en fonction de l'intégration de r dans E. Quand tous les sommets ont été insérés dans E, tous les chemins obtenus sont les chemins les plus courts par rapport à la source.

La figure 29 donne le pseudo-code de l'algorithme *Shortest Path First* de Dijsktra.

On considère que les sommets sont numérotés de 1 à n avec 1 comme numéro pour la source
Soit E , l'ensemble des sommets dont on connaît le chemin le plus court vers la source ;
Soit S, l'ensemble des sommets à l'exception de la source ;
Soit P, un tableau représentant le poids des arcs (i,j) ;
Soit D, un tableau donnant pour chaque sommet, le coût de son chemin minimal vers la source (en ne passant que par des nœuds présents dans *E* sauf pour le sommet destination) ou l'infini si ce chemin n'est pas actuellement déterminable ;
Soit C, un tableau de sommets tel que C(s) contient le nom du sommet précédant immédiatement s dans le plus court chemin ;

E := {1} -- *initialisation de E*

Pour i := 2 **à** n **Faire**
D[i] := P [1,i] -- *initialisation de D*
C[i] := 1 -- *initialisation de C*

Pour i := 1 **à** n – 1 **Faire**
Choisir un sommet t tel que $D[t] = \min_{x \in S \setminus E} D[x]$ -- *choix du noeud le plus proche*

E := E ∪ {t} -- *intégration de ce noeud dans l'ensemble des noeuds les plus proches*

Pour ∀ s ∈ S \ E **Faire**

D[s] := min (D[s], D[t] + P[t,s]) --*évaluation du poids des noeuds restants*
Si D[t] + P[t,s] < D[s] **alors** C[s] := t -- *adaptation éventuelle des chemins*

Figure 29 : Pseudo - code de l'algorithme SPF de Dijsktra

2) Architecture de la création des tables de routage

Le simulateur OSPF se base sur l'algorithme de Dijsktra pour déterminer sa table de routage. Celle-ci est élaborée lorsque l'utilisateur du simulateur en fait la demande. Le calcul se déroule en quatre étapes distinctes.

1. Détermination de la topologie actuelle du domaine telle qu'elle est perçue par le routeur qui crée sa table de routage. Ce routeur retire les informations topologiques de sa base de données d'état. Le domaine est représenté par un tableau où les en-têtes de lignes et de colonnes représentent les routeurs et où chaque intersection (X,Y) possédant une valeur nulle ou positive représente le poids d'une liaison orientée unissant le routeur X au routeur Y. La méthode *makeNetworkArray* de la classe *Router* est responsable de cette étape.

2. Application de l'algorithme de Dijsktra. A partir du tableau calculé au point précédent, l'algorithme va déterminer, pour chaque destination, le coût du chemin le plus court depuis la source (qui est ici le routeur pour lequel on calcule la table de routage) ainsi que l'adresse du nœud qui précède directement la destination dans le chemin le plus court. Il est possible qu'une destination ait plusieurs chemins de coûts minimaux égaux ; dans ce cas, ils seront tous pris en compte. La méthode *shortestPathFirst* de la classe *Router* est responsable de cette étape.

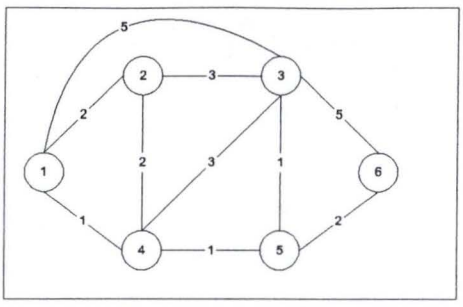
3. Détermination du prochain saut pour chaque destination. A partir des résultats obtenus dans la seconde étape, il est possible de remonter un chemin de sa destination jusqu'à sa source (tout nœud sauf le nœud source possède un prédécesseur). La méthode *createPathTable* en

est responsable ; elle va fournir, pour chaque chemin, la liste complète des nœuds à parcourir. De ce résultat, on connaît alors directement le premier nœud suivant la source sur le meilleur chemin pour joindre une destination. La méthode *createRoutingTable* de la classe *Router* est responsable de l'extraction de cette information.

4. Affichage du résultat sous la forme d'un tableau reprenant pour chaque destination le coût et le(s) routeur(s) suivant(s) sur le(s) chemin(s) vers cette destination. Cette étape est prise en charge par la méthode *printRoutingTable* de la classe *Router*. Le contenu de la table de routage est également stocké au niveau du champ *routingTable* du routeur source. Cette méthode est directement invoquée lorsque l'utilisateur du simulateur demande à obtenir la table de routage d'un routeur qu'il désigne.

L'algorithme de Dijsktra ne détermine les tables de routage que pour les sources et les destinations actives (les routeurs non démarrés ne sont pas pris en compte). Le nombre de routeurs actifs qu'on considère dans l'algorithme est celui qu'on a au début de l'appel. Toute insertion de routeurs durant l'exécution de l'algorithme n'est pas prise en compte. De plus, la suppression d'un routeur durant le calcul est interdite (tous les routeurs sont d'ailleurs suspendus durant un effacement).

La figure 30 donne un exemple d'exécution de l'algorithme de Dijskstra. Dans cet exemple, on recherche les plus courts chemins pour joindre le nœud 1 à tous les autres nœuds du réseau.



Topologie complète du réseau

Légende

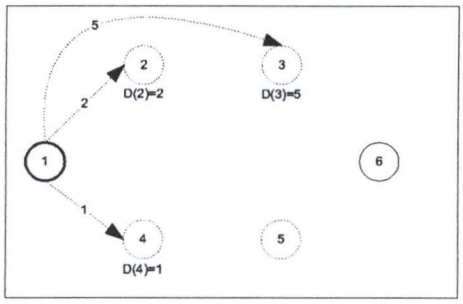
4 Noeud ajouté à D

1 Noeud appartenant à D

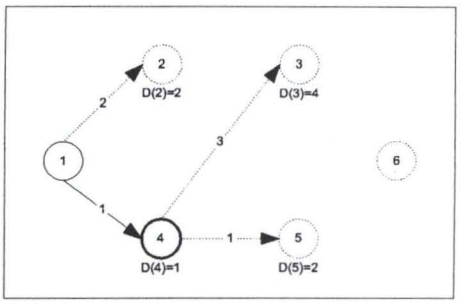
3 Noeud candidat

1 Liaison effective avec son poid

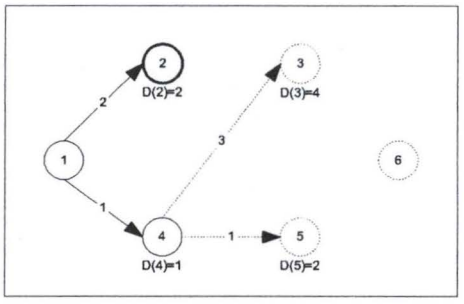
2 Liaison candidate avec son poid



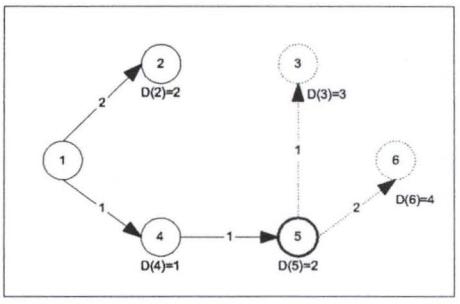
Initialisation



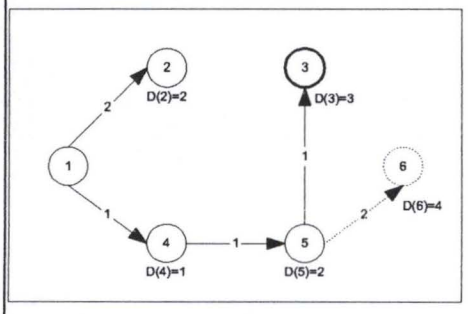
Première Itération



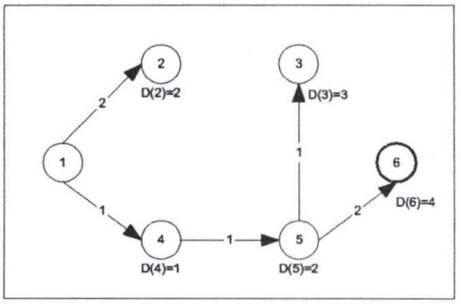
Seconde Itération



Troisième Itération



Quatrième Itération



Cinquième Itération

Itération	E	D[2]	C[2]	D[3]	C[3]	D[4]	C[4]	D[5]	C[5]	D[6]	C[6]
Init	{1}	2	1	5	1	1	1	ω	-	ω	-
1	{1,4}	2	1	4	4	1	1	2	4	ω	-
2	{1,2,4}	2	1	4	4	1	1	2	4	ω	-
3	{1,2,4,5}	2	1	3	5	1	1	2	4	4	5
4	{1,2,3.4,5}	2	1	3	5	1	1	2	4	4	5
5	{1,2,3,4,5,6}	2	1	3	5	1	1	2	4	4	5

Tableau reprenant les différentes itérations de l'algorithme de Dijsktra

Figure 30 : Exemple d'application de l'algorithme de Dijsktra

Chapitre 4 : architecture de OMP

1. Spécification du simulateur OMP

OMP est avant tout un protocole additionnel au protocole OSPF. A la différence de OSPF que l'on pouvait aisément diviser en deux parties, l'une étant indispensable et l'autre ne faisant qu'améliorer les performances, OMP forme un tout indivisible ; chaque partie du protocole (Next Hop Structure – Opaque LSA – Calcul de la répartition des charges) étant complémentaire des autres. OMP sera donc ici considéré dans son entièreté en émettant cependant deux nuances :

1) pour bien percevoir les effets de OMP par rapport à une répartition classique de la charge (ECMP), il faut que l'utilisateur indique le trafic existant entre les différents routeurs du domaine[31]. Dans un tel contexte, l'utilisation des variables SNMP n'est pas envisageable. Celles-ci sont donc remplacées par des valeurs issues du calcul de la charge affectée à chacune des liaisons et ce, sans faire de différence entre les valeurs entrantes et les valeurs sortantes. Afin de simplifier les calculs, nous considérons également que la capacité de toutes les lignes est identique (ce paramètre pourra donc être négligé) ;
2) la spécification de OMP prévoit qu'il ne peut y avoir qu'un seul segment critique pour une destination donnée. Aucune règle n'est prévue pour traiter l'existence, pour une même destination, de plusieurs segments critiques ayant chacun la même valeur pour la charge équivalente (par segment, on entend une liaison point-à-point joignant deux routeurs). Dans un tel cas de figure, il a été décidé de déroger à la règle et d'autoriser la présence de plusieurs segments critiques pour une même destination.[32]

a. *Spécification des fonctionnalités*

Le protocole OMP doit s'intégrer dans le simulateur OSPF. Chaque fois que possible, les mécanismes déjà existants doivent être utilisés. Cela est vrai, entre autres, pour le transfert des informations de charge (opaque LSA) entre les différents routeurs du domaine.

Le simulateur doit offrir la possibilité à l'utilisateur d'introduire, à l'aide d'un tableau, le trafic qu'il veut *générer* entre les différents routeurs du domaine. Ces routeurs ne doivent pas obligatoirement être directement reliés. Le simulateur, sur base de ces informations, doit être capable de calculer et d'afficher pour chaque liaison du domaine, sa charge ainsi que ses pertes en se basant soit sur une répartition classique de la charge soit sur les concepts de OMP. A tout moment, le trafic donné par l'utilisateur peut être modifié. La matrice du trafic peut être sauvegardée dans un fichier et récupérée ultérieurement.

En se basant sur la charge calculée des liaisons et sur l'algorithme de renflouage de OMP, tout routeur doit être capable de diffuser dans le domaine, à l'aide de opaque LSA, et lorsque le protocole OMP le prévoit, la charge et les pertes de chacune des liaisons dont il est à l'origine. Ces opaque LSA doivent être récupérables et analysables par tous les routeurs du domaine. De plus, ces opaque LSA doivent être visualisables dans l'interface utilisateur lors de la consultation de la base de données locale.

Tout routeur dans le domaine doit posséder une *next Hop Structure* reprenant toutes les informations nécessaires à savoir :

[31] Par trafic, on entend la quantité d'information qu'un routeur A veut envoyer au travers du domaine à un routeur B qui ne lui est pas forcément directement connecté.
[32] Voir infra.

- pour chaque destination, la liste des chemins décrits complètement et permettant de rejoindre cette destination (le calcul des chemins devra se baser sur les algorithmes existants dans OSPF) ;
- pour chaque destination, la liste des segments critiques avec la valeur de leur charge équivalente ;
- pour chaque chemin décrit, l'ensemble des variables qui permettent de déterminer la proportion du trafic accordé à ce chemin ainsi que les variables utilisées pour déterminer l'évolution de l'accroissement de la proportion de trafic accordé.

Cette structure peut être visualisée complètement via l'interface utilisateur.

Toute modification dans la topologie du domaine, diffusée à l'aide des LSA de mise à jour, doit automatiquement entraîner une modification en conséquence de la *next Hop Structure* locale.

Toute réception d'un opaque LSA doit immédiatement entraîner un calcul de la répartition de la charge pour les destinations vers lesquelles la liaison décrite par le opaque LSA est considérée comme critique. De la même manière, toutes les 15 secondes, il est vérifié, en fonction des critères édictés par le protocole OMP, s'il est nécessaire de réaliser une adaptation de la répartition du trafic.

L'adaptation de la répartition se déroule en plusieurs étapes telles que décrites dans le protocole. Celles-ci sont, pour chaque destination joignable par un routeur donné :

1) détermination de la possession par les différents chemins d'au moins un segment critique ;
2) adaptation éventuelle des valeurs d'incrémentation ;
3) calcul effectif de la répartition de la charge entre les différents chemins vers la destination.

Lorsque toutes les destinations sont traitées, un nouveau calcul des charges des différentes liaisons est entamé. Son résultat peut localement entraîner la diffusion d'un opaque LSA.

2. Architecture de OMP

Partant des spécifications du point précédent, nous allons maintenant décrire les grands concepts de l'implémentation de OMP dans le simulateur OSPF. Les remarques soulevées dans l'architecture du simulateur OPSF concernant JAVA et l'interface graphique restent d'application. Les champs et les méthodes propres à OMP ont été directement ajoutés, lorsque cela s'avérait nécessaire, aux classes déjà existantes.

L'architecture sera décrite de la manière suivante :

- description de la *next Hop Structure*, de sa création et de son adaptation aux modifications topologiques ;
- gestion de l'adaptation de la charge des différents chemins en abordant tour à tour les informations supplémentaires contenues dans la description des liaisons possédées par un routeur, la gestion des opaque LSA et finalement la description de la procédure d'adaptation à proprement parler;
- description de l'algorithme de calcul de la charge des différentes liaisons dans un domaine ;
- description des grandes modifications apportées aux classes *Domain* et *Router*.

a. Next Hop Structure

La *next Hop Structure* est une structure de données présente dans tous les routeurs qui appliquent le protocole OMP. Cette structure poursuit deux grands buts :

1) donner pour chaque destination joignable dans le domaine, l'ensemble des chemins utilisables pour l'atteindre ainsi que la répartition du trafic et la gestion de cette répartition entre les différents chemins vers cette destination ;
2) détecter pour chaque destination joignable les segments critiques.

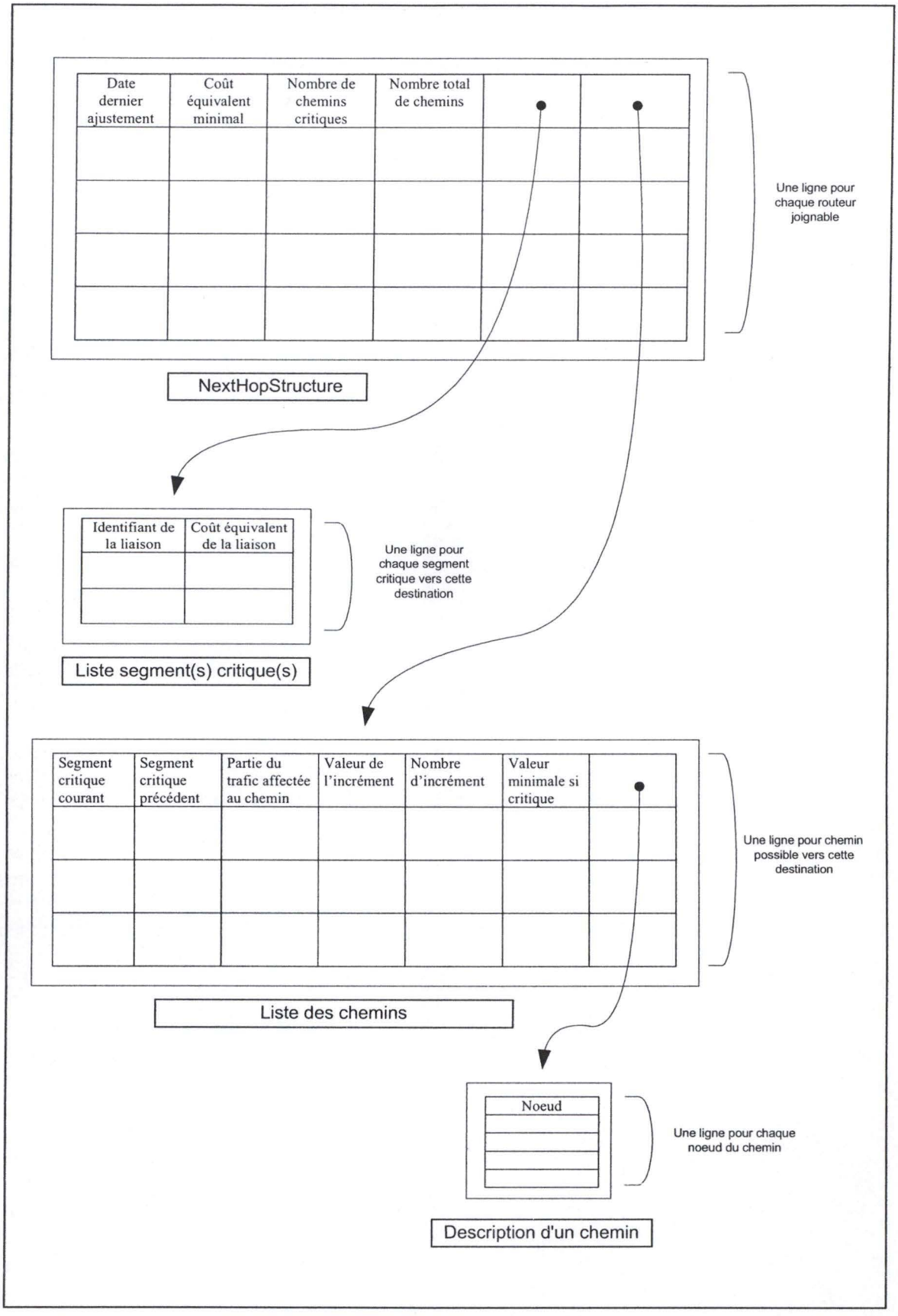


Figure 31 : Next Hop Structure d'un routeur

1) Description de la structure

La *next Hop Structure* est représentée dans une instance de la classe *Router* par le champ *localNextHopStruct*. Ce champ est en fait un tableau dont chacune des lignes contient les informations nécessaires pour gérer une relation avec un routeur distant. On retrouvera donc autant de lignes qu'il y a de routeurs actifs dans le domaine. Chaque ligne est une instance de la classe *NextHopStruct*. Cette classe possède plusieurs champs [33] :

- *date dernier ajustement - lastReadjust* (double) donne la date à laquelle cette instance a été ajustée pour la dernière fois. Cette valeur, combinée avec d'autres, permettra de déterminer le moment de l'ajustement suivant ;
- *coût équivalent minimum - lightestEquivLoad* (double) indique la charge équivalente la plus petite parmi tous les segments utilisés pour joindre la destination traitée. Ce champ servira également à déterminer le moment de l'ajustement suivant ;
- *nombre de chemins critiques - numberWithCritical* (int) indique le nombre de chemins qui possèdent le(s) segment(s) critique(s) ;
- *nombre total de chemins - totalPath* (int) donne le nombre de chemins différents découverts par OMP pour joindre la destination courante ;
- *localPath* (Vector) donne la liste des chemins découverts. Chaque ligne représente un chemin qui est décrit par une instance de la classe *SetOfPath*. Cette classe possède plusieurs champs pour décrire un chemin :

    - *segment critique courant - hasCriticalSegment* (boolean) indique si le chemin courant passe par au moins un des segments critiques ;
    - *segment critique précédent - hasPreviousCriticalSeg* (boolean) indique si le chemin contenait un des segments critiques lors de l'ajustement précédent ;
    - *partie du trafic affectée au chemin - trafficShare* (int) indique la proportion du trafic envoyée via ce chemin vers la destination. Cette proportion est un nombre allant de 0 (0% du trafic est affecté à ce chemin) à 65535 (100% du trafic est affecté à ce chemin) ;
    - *valeur de l'incrément - moveIncrement* (int) indique la proportion supplémentaire de trafic qui va être affectée à ce chemin dans le cas où OMP déciderait de déplacer du trafic sur ce chemin. Sa valeur par défaut correspond à 1 % du trafic. Au fur et à mesure que du trafic est déplacé vers ce chemin, sa valeur va augmenter ;
    - *nombre d'increment - moveCount* (int) indique le nombre de fois consécutives que OMP a déplacé du trafic vers ce chemin. Cette valeur est utilisée en combinaison avec la valeur précédente ;
    - *Valeur minimale si critique - minRateWithCritical* (int) indique la valeur minimale que devra prendre le champ *moveIncrement* lorsque celui-ci n'inclura plus un segment critique. Cette valeur, petite, permettra à un routeur de réorienter une petite partie du trafic sur cette liaison sans causer de problèmes d'oscillations.
    - *path* (Vector) décrit le chemin du routeur courant vers la destination. Chaque ligne de ce vecteur indique un nœud (sous la forme d'une instance de la classe *IP*) intermédiaire sur le chemin. Ainsi, deux nœuds consécutifs donnent une liaison utilisée par ce chemin.

- *listOfCriticalSegment* (Vector) donne la liste des segments critiques retenus pour cette destination. Chaque ligne de ce vecteur va décrire un segment critique sous la forme d'une instance de la classe *listOfCriticalSegment*. Cette classe possède plusieurs champs :

[33] Nous donnerons pour chaque champ, son nom sur la figure 31, son nom dans la classe et entre parenthèses son type. Cette règle est aussi valable pour les descriptions des structures suivantes.

- *identifiant de la liaison - segmentUniqueID* (int) identifie de manière unique une liaison au sein du domaine. On se base ici sur le champ *uniquePictureID* de la classe *PointToPointLiaison* ;
- *Coût équivalent de la liaison - linkEquivLoad* (double) donne le coût équivalent calculé pour cette liaison.

2) Création de la structure

La création d'une *next Hop Structure* complète pour un routeur donné est réalisée en appelant la méthode *createNextHopStructure* de la classe *Router*. Cette méthode prend en paramètre le nombre de routeurs en fonctionnement dans le réseau et retourne un tableau décrivant la *next Hop Structure*. Cette création se déroule en plusieurs étapes :

1. création de la table de routage classique du routeur par appel à la méthode *createRoutingTable* de la classe *Router* ;

2. création de la table de routage étendue du routeur. Cette table reprend les tables de routage de tous les routeurs directement connectés au routeur courant. La création de cette table est réalisée par la méthode *createRelaxedRoutingTable* de la classe *Router*. Il faut noter qu'ici, chacune des tables de routage des voisins ne donnera pas uniquement l'identifiant du routeur suivant vers une destination mais également l'ensemble des routeurs à traverser pour rejoindre cette destination. L'établissement du chemin complet pour chacune des destinations est réalisé par la méthode *createPathTable* de la classe *Router*. Une destination peut offrir plusieurs chemins de coût minimal ;

3. initialisation de la NextHopStructure. Cette étape consiste à introduire dans la *nextHopStructure* les informations connues localement (on exclut donc temporairement les informations obtenues pour les routeurs voisins – point 2) :

   i. un appel à *createPathTable* en partant du routeur local, permet de connaître le(s) meilleur(s) chemin(s) complet(s) vers chacune des destinations joignables dans le domaine ;

   ii. pour chaque destination (une destination correspond à une ligne dans la nextHopStructure), on va introduire dans la structure le ou les chemins (chaque chemin correspondra à une entrée dans le vecteur *localPath* de la ligne courante) découverts à l'étape 3.i. Suivant le nombre de chemins trouvés pour cette destination, on répartira la charge du trafic équitablement entre ceux-ci. A la fin, on aura donc une structure qui correspondra à la répartition de la charge entre les différents chemins possibles (à ce niveau, on ne prend en compte que les meilleurs chemins ; les autres chemins, découverts par OMP en analysant les tables de routages des voisins (point 2), seront considérés dans l'étape suivante).

4. pour chacun des voisins directs du routeur courant, on va traiter, pour chacune des destinations que ceux-ci peuvent atteindre, leurs différents chemins possibles vers ces destinations. Ces chemins ont été calculés lors de la seconde étape. Le traitement va se dérouler pour chaque voisin et pour chaque destination dans le domaine de la manière suivante :

   i. rechercher le coût entre le routeur courant et le voisin traité ;

   ii. récupérer le(s) chemin(s) allant de ce voisin vers la destination traitée ;

   iii. vérifier si le chemin courant n'existe pas déjà pour cette destination dans la Next Hop Structure. Il suffit ici de prendre la liste des chemins déjà connus (champ

path) de la ligne correspondante à la destination dans le champ *localNextHopStruct* du routeur courant et de faire une comparaison ;

iv. si le chemin n'existe pas dans la structure, et si le coût de ce chemin est strictement inférieur au coût du chemin calculé à la première étape entre le routeur courant et la destination traitée et si le coût du chemin augmenté du coût récupéré à l'étape 4.i est strictement supérieur au coût du chemin calculé à la première étape entre le routeur courant et la destination traitée[34], alors ce chemin peut être ajouté dans la liste des chemins de la destination traitée.

3) Adaptation de la next Hop Structure

L'adaptation de la *next Hop Structure* d'un routeur est réalisée par la méthode *adaptOneHop* de la classe *Router*. Plusieurs circonstances peuvent entraîner le besoin de modifier la *next Hop Structure* d'un routeur donné. Ces circonstances sont :

- les modifications topologiques des liaisons entre routeurs. Par modification, nous entendons l'ajout, la suppression d'une liaison ainsi que la modification du coût affecté à une liaison. Ces informations sont perçues directement si elle affecte la topologie locale du routeur (on appellera donc *adaptOneHop* dans les méthodes qui activent une liaison (*activateLink*), qui supprime une liaison (*removeInterface*) ou qui en modifie le coût (*modifyInterface*)). Ces informations peuvent aussi être perçues lors de la réception de messages de configuration LSA qui modifie une donnée topologique (on retrouvera donc aussi un appel à *adaptOneHop* dans la méthode *updateDb*) ;
- la suppression d'un LSA de la base de données car il dépasse la limite de vie maximale ;
- le démarrage du domaine pour remplir la *next Hop Structure* de tous les routeurs démarrés ;
- la suppression physique d'un routeur. Cette dernière circonstance est purement technique pour le simulateur. Elle permet uniquement d'adapter la taille des différents tableaux car la suppression d'un routeur ne donne lieu à aucun message de configuration.

La figure 32 donne un aperçu de l'algorithme d'adaptation de la *next Hop Structure*.

[34] Il est impossible d'avoir un coût inférieur car dans ce cas, ce coût inférieur serait le coût minimal en faisant abstraction de OMP. Si on obtient l'égalité, alors, ce chemin existe déjà dans la structure car ce chemin correspond à un des chemins minimaux déjà calculés au point 3.ii.

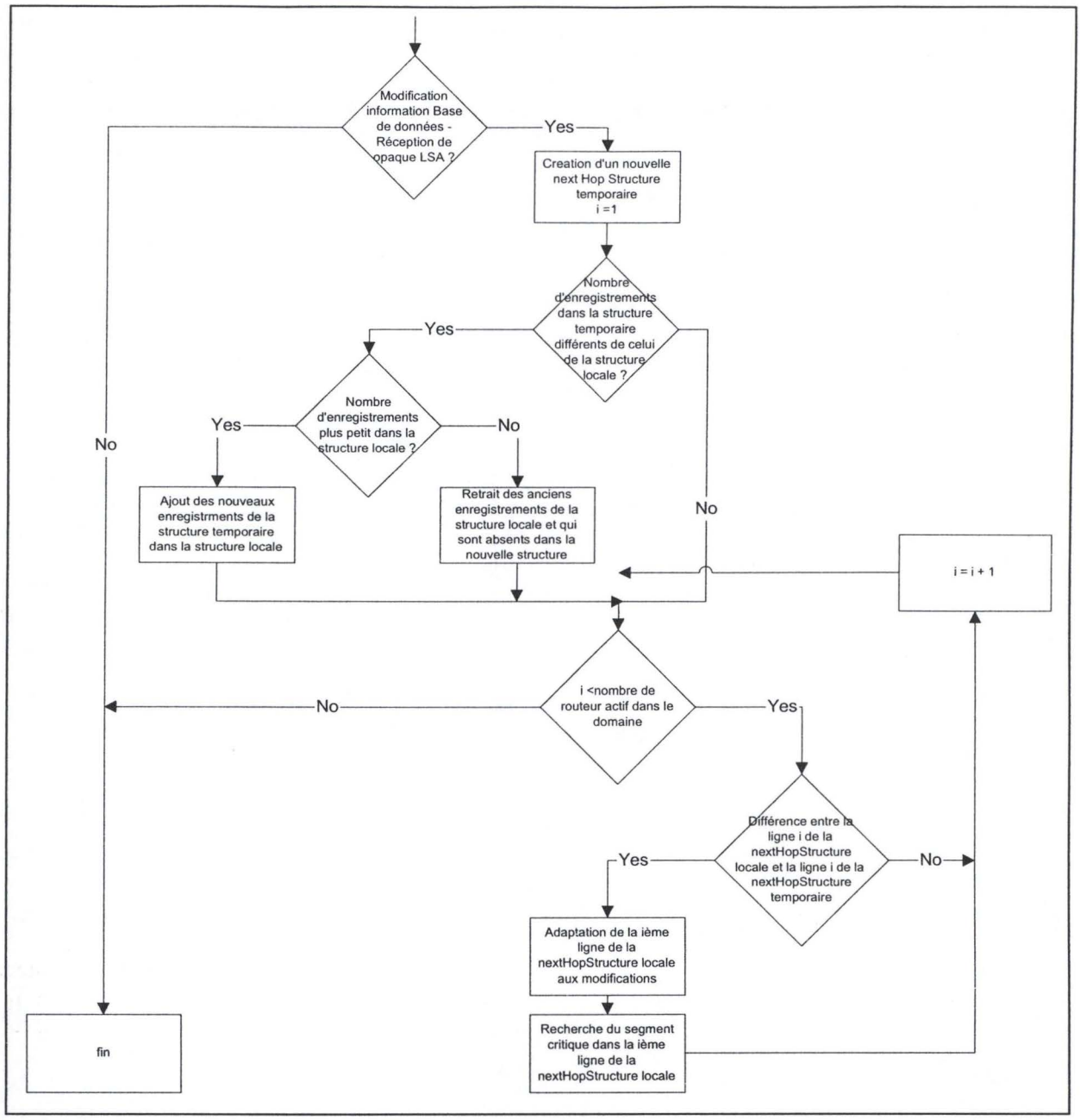


Figure 32 : adaptation de la *next Hop Structure*

- une Next Hop Structure temporaire est créée avec la méthode *createNextHopStructure* de la classe *Router*. Cette nouvelle structure contient tous les chemins « à jour » vers toutes les destinations mais ne possèdent pas d'informations historiques . A contrario, la *localNextHopStructure* du routeur courant n'est pas à jour pour ses chemins mais elle possède l'information historique. Il est donc nécessaire de faire une fusion entre les deux.
- pour chaque destination joignable, les lignes correspondantes des deux Next Hop Structure vont être comparées et la *localNextHopStructure* du routeur courant va être adaptée en fonction de l'apparition ou de la disparition de chemins dans la structure temporaire. En ce qui concerne la répartition du trafic :
    - si le domaine n'a aucune charge, alors, le trafic est réparti équitablement entre tous les chemins ;

- si le domaine est sous charge, alors, si on supprime un chemin, son trafic sera réparti proportionnellement entre tous les autres ; si on ajoute un chemin, son trafic initial sera de 0 ;

- si la *localNextHopStructure* du routeur courant a été modifiée pour une destination, alors il faut également effectuer une nouvelle recherche du segment critique. Cette recherche est réalisée par la méthode *criticalSegmentSearch* de la classe *NextHopStruct*. Le principe est le suivant :

  pour chaque liaison utilisée pour rejoindre la destination courante, on va déterminer, en consultant les informations contenues dans les opaque LSA de la base de données, la charge équivalente de cette liaison. Parmi toutes celles consultées, on ne conservera que celle(s) qui a (ont) la charge la plus élevée par rapport à toutes les autres. Cette liste sera placée dans le champ *listOfCriticalSegment* de la ligne du champ *localNextHopStructure* correspondant à la destination traitée.

b. Gestion de l'ajustement

1) Informations gérées sur les liaisons

Dans la pratique, OMP prévoit que chaque routeur évalue régulièrement le nombre de paquets lus et écrits sur une interface ainsi que la quantité d'informations perdues en entrée et en sortie. Toutes ces informations permettent de calculer la charge équivalente d'une interface (on calcule en fait la charge équivalente en entrée et en sortie et on ne retient que la donnée la plus élevée des deux).
En ce qui concerne le simulateur, certains choix ont dû être opérés. Ils ont été guidés par le fait que les spécifications demandent, en partant d'une matrice qui donne pour chaque routeur une quantité de trafic vers chacune des destinations du domaine, la charge sur chacune des liaisons. La simulation des variables SNMP en entrée et en sortie est donc impossible à réaliser ; aussi les variables d'entrée et de sortie ont été fusionnées en une seule et même variable. Ainsi, une liaison sera décrite par deux variables supplémentaires (on ajoute deux champs supplémentaires dans la classe *PointToPointLiaison*) :

1) *linkLoading* (double) qui donne la charge du réseau et qui peut prendre une valeur allant de 0.0 à 100.0 ;
2) *linkDrop* (double) qui donne la partie du trafic perdue pour cette liaison. Sa valeur est supérieure à 0.0 lorsque la variable *linkLoading* vaut 100.0. Par valeur perdue, il faut entendre la quantité d'information que le routeur ne sait pas envoyer sur une liaison car elle est saturée[35].

2) Opaque LSA

L'information décrivant la charge d'une liaison est diffusée dans le réseau à l'aide des opaque LSA. La diffusion de ceux-ci repose sur le principe d'inondation exposé dans OSPF. Les opaque LSA sont placés dans la base de données locale des routeurs. Outre le contenu, deux différences sont à noter par rapport à la gestion classique des LSA :

1) un opaque LSA est envoyé pour la description de chaque liaison tandis qu'un seul LSA suffisait pour décrire toutes les liaisons d'un routeur ;
2) la réception d'un opaque LSA ne cause pas uniquement un éventuel ajout dans la base de données (et une rediffusion) mais implique également, si nécessaire, une adaptation de la répartition du trafic au niveau de ce routeur.

En ce qui concerne le contenu, la classe *OpaqueLSA*, descendant direct de la classe *LSA*, indique, pour une liaison, la valeur du *linkLoading* et du *linkDrop*.

[35] Afin de simplifier le modèle de calcul du trafic à partir de la matrice d'envoi, le débit de toutes les lignes du domaine est considéré comme fixe et identique. Celui-ci ne sera dès lors pas pris en compte.

3) Gestion de la répartition du trafic

Un nouveau calcul de la répartition du trafic doit être réalisé quand une des deux conditions suivantes est réalisée :

1) si le routeur courant reçoit un opaque LSA qui décrit une liaison considérée comme critique pour ce routeur. Cette détection est réalisée dans la méthode *updateDb* de la classe *Router* (cette méthode dans le simulateur est responsable en général du traitement d'un LSA reçu via le buffer d'entrée) par la méthode *containCriticalSegment* de la classe *NextHopStruct*. On ne réadaptera que les lignes (c'est à dire les destinations) de la *next Hop Structure* pour lesquelles on retrouve cette liaison critique dans la liste des segments critiques. Il faut aussi noter que la réception d'un opaque LSA engendre automatiquement un nouveau calcul des segments critiques de ce routeur pour toutes les destinations ;
2) si l'évaluation réalisée par la méthode *checkForReadjust* de la classe *NextHopStruct* est positive. Toutes les 15 secondes, chaque ligne de la *next Hop Structure* d'un routeur est évaluée par la méthode précitée. Elle porte sur trois critères à savoir le temps écoulé depuis la dernière modification, la charge équivalente du segment critique et la différence entre la charge équivalente la plus élevée et la moins élevée. La figure 33 donne le pseudo-code de prise de décision du réajustement.

```
LastReadJust            //Date du dernier ajustement
LightestEquivLoad       //Charge équivalente la plus petite de tous les segments vers la
                         destination
MaxEquivLoad            //Charge du(des) segment(s) critique(s)

Elapsed := Now – LastReadJust
AbsDiff := MaxEquivLoad - LightestEquivLoad

Si ((Elapsed >= 60 Et AbsDiff > 4.5 Et MaxEquivLoad > 95) Ou
  (Elapsed >= 90 Et AbsDiff > 3 Et MaxEquivLoad > 95) Ou
  (Elapsed >= 120 Et AbsDiff > 1 Et MaxEquivLoad > 97) Ou
  (Elapsed >= 240 Et AbsDiff > 0.5 Et MaxEquivLoad > 98) Ou
  (Elapsed >= 90 Et AbsDiff > 5 Et MaxEquivLoad > 90) Ou
  (Elapsed >= 120 Et AbsDiff > 3 Et MaxEquivLoad > 90) Ou
  (Elapsed >= 180 Et AbsDiff > 1 Et MaxEquivLoad > 90) Ou
  (Elapsed >= 300)) alors
      /* il faut réajuster ce chemin*/...
```

Figure 33 : Pseudo-Code de détermination du réajustement [VIL99 p . 25]

L'adaptation de la charge affectée aux différents chemins d'une destination donnée se déroule en plusieurs étapes :

1. recherche du(des) segment(s) critique(s) dans l'ensemble des chemins avec la méthode *criticalSegmentSearch* de la classe *NextHopStruct*. Cette étape n'est pas réalisée dans le cas où l'adaptation aurait été demandée suite à la réception d'un opaque LSA critique ;

2. pour chaque chemin, on va déterminer s'il possède le(s) nouveau(x) segment(s) critique(s) et s'il possédait un des anciens segments critiques[36]. On va ainsi, en fonction de cette détermination, affecter une valeur booléenne aux champs *segment critique courant - hasCriticalSegment* et *segment critique précédent - hasPreviousCriticalSeg* des instances de *SetOfPath* représentant les chemins. Par la même occasion, on va rechercher la valeur du plus petit *move increment* parmi tous les chemins possédant le nouveau segment

[36] On conserve ici la trace que le chemin possédait le segment critique avant que ce dernier ne soit remplacé par un autre. Cela permet d'affecter une valeur d'incrémentation très petite à ce chemin de telle manière à éviter qu'un accroissement trop rapide de sa proportion de trafic ne le rende à nouveau critique.

critique. Le calcul pour un chemin est réalisé par la méthode *adaptCriticalPath* de la classe *SetOfPath*. L'ensemble des chemins vers une destination est traité par la méthode *adjustLoadingFirstPass* de la classe *NextHopStruct* ;

3. pour chaque chemin, la valeur de son champ *valeur de l'incrément - moveIncrement* va être adaptée. Plusieurs cas, basés sur les deux variables booléennes déterminées au point précédent, existent :

   - possession d'un segment critique auquel cas sa valeur reste inchangée ;
   - non-possession du segment critique précédent auquel cas la valeur de l'accroissement de son incrément dépend du nombre de fois qu'il a déjà été successivement augmenté et du nombre de chemins possédant le(s) segment(s) critique(s) pour cette destination ;
   - possession du segment critique précédent auquel cas l'incrément est mis à la valeur du plus petit incrément parmi tous les chemins possédant le segment critique.

La méthode *adaptMoveIncrement* de la classe *SetOfPath* dont le pseudo-code est représenté à la figure 34 réalise ce calcul pour un chemin. L'ensemble des chemins d'une destination est géré par la méthode *pathIncrementAdaptation* de la classe *NextHopStruct* ;

```
HasCriticalSegment          // indique que le chemin traité possède le segment critique
HasPreviousSegmentCritique  // indique que le chemin traité possédait le segment critique
                               précédent
MoveCount                   // nombre de fois que le chemin a vu son incrément augmenté
NumberWithCritical          // nombre de chemins vers une destination possédant le segment
                               critique
MinRateWithCritical         // l'incrément le plus petit parmi tous les chemins possédant le
                               segment critique
MoveIncrement               // valeur de l'incrément

Si (non HasCriticalSegment) alors
   {
      Si (non HasPreviousCriticalSegment) alors
         {
            ++MoveCount
            Si (MoveCount > 4) alors
               {
                  MoveIncrement := MoveIncrement + MoveIncrement / 2*(1+NumberWithCritical)
               }
            Sinon
               {
                  MoveIncrement := MoveIncrement + MoveIncrement / 4*(1+NumberWithCritical)
               }
         }
      Sinon
         {
            Si (MoveIncrement > MinRateWithCritical) alors
               {
                  MoveIncrement := MinRateWithCritical
               }
            MoveIncrement := MoveIncrement / 2
            MoveCount := 0
         }
   }
```

Figure 34 : pseudo-code du calcul de la valeur de l'incrément [VIL99 p.27]

4. la proportion de trafic affectée à chacun des chemins vers une destination donnée est adaptée. Le principe général est le suivant pour une destination : Si on a X chemins critiques, pour chaque chemin A qui possède le segment critique, on va prendre chaque chemin B qui ne possède pas le chemin critique et on va augmenter la proportion du trafic sur le chemin B de la valeur de son *moveIncrement* divisée par X tandis qu'on va diminuer de la même quantité le trafic sur le chemin A. De cette manière, tout accroissement du trafic d'un chemin est directement compensé par une perte équivalente sur un autre chemin, ce qui permet de conserver un équilibre. La méthode *trafficAdjust* de la classe *NextHopStruct* gère le calcul de la répartition du trafic sur les différents chemins. La figure 35 donne le pseudo-code de cette répartition ;

```
Path1                  // un chemin parmi les chemins existants pour une destination
Path2                  // un chemin parmi les chemins existants pour une destination
NumberWithCritical     // nombre de chemins possédant le segment critique
HasCriticalSegment     // indique que le chemin donné possède le segment critique
MoveIncrement          // valeur de l'incrément d'un chemin donné
TrafficShare           // proportion de trafic affectée à un chemin donné

Pour Chaque Path1 Faire
  {
    Si (Path1. HasCriticalSegment) alors
      {
        Pour Chaque Path2 Faire
          {
            Si (non Path2.HasCriticalSegment) alors
              {                          /* on détermine la valeur de l'incrément en
                                         fonction du nombre de chemins qui possède
                                         le segment critique de telle manière à ce que
                                         l'incrément maximal d'un chemin soit
                                         finalement la valeur de sa variable
                                         « moveIncrement »*/

                Move := Path2.MoveIncrement / NumberWithCritical
                Si (Move > (65536 – path2.TrafficShare)) alors
                  {
                    Move := 65536 – path2.TrafficShare
                    Path2.moveIncrement = Move
                  }
                Si (Move > Path1.TrafficShare) alors
                  {
                    Move := Path1.TrafficShare
                  }
                Path2.trafficShare := Path2.trafficShare + move
                Path1.trafficShare := Path1.trafficShare – move
              }
          }
      }
  }
```

Figure 35 : Pseudo-code de la répartition du trafic [VIL99 p. 27-28]

5. lorsque le trafic de tous les chemins vers toutes les destinations a été redistribué, la charge globale du domaine peut être recalculée en fonction des nouvelles répartitions. Le calcul est assuré par la méthode *organizeTrafficWithAdjust* de la classe *LinkLoading* ;

6. la modification de la charge du réseau calculée au point précédent entraîne le besoin de recalculer pour chacune des interfaces du routeur courant sa nouvelle charge équivalente. Ce calcul se base sur les informations possédées par une liaison concernant sa charge et ses pertes. Le calcul de la charge équivalente est réalisé par la méthode

calculateNewEquivLoad de la classe *PointToPointLiaison*. La figure 36 en donne le pseudo-code ;

```
RawUtil          // charge actuelle de la liaison entre 0.0 et 100.0
Loss             // trafic perdu sur la liaison
EquivLoad        // charge équivalente de la liaison
PrevEquivLoad    // charge équivalente précédente de la liaison

PrevEquivLoad := EquivLoad
Si (Loss < 0 .5) alors
   {
      EquivLoad := RawUtil
   }
sinon
   {
      Si (Loss <= 9.0) alors
         {
            LossComp = 10 * √Loss
         }
      sinon
         {
            LossComp = 3
         }
      EquivLoad := RawUtil * LossComp
   }
```

Figure 36 : Pseudo-Code du calcul de la charge équivalente [VIL99 p. 22]

7. les modifications apportées à la charge équivalente d'une liaison donnée doivent être diffusées dans tout le domaine avec un opaque LSA de manière à ce que les autres routeurs puissent être informés des modifications. Cependant, pour éviter une trop grande diffusion de messages dont l'impact serait peu significatif, les modifications ne seront diffusées que lorsqu'un critère mélangeant la charge équivalente, son évolution et le temps écoulé depuis la dernière diffusion sera rencontré. La méthode *checkIfReflood* de la classe *PointToPointLiaison* gère la prise de décision d'envoi d'un opaque LSA. La diffusion est semblable à celle des LSA classiques. La figure 37 donne le pseudo-code de la prise de décision.

```
EquivLoad          // charge équivalente de la liaison
PrevEquivLoad      // charge équivalente précédente de la liaison
LastReflood        // date du dernier envoi d'un Opaque LSA concernant la liaison courante

Diff := max (EquivLoad – PrevEquivLoad) / PrevEquivLoad
Elapsed := Now - LastReflood
Si (( EquivLoad > 100) Et (Diff > 0.05) Et (Elapsed >=30) Ou
   ( EquivLoad > 100) Et (Diff > 0.02) Et (Elapsed >=60) Ou
   ( EquivLoad > 100) Et (Diff > 0.01) Et (Elapsed >=90) Ou
   ( EquivLoad > 100) Et (Elapsed >=180) Ou
   ( EquivLoad > 90) Et (Diff > 0.05) Et (Elapsed >=60) Ou
   ( EquivLoad > 90) Et (Diff > 0.02) Et (Elapsed >=240) Ou
   ( EquivLoad > 90) Et (Diff > 0.01) Et (Elapsed >=480) Ou
   ( EquivLoad > 90) Et (Elapsed >=600) Ou
   ( EquivLoad > 70) Et (Diff > 0.10) Et (Elapsed >=60) Ou
   ( EquivLoad > 70) Et (Diff > 0.05) Et (Elapsed >=120) Ou
   ( EquivLoad > 70) Et (Diff > 0.02) Et (Elapsed >=480) Ou
   ( EquivLoad > 70) Et (Elapsed >=900) Ou
   ( EquivLoad > 50) Et (Diff > 0.10) Et (Elapsed >=60) Ou
   ( EquivLoad > 50) Et (Diff > 0.05) Et (Elapsed >=300) Ou
   ( EquivLoad > 25) Et (Diff > 0.25) Et (Elapsed >=120) Ou
   ( EquivLoad > 25) Et (Elapsed >=1200))
   /* Envoi d'un Opaque LSA décrivant la liaison */...
```

Figure 37 : Prise de décision d'envoi d'un Opaque LSA [VIL99 p.23]

c. *Calcul du trafic*

La classe *LinkLoading* est responsable de la gestion du trafic au sein du domaine.

L'utilisateur donne, via l'interface, la quantité de trafic qu'un routeur génère à destination de chacun des autres routeurs du domaine. La somme de ces quantités pour un routeur ne peut jamais excéder 100. Toutes ces données sont stockées dans un tableau *initialTraffic* décrit dans la classe *LinkLoading*, chaque case [i] [j] représentant la quantité de trafic que le routeur i génère et envoie vers le routeur j.

A partir des données précitées, il est possible de calculer deux répartitions différentes de la charge sur les liaisons du domaine :

1) une charge basée uniquement sur OSPF. Dans ce cas, le trafic est réparti équitablement entre les différents chemins (c'est à dire uniquement les chemins de coûts minimaux) pour une destination. La méthode *organizeTrafficCritic* de la classe *LinkLoading* en est responsable ;
2) une charge basée sur OMP. Dans ce cas, le trafic est réparti entre les différents chemins (c'est à dire les chemins découverts par OMP) pour une destination suivant les proportions déterminées par les algorithmes de OMP. La méthode *organizeTrafficWithAdjust* de la classe *LinkLoading* en est responsable.

L'algorithme qui calcule la répartition est fort semblable pour les deux méthodes. Les seules différences portent sur le nombre de chemins et la répartition du trafic entre ceux-ci. Dans ce qui suit, nous allons expliquer cet algorithme de répartition. La figure 38 en donne une version en pseudo-code.

```
Tcourant [i][j]          // tableau à double entrée indiquant le trafic circulant sur une liaison physique entre
                            deux routeurs. –1 pour une case [i][j] sur le tableau indique l'absence de liaison
                            physique entre les routeurs. Une valeur égale ou supérieure à 0.0 pour une case [i][j]
                            indique l'existence d'une liaison entre les routeurs i et j ainsi que la charge de cette
                            liaison.
Tinitial [i][j]          // tableau à double entrée indiquant le trafic généré par un routeur i à destination d'un
                            routeur j.
Tprec [i][j]             // tableau à double entrée indiquant le trafic reçu par un routeur i et qu'il doit envoyer
                            à un routeur j. Ce trafic correspond à la somme de tous les trafics pondérés générés
                            par tous les routeurs à l'exception de i, à destination de j et dont au moins un chemin
                            passe par i. On utilise le terme pondéré pour indiquer la proportion de trafic vers une
                            destination dédicacée à un chemin particulier.
NoModif                  // booléen indiquant l'état de stabilité de la charge des lignes. Sa valeur est initialisée
                            à true.
tempTraffic              // tableau temporaire pour vérifier la stabilité du trafic dans le domaine.
Chemin                   // liste des chemins existants entre deux routeurs.
chemin(k).trafficShare   // proportion du trafic accordé au chemin (k).
chemin(k).routeurSuivant // routeur qui suit directement le routeur source du chemin (k).

Tant que (noModif)

{
    noModif=false
    tempTraffic : = Tcourant
    Tcourant : = 0.0      // On initialise le tableau Tcourant en mettant tous les trafics sur les lignes à la valeur 0.0
    Pour chaque routeur (i) Faire      // On traite chaque routeur du domaine

    {
        // Cette première partie de l'algorithme traite le trafic sur les liaisons sortant du routeur (i)

        Pour chaque routeur (j) Faire      // On va regarder chaque routeur (j) du domaine en fonction du routeur
                                              (i) précédemment récupéré.
                                           // On peut ici considérer que le routeur (i) sera la source des chemins
                                              qu'on va traiter et que le routeur (j) en sera la destination.
        {
            chemin : = liste des chemins de i à j en tenant compte des critères de chemins de omp
            Pour chaque chemin (k) Faire      // On traite individuellement chaque chemin(k) allant du routeur
                                                 (i) au routeur (j).
            {
                Proportion : = chemin(k).trafficShare      // On détermine la proportion de trafic accordée
                                                              au chemin (k) traité.
                Suivant : = chemin(k).routeurSuivant       // On détermine le routeur qui suit le routeur (i)
                                                              sur le chemin (k).

                Tcourant[i][suivant] : = Tcourant[i][suivant] + (Tinitial[i][j] + Tprec[i][j]) / Proportion

                                                   // Le supplément de trafic engendré par le chemin (k)
                                                      actuellement traité sur la liaison unissant le
                                                      routeur(i) au routeur Suivant équivaut à la somme
                                                      pondérée du trafic généré par le routeur (i) à
                                                      destination du routeur (j) et du trafic que le routeur
                                                      (i) reçoit d'un autre routeur (voir deuxième partie
                                                      de l'algorithme) à destination du routeur (j). La
                                                      pondération est la proportion de trafic accordée au
                                                      chemin (k) traité.
            }
        }

    // Cette seconde partie de l'algorithme traite le trafic entrant chez les routeurs auxquels le routeur (i) est connecté

        Pour chaque routeur (j) Faire      // On va regarder chaque routeur (j) du domaine en fonction du routeur
                                              (i) précédemment récupéré.
```

```
                                        // On peut ici considérer que le routeur (i) sera la source des chemins
                                           qu'on va traiter et que le routeur (j) en sera la destination.
        {
            chemin : = liste des chemins de i à j en tenant compte des critères de chemins de omp

            Pour chaque chemin (k) Faire    // On traite individuellement chaque chemin(k) allant du routeur
                                               (i) au routeur (j).
            {
                Proportion : = chemin(k).trafficShare    // On détermine la proportion de trafic accordée au
                                                            chemin (k) traité.
                Suivant : = chemin(k).routeurSuivant     // On détermine le routeur qui suit le routeur (i) sur le
                                                            chemin (k).

                Si (Routeur(j) != Suivant) alors
                                        // Si le routeur (Suivant) qui suit le routeur (i) sur le chemin (k)
                                           n'est pas la destination de ce chemin, alors ce routeur (Suivant)
                                           devra propager le trafic que le routeur (i) lui envoie, et cela vers
                                           le routeur (j). Ce trafic à propager sera inscrit dans le tableau
                                           Tprec à la position [Suivant][j]. Par contre, si le routeur
                                           (Suivant) est la destination du chemin, alors le routeur
                                           (Suivant) ne devra pas propager le trafic mais au contraire, il
                                           l'absorbera.
                {
                    Si (Tcourant[i][Suivant] > 100.0) alors
                                                // Si la liaison entre le routeur (i) et le routeur
                                                   (Suivant) connaît des pertes, alors, ces pertes sont à
                                                   répercuter sur la quantité de trafic que le routeur (i)
                                                   fournira effectivement au routeur (Suivant).
                    {
                        Proportion : = Proportion * (100 / Tcourant[i][Suivant])
                    }

                    Tprec[Suivant][j] := Tprec[Suivant][j] + ((Tinitial[i][j] + Tprec[i][j]) * Proportion
                }
            }

            Tprec [i][Suivant] : = 0.0

            // Le trafic « en transit » entre le routeur (i) et le routeur (Suivant) a été distribué. Il peut dès lors être
               remis à zéro.
        }
    }

    // La troisième partie vérifie si le trafic courant calculé après ce passage dans l'itération tant que est le même que
       celui calculé au tour précédent. Si c'est le cas, on sort de l'algorithme sinon, on fait un nouveau passage.

    Si (tempTraffic = = Tcourant) alors
    {
        noModif := false
    }
    sinon
    {
        noModif := true
    }
}
```

Figure 38 : algorithme du calcul de la charge du domaine

On peut considérer que cet algorithme se déroule en trois étapes distinctes. Ces 3 étapes seront répétées jusqu'à stabilisation :

1. Cette étape sert à déterminer la charge affectée aux liaisons (trafic sortant d'un routeur). Pour chaque routeur, on va analyser chacune des destinations qu'il peut atteindre. Pour chacune de ces destinations, on va analyser chacun des chemins possibles :

   - rechercher le routeur suivant sur ce chemin vers la destination ;
   - déterminer la proportion du trafic qui est affecté à ce chemin ;
   - déterminer la charge de la liaison joignant le routeur courant au routeur suivant en sommant la charge déjà existante sur la liaison avec une proportion (en fonction du chemin – point précédent) du trafic (*Tinitial*)[37] généré au niveau du routeur courant et avec une proportion (en fonction du chemin – point précédent) du trafic (*Tprec*)[38] transitant par le routeur courant pour joindre la destination.

2. Cette étape sert à déterminer le trafic supplémentaire que doivent supporter les routeurs qui font partie d'un chemin (trafic entrant dans un routeur). Pour chaque routeur, on va analyser chacune des destinations qu'il peut atteindre. Pour chacune de ces destinations, on va analyser chacun des chemins possibles :

   - rechercher le routeur suivant sur ce chemin vers la destination ;
   - déterminer la proportion du trafic qui est affecté à ce chemin ;
   - déterminer la quantité de trafic entrant que le routeur suivant sur ce chemin devra assumer. Trois cas peuvent se présenter :

     - si le routeur suivant est la destination du chemin, alors, il n'y a aucun report de trafic (le trafic est en fait absorbé par le routeur) ;
     - si la liaison unissant le routeur courant avec le suivant a une charge inférieure ou égale à 100 %, alors, le trafic supplémentaire du routeur suivant sera égal au trafic supplémentaire existant du routeur suivant auquel on ajoute la somme proportionnelle (la proportion est déterminée au point précédent) du trafic initial et du trafic supplémentaire du routeur courant vers la destination.
     - si la liaison à une charge supérieure à 100 %, alors, la quantité de trafic supplémentaire pour le routeur suivant est calculée de la même manière que pour le point précédent à la différence près que la somme proportionnelle sera réduite d'un coefficient de perte[39] car une partie du trafic sur cette liaison est perdue.

   - le trafic supplémentaire qui a été réparti parmi tous les suivants d'un routeur est remis à 0.

3. Cette étape détermine la stabilité de la charge du domaine. Si on a une stabilité, on peut fournir une charge correcte des différentes liaisons du domaine. Si on n'a pas de stabilité (c'est à dire que les charges calculées durant ce tour sont différentes de celles calculées au tour précédent), alors il faut recommencer le calcul mais en conservant le trafic supplémentaire qui n'a pas été remis à 0. De cette manière, ce trafic pourra au tour suivant être reparti de manière correcte. On pourra considérer qu'un domaine est stable lorsqu'on aura plus de trafic supplémentaire.
   On peut montrer de manière informelle que cet algorithme arrivera à une stabilité en un temps fini pour autant que :

---

[37] Le trafic initial est celui qui est donné par l'utilisateur via l'interface.

[38] Le trafic résiduel est le trafic qui doit transiter par le routeur courant pour rejoindre la destination. Il est calculé dans la seconde partie de l'algorithme. Par exemple, si le chemin entre les routeurs A et B transite par C et que l'utilisateur donne un trafic initial de A à B ayant une valeur de 40, il est alors évident que le trafic sera de 40 sur la liaison A-C et également de 40 sur la liaison C-B. Au niveau du routeur C, la valeur du trafic initial de C vers B sera de 0 tandis que la valeur du trafic résiduel de C vers B sera fixée à 40.

[39] Rapport entre la charge de la ligne et 100.

- *Tinitial* ne varie pas durant l'exécution de l'algorithme ;
- tous les chemins fournis par OMP sont sans boucle.

En effet, on vérifiera la stabilité de l'algorithme lorsque deux passages successifs ne modifieront pas les valeurs dans le tableau *Tcourant* (tableau qui donne le trafic sur les liaisons). *Tcourant* est fonction de *Tinitial* (on peut le négliger ici car il est considéré comme constant) et de *Tprec*. Dans l'algorithme, *Tprec* sert à montrer la circulation du trafic entre la source (où il est généré à partir de *Tinitial*) et la destination (où il est absorbé). Entre les deux, le *Tprec* qu'un routeur communique à son voisin équivaut à tout le trafic qu'il doit envoyer vers la destination (pour le chemin considéré) c'est à dire son trafic initial (celui qu'il génère) et celui qu'il reçoit du routeur qui le précède (pour le chemin). Comme le trafic initial est considéré comme une constante et que tout trafic généré sera absorbé étant donné qu'aucun chemin ne boucle, *Tprec* arrivera à une stabilité (cette stabilité sera obtenue lorsque tout le trafic généré sera absorbé – en une itération, tout le trafic *Tprec* existant recevra la valeur 0). Comme le tableau est parcouru de manière séquentielle, le nombre de fois que l'itération devra être parcourue correspondra au nombre de tronçons en sens inverse[40] du chemin ayant le plus grand nombre de tronçons inverses auquel il faut ajouter un passage pour le traitement en sens normal et un passage pour montrer la stabilisation.

Lorsque le trafic a été calculé, il faut modifier les charges et les pertes des différentes liaisons du domaine. Cela est réalisé par la méthode *giveLinkCost* de la classe *LinkLoading* qui modifie les champs décrivant la charge des liaisons.

d. Modifications dues à OMP dans la dynamique du domaine et des routeurs

La classe *Domain* reçoit un champ supplémentaire *currentLoad* qui est une instance de la classe *LinkLoading* et qui reprend les tableaux décrivant le trafic donné par l'utilisateur, la charge du réseau en appliquant OMP et la charge du réseau classique.

La classe *Domain* se voit également ajouter une variable *localTime* qui représente l'écoulement du temps à l'intérieur du simulateur. Cette variable est nécessaire car il n'est pas possible de se reposer sur le temps fourni par le système étant donné que l'évolution d'un domaine n'est pas constante (on peut figer un domaine ou accélérer son fonctionnement). La gestion de cette variable est semblable à la gestion du vieillissement des LSA.[41]

Il faut noter que certaines méthodes des routeurs se sont vues ajouter certains composants propres à OMP. Ces modifications ont déjà été commentées précédemment.

Le démarrage d'un routeur est complexifié :

- chaque nouveau routeur ajouté à un domaine déjà en fonctionnement doit entraîner l'adaptation des tableaux de description des charges (Ces tableaux doivent désormais tenir compte de son existence). Cela est réalisé par la méthode *modified* appliquée sur le champ *currentLoad* de la classe *Domain* (cette méthode a pour but d'élargir les différents tableaux d'une ligne et d'une colonne supplémentaire) ;
- création de la nextHopStructure de ce routeur ;
- avant de commencer l'analyse de leur buffer d'entrée, tous les routeurs au démarrage d'un domaine sont bloqués. Ce blocage est mis en place de manière à synchroniser les différents routeurs. Il est réalisé avec le champ *amorce* de la classe *Domain*. Ce champ *amorce* est une

[40] Considérons un tableau constitué par les routeurs A, B, C, D. Dans l'algorithme, le routeur (i) sera successivement A puis B puis C et finalement D. Avec un tel parcours, le tronçon B – C sera considéré comme étant dans le bon sens tandis que le tronçon C – B sera considéré comme étant en sens inverse. Le report de trafic sur un tronçon en sens inverse sera pris en considération lors de la prochaine itération de l'algorithme.

[41] voir supra

instance de la classe *StartUp*. Cette classe attend que tous les routeurs soient bloqués puis, pour chacun d'eux, elle va adapter leur *nextHopStructure* via la méthode *adaptOneHop* puis créer la répartition initiale du trafic entre les différents chemins existants. Suite à cela, tous les routeurs pourront être libérés et continuer leur exécution normale.

La suppression d'un routeur entraîne la suppression d'une ligne dans la *nextHopStructure* des routeurs restants et un calcul d'une nouvelle répartition globale du trafic. Cette adaptation des routeurs a lieu durant leur suspension (en effet, la suppression d'un routeur a pour conséquence, entre autres, l'arrêt momentané du domaine). Cette pratique peut sembler artificielle mais elle reflète tout de même la réalité. En effet, la disparition d'un routeur implique des modifications topologiques qui se répercutent directement par une adaptation de tous les chemins et donc un nouveau calcul de la charge. En général, une modification topologique quelconque a pour conséquence de créer des perturbations dans le domaine. Après quelques temps, ce dernier retrouvera un nouveau point d'équilibre.

Conclusion

Un routeur dans un réseau (au sens large du terme) joue un rôle d'aiguilleur pour les paquets de données circulant entre les « sous-réseaux » que ce routeur interconnecte. A tout paquet de données entrant par une de ses interfaces, le routeur détermine l'interface de sortie la plus adéquate de telle manière à ce que ce paquet rejoigne dans les meilleures conditions possibles sa destination. Un paquet de données va donc circuler de routeurs en routeurs depuis sa source jusqu'à sa destination. Pour atteindre ce but, tous les routeurs participant à un même réseau doivent connaître la topologie de ce réseau.

OSPF est un protocole qui permet à un routeur d'une part, d'apprendre dynamiquement la topologie du réseau et d'autre part, de proposer pour chaque destination du réseau, le meilleur chemin à prendre en évitant tout bouclage. OSPF est utilisé dans le cadre du routage IP.

L'apprentissage de la topologie est essentiellement réalisé par un algorithme distribué et coopératif. Chaque routeur diffuse régulièrement sa configuration topologique locale (c'est à dire les liaisons du réseau dont il est à l'origine) à tous les autres routeurs du réseau. Chaque routeur est également à l'écoute permanente de tous les messages de configuration envoyés par tous les autres routeurs du réseau. La réception de tous ces messages permet à un routeur de posséder l'image complète, correcte et à jour du réseau. Tous les routeurs doivent posséder la même image.

En se basant sur la carte du réseau, chaque routeur peut, en utilisant l'algorithme S.P.F. (recherche du plus court chemin vers une destination dans un graphe), pour chaque paquet de données qu'il reçoit et pour lequel il connaît la destination, déterminer sans ambiguïté le routeur suivant vers lequel il doit diriger ce paquet pour que celui-ci atteigne sa destination de la meilleure façon. Comme tous les routeurs possèdent la même topologie et implémente le même algorithme S.P.F., on peut affirmer qu'ils collaborent entre eux pour permettre un transfert optimal des paquets de données à travers un réseau.

Ce mémoire a pour but initial de présenter, via l'implémentation d'un outil de simulation graphique, les mécanismes de base utilisés par OSPF. Ces mécanismes sont : la découverte par un routeur de sa topologie locale, la diffusion régulière de cette topologie locale, le traitement de la réception par un routeur de messages topologiques et la création d'une table récapitulative par routeur indiquant pour chaque destination, sur base de sa carte locale, le routeur qui suit directement le routeur local sur le chemin vers cette destination. Pour montrer l'évolution dynamique de OSPF, le simulateur permet, à tout moment de modifier les éléments topologiques (création et suppression de routeurs, création et suppression de liaisons, modification du poids associé à une liaison). Il offre également la possibilité de consulter les informations importantes utilisées par OSPF pour sa gestion.

La conception de ce simulateur, réalisée en JAVA (JDK1.2 de SUN), a été guidée par le principe suivant : **Respecter au maximum l'idée de décentralisation du concept OSPF de manière à calquer au mieux son fonctionnement** ; et ce parfois, il est vrai, au détriment d'une optimisation du fonctionnement de ce simulateur.

Partant de ce principe, le simulateur peut se résumer à un ensemble de processus indépendants en interaction avec l'environnement, chacun représentant un routeur. Tous ces processus ont la possibilité de communiquer entre eux pour s'échanger des messages (pour autant que deux routeurs en communication soient directement reliés par une liaison). Cette communication est abordée de manière défensive afin d'éviter des conflits propres à la concurrence entre processus. Chaque *processus-routeur* possède donc tous les moyens nécessaires pour atteindre les buts précités.

Plus précisément, chacun entretient un ensemble de structures de données qui reprennent :

- la description des liaisons dont le routeur est à l'origine et qui le connecte à un autre routeur du réseau ;
- la description de tous les messages topologiques les plus récents qu'il a reçu ;
- les éléments propres à la réception de messages venant d'autres routeurs.

Chaque *processus-routeur* est également capable de manipuler ses données locales pour obtenir une carte du réseau (sous la forme d'un graphe orienté) et en déduire, en utilisant l'algorithme de Dijsktra, une table de routage locale.

Finalement, un *processus-routeur* peut prendre un certain nombre d'initiatives (tout en respectant les prescriptions de OSPF) concernant la découverte de sa topologie locale et la diffusion régulière de celle-ci.

Tous les *processus-routeur* sont des « fils » du processus qui est responsable de l'exécution du simulateur à proprement parler. Le *processus-simulateur* représente dès lors la structure du réseau dans sa globalité. Ses fonctions se bornent à la gestion centralisée de l'ajout et de la suppression des routeurs (cela revient à ajouter ou supprimer des *processus-routeur*) ainsi qu'à la gestion de l'interface graphique de représentation.

L'utilisateur du simulateur n'interagit qu'avec l'interface de représentation. Il peut réaliser des modifications topologiques (celles-ci peuvent affecter le *processus-simulateur* dans le cas de la création ou de la suppression d'un routeur, et/ou peuvent indirectement affecter un processus-routeur dans le cas de modification de sa topologie locale), obtenir l'affichage de certaines informations contenues par un routeur particulier du réseau et modifier la vitesse d'exécution des relations entre les différents *processus-routeur*.

Le protocole OSPF n'utilise, pour déterminer les chemins, que le poids affecté à chacune des liaisons dans le réseau. Cette approche ne tient pas compte d'autres facteurs comme la charge du réseau. Or, les conséquences d'une surcharge de celle-ci peuvent être dramatiques au point de causer un blocage complet de tout le réseau. OMP (Optimized Multi Path) est une proposition d'amélioration à OSPF qui vise à tenir compte de la charge dans la gestion des chemins et l'envoi des paquets de données sur ces chemins. Elle aborde la problématique des chemins de manière beaucoup plus globale et dynamique que ne le fait OSPF. Cette amélioration comprend 3 grands points :

1) tout routeur du domaine doit être informé régulièrement de la charge de toutes les liaisons du réseau ;
2) le nombre de chemins possibles pour rejoindre une destination est augmenté en rendant l'algorithme SPF plus souple ;
3) chaque chemin vers une destination à partir d'une source va se voir affecter une proportion du trafic émis à partir de cette source en fonction de l'état de charge des différentes liaisons constituant ce chemin.

Le simulateur OSPF a été adapté de manière à prendre en compte les éléments décrits dans OMP tout en respectant le principe général de décentralisation. Cette extension au simulateur permet de montrer les mécanismes de fonctionnement de OMP ; elle peut également servir d'outil de recherche, d'étude du comportement d'un réseau implémentant OMP.

La conception de OMP dans le simulateur relève de deux grands axes : d'une part, la modification des *processus-routeur* afin de leur intégrer les mécanismes de OMP et d'autre part, la modification du *processus-simulateur* de manière à ce que celui-ci puisse également simuler du trafic dans le réseau.

Les *processus-routeur* se voient ajouter une nouvelle structure de données décrivant pour chaque destination dans le réseau, l'ensemble des chemins empruntables pour les rejoindre et l'importance accordée à chacun de ces chemins. Les routeurs ont également été dotés de deux nouveaux mécanismes permettant d'une part de diffuser dans le domaine, en réutilisant les mécanismes de diffusion proposés dans le simulateur de base, la charge des liaisons dont ils sont responsables et d'autre part d'analyser la charge globale du réseau reçue de par les diffusions dans le but de déterminer la répartition du trafic sur les différents chemins.

Le *processus-simulateur* se voit ajouter la tâche de détermination globale de la charge des liaisons du réseau. Sur base du trafic généré par les routeurs du domaine pour toutes les destinations qu'ils peuvent

joindre (ce trafic est déterminé par l'utilisateur du simulateur) et sur base des chemins donnés par les *processus-routeur* (et de la répartition du trafic entre eux), le *processus-simulateur* calcule la charge et les pertes de chacune des liaisons du réseau. Le résultat obtenu peut alors être exploité par les *processus-routeur* pour s'adapter en conséquence. Il est à noter que toute modification topologique ou de répartition de la charge entre deux chemins entraîne automatiquement un nouveau calcul de la charge du réseau. Ainsi, l'adaptation globale de la charge et l'adaptation de la répartition du trafic sur les chemins s'appelleront mutuellement jusqu'à l'obtention d'un point d'équilibre.

Le simulateur OSPF n'est qu'un point de départ. Il ne fait qu'implémenter les mécanismes de base de OSPF. Le protocole en lui-même contient un ensemble d'optimisations qui lui permette entre autres d'être gérer de manière plus souple lorsque le nombre de routeurs dans le réseau s'accroît. Beaucoup d'extensions ont été ajoutées ou proposées à OSPF ; OMP en fait partie ; le simulateur a été écrit de manière à faciliter l'ajout de nouvelles extensions.

OMP, quant à lui, n'est qu'un outil. Il peut être utilisé pour analyser et comprendre en détail le comportement des routeurs qui l'implémente, suivant certaines configurations de réseau et de charge. Cette partie du simulateur peut également être améliorée en lui incluant des modèles de comportement et d'évolution du trafic plus souple.

N'oublions jamais l'adage populaire : TOUS LES CHEMINS MENENT A ROME.

Bibliographie

1. OSPF et OMP

[MOY98a] John MOY, **OSPF Version 2**, RFC 2328, avril 1998.

[MOY98b] John MOY, **OSPF Anatomy of an Internet Routing Protocol**, Addison-Wesley, janvier 1998.

[HUI94] Christian HUITEMA, **Le routage dans l'Internet**, Eyrolles, octobre 1994.

[THO98] Thomas M. THOMAS, **OSPF Network Design Solutions**, Cisco Press, MacMillan, 1998.

[VIL99] Curtis VILLAMIZAR, **OSPF Optimized Multipath**, draft-ietf-ospf-omp-02, février 1999.

[MKR91] MCCLOGHRIE, K., M. ROSE, **Management Information Base for Network Management of TCP/IP – based internets : MIB-II**, RFC1213, Mars 1991.

[COL98] R. COLTUN, **The OSPF Opaque LSA Option**, RFC 2370, Juillet 1998.

[MSM97] M. MATHIS, J. SEMKE, J. MAHDAVI, T. OTT, **The macroscopic behavior of the TCP congestion avoidance algorithm**, ACM Communication Review, 27(3), Juillet 1997.

[AHU87] A. AHO, J. HOPCROFT, J. ULLMAN, **Structures de données et algorithmes**, InterEditions, 1987.

2. JAVA

[HCO99] Cay S. HORSTMANN, Gary CORNELL, **Au cœur de JAVA Volume 1 – Notions fondamentales**, CampusPress, 1999.

[ECK98] Bruce ECKEL, **Thinking in Java**, Prentice-Hall, 1998.

[MIR99] Antoine MIRECOURT, **Le développeur JAVA2 édition 1999**, Osman Eyrolles Multimédia, 1999.

[OWO00] Scott OAKS, Henry WONG, **Java Threads**, O'Reilly, 2000.

Annexe 1 – Manuel utilisateur

1. Table des matières

2. Table des figures

3. Introduction

Cette annexe décrit le fonctionnement du simulateur OPSF. Cette description est abordée de manière opérationnelle et systématique. On suppose que les concepts sous-jacents aux protocoles OSPF et OMP (chapitre 1 & 2 de ce travail) sont connus et maîtrisés par le lecteur de cette annexe ; aucune référence ni explications ne seront donc ici données.

Ce manuel utilisateur est divisé en 4 grandes sections :

1) la méthode pour installer et démarrer correctement le simulateur ;
2) la description générale de l'interface ;
3) la description des différents modes de fonctionnement ;
4) la description systématique de toutes les fonctionnalités du simulateur.

4. Installation et démarrage

Le simulateur OPSF a été écrit en JAVA 1.2. Cette version ne fonctionne que sous Microsoft Windows 95 ou 98 et Microsoft Windows NT 4.0. Dans ce qui suit, nous supposons que l'utilisateur a préalablement installé sur sa machine soit le JDK1.2, soit le JRE 1.2. Ceux-ci sont disponibles gratuitement chez http:\\.java.sun.com.

Pour une utilisation optimale du simulateur, il est vivement recommandé de copier le répertoire \OPSFcode de la disquette et son contenu sur le disque dur local.

Le simulateur peut être démarré en double-cliquant sur le fichier OSPF_Sim.jar situé dans le répertoire OSPFCode et copié sur le disque dur.

Il peut aussi être ouvert via une fenêtre DOS. La commande (les éléments en gras sont des éléments imposés – les éléments en italique sont fonction de la configuration locale de la machine) est :

DisqueJava:*cheminJava***java –jar** *DisqueApplic*:*cheminApplic*:**OSPFcode\OSPF_Sim.jar** *option*

Les différents éléments de cette commande sont

DisqueJava : indication du disque sur lequel l'interpréteur java se situe ;
cheminJava : indication du chemin pour atteindre l'interpréteur java ;
Habituellement, le chemin est dans le cas de JDK1.2 : **jdk1.2\bin**
le chemin est dans le cas de JRE1.2 : **progra~1\javaSoft\jre\1.2\bin**
DisqueApplic : indication du disque sur lequel se trouve le simulateur ;
cheminApplic : indication du chemin pour atteindre le simulateur ;
option : champ facultatif qui permet de choisir le type d'interface. Si sa valeur est 0, alors, l'interface du simulateur utilisera le modèle de fenêtre de Windows. Si une autre valeur est choisie, alors, l'interface du simulateur utilisera le modèle de fenêtre de JAVA.

Par exemple, si le répertoire \OSPFCode a été copié au niveau de la racine du lecteur C, qu'on utilise l'interpréteur JDK1.2 et qu'on veut un fenêtrage style « Windows », alors, la ligne de commande sera :

c:\jdk1.2\bin\java –jar c:\OSPFCode\OSPF_Sim.jar 0

Tout problème concernant le démarrage du simulateur et son fonctionnement peuvent être envoyés à l'adresse mail suivante : chr.gaussin@village.uunet.be

5. Description Générale de l'interface

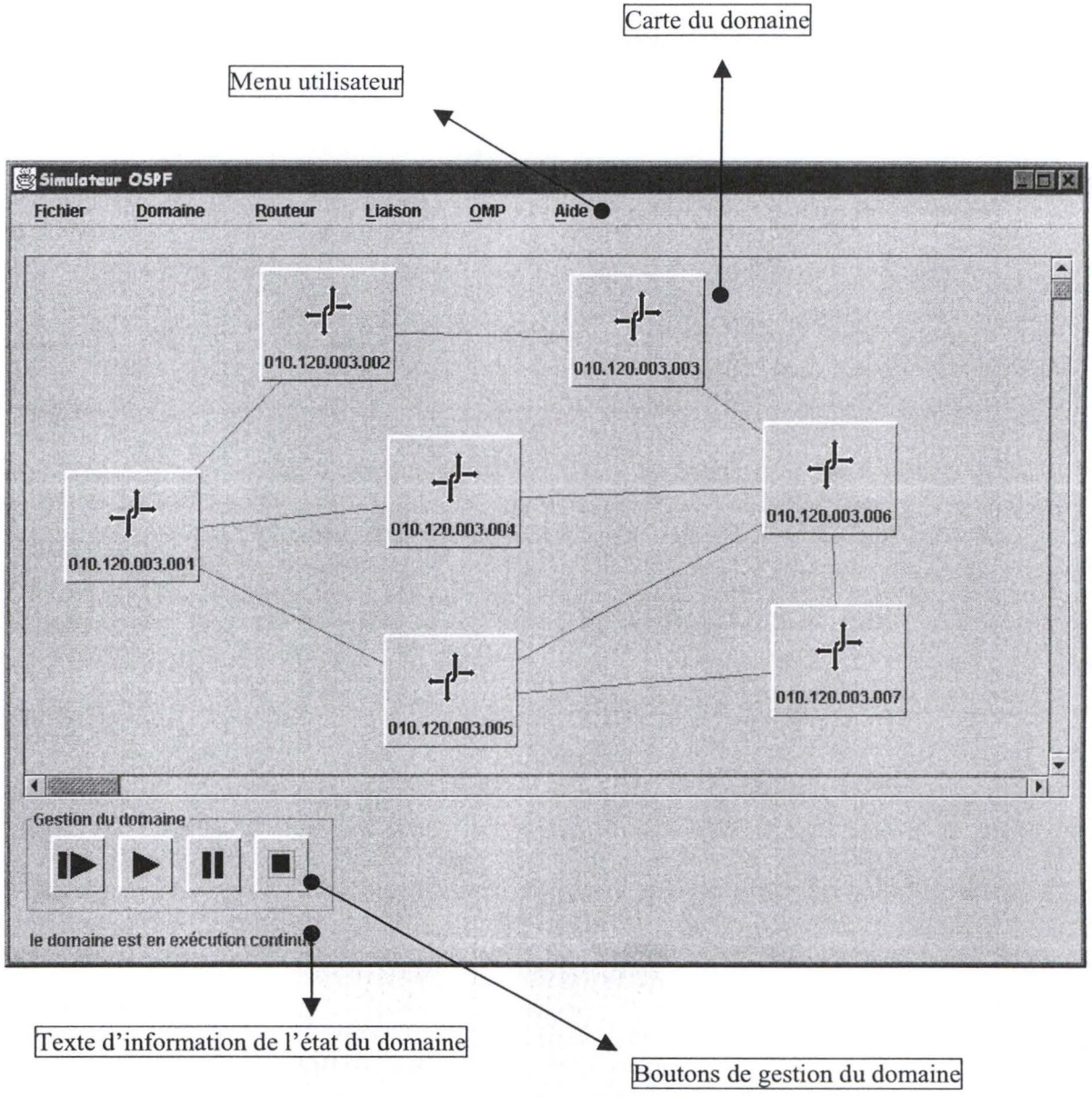


Figure 1 : interface utilisateur

L'interface utilisateur (figure 1) est divisée en 4 zones :

1) le menu ;
2) la carte du domaine qui permet de voir l'évolution graphique du domaine ;
3) la gestion du domaine qui permet de déterminer le mode d'évolution du domaine ;
4) l'information de l'état actuel du domaine .

En toute généralité, un domaine peut se trouver dans deux états :

1) l'état *d'arrêt* qui est son état lors du démarrage de l'application. Dans cet état, il est possible de créer une configuration de domaine (création de routeurs et de liaisons) mais il n'est pas possible de lui affecter du trafic ;

2) l'état *démarré* qui permet le fonctionnement normal des routeurs et l'utilisation de toutes les possibilités du simulateur.

6. La carte du domaine

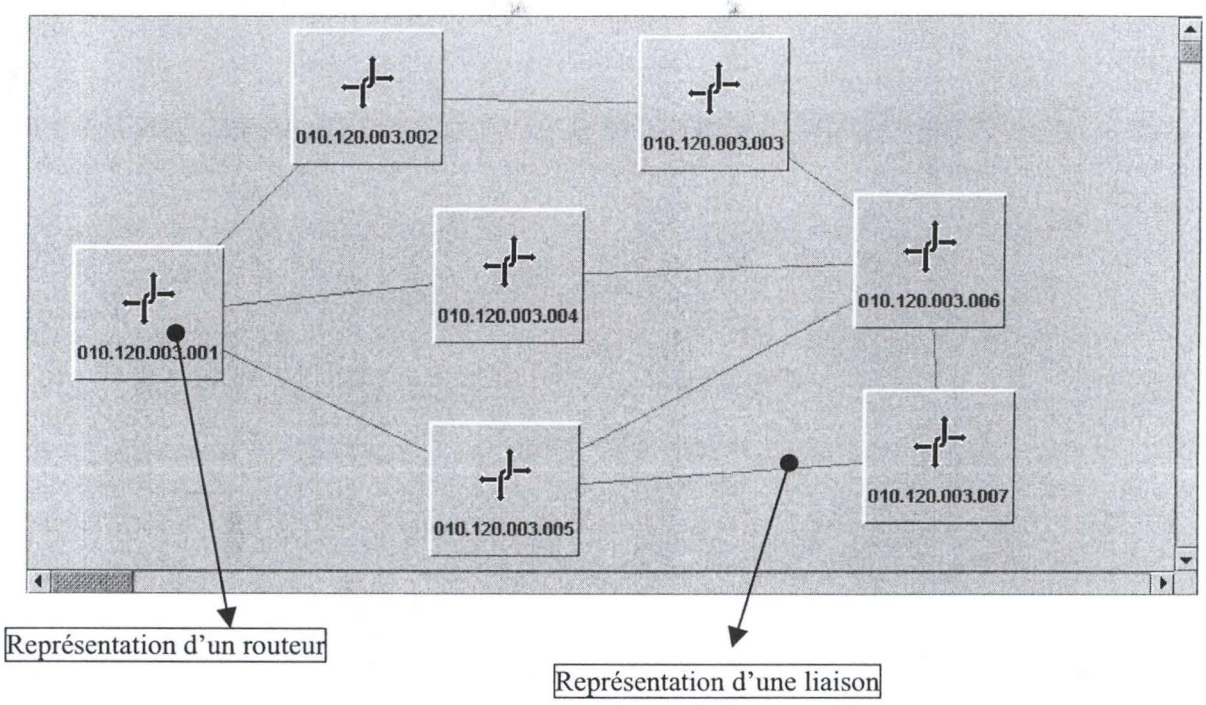


Figure 2 :Carte d'un domaine OSPF

a. Représentation d'un routeur

Un routeur est représenté par un rectangle en relief. Il se compose d'un symbole et d'une adresse IP. Cette adresse prendra une couleur rouge si le routeur n'est pas démarré et prendra une couleur verte si le routeur est démarré.

Un routeur peut être déplacé sur la carte en utilisant la technique du « drag & drop » avec la souris.

b. Représentation d'une liaison

Une liaison est représentée par un segment unissant deux routeurs. Elle peut prendre une des couleurs citées ci-après témoignant de son état :

- **Gris** : la liaison est inactive, c'est-à-dire qu'elle ne peut pas servir au transfert de données ;
- **Jaune** : Passage sur la liaison d'un message du protocole *Hello* ;
- **Noir** : la liaison est active, c'est-à-dire qu'elle est apte au transfert de données et qu'elle est donc bidirectionnelle ;
- **Rouge** : Passage sur la liaison d'un message du protocole d'*inondation* et contenant un LSA de type 1 (description d'un routeur) ;
- **Bleu** : Passage sur la liaison d'un message du protocole d'*inondation* et contenant un OpaqueLSA de type 10 (description de la charge d'une liaison).

7. La gestion du domaine

Les quatre boutons qui vont être présentés servent à déterminer la manière dont le domaine évolue dans le temps. On retrouvera une évolution continue, une évolution pas-à-pas (dans ce cas, chaque seconde ou chaque traitement d'un message au niveau d'un routeur est considéré comme une étape), un figement de l'évolution du domaine et son arrêt.

Un bouton qui apparaît en grisé ne peut pas être utilisé.

Avance pas-à-pas. Ce bouton peut être utilisé pour démarrer un domaine mais également durant son exécution. Chaque utilisation de ce bouton implique :

- le vieillissement de tous les enregistrements des bases de données locales des routeurs d'une seconde ;
- le traitement et l'envoi d'un message au niveau des routeurs qui possèdent encore des informations reçues et non traitées.

Le passage à l'étape suivante se fait en cliquant une nouvelle fois sur ce bouton.
On quitte l'état pas-à-pas en cliquant sur :

- le bouton *avance continue* (suppression de l'exécution pas-à-pas) ;
- le bouton *arrêt* (on quitte l'application) .

Exécution continue. Ce bouton peut être utilisé pour démarrer le domaine ou pour quitter l'état d'exécution en pas-à-pas. L'utilisation de ce bouton lance le déroulement du domaine de manière continue. On quitte l'état continu en cliquant soit sur :

- le bouton *avance pas-à-pas* pour réaliser une exécution pas-à-pas, ;
- le bouton *pause* pour figer l'exécution ;
- le bouton *arrêt* pour quitter l'application.

Pause. Ce bouton sert à figer un domaine en cours d'exécution. Il ne peut être utilisé que durant une exécution continue. On quitte l'état figé en cliquant soit sur :

- le bouton *exécution continue* pour relancer le domaine ;
- le bouton *arrêt* pour quitter l'application.

Arrêt. Ce bouton sert à quitter l'application. Il peut être utilisé à tout moment et dans n'importe quel état du domaine.

Il faut noter que le démarrage du domaine entraîne de manière automatique le démarrage de tous les routeurs déjà créés dans le domaine. Par la suite, les routeurs créés devront être explicitement démarrés.

8. Les menus

a. la structure complète des menus

On donne ici la structure complète du menu *utilisateur* représenté dans la barre au sommet de l'application. Un menu souligné en trait discontinu indique que celui-ci, ainsi que ses sous-éléments, ne sont disponibles que lorsque le domaine est démarré. De la même manière, un menu souligné en trait continu indique son indisponibilité lorsque le domaine est démarré.

- Fichier
  - Nouveau
  - Ouvrir
  - Sauver
  - Quitter

- Domaine
  - Dump des messages
  - Faire le dump
  - Vitesse
    - Vitesse OSPF
    - Vitesse OMP

- Routeur
  - Créer Routeur
  - Démarrer Routeur
  - Supprimer Routeur
  - Base de Données
  - Table de Routage

- Liaison
  - Créer Liaison
  - Supprimer Liaison
  - Modifier cout Liaison
  - Option
    - Liaison Bidirectionnelle

- OMP
  - Trafic
    - Adapter Trafic
    - Ouvrir
    - Sauver
  - Charge
    - Charge Classique
    - Charge Pondérée
  - NextHopStructure

- Aide
  - Manuel
  - A Propos

b. Menu Fichier

Le menu *fichier* gère l'ouverture, la sauvegarde et la terminaison de l'application. Il se compose de quatre éléments :

1) Nouveau : création d'un nouveau domaine à l'arrêt ;
2) Ouvrir : ouverture d'un domaine sauvegardé ;
3) Sauvegarder : sauvegarde du domaine courant ;
4) Quitter : quitter l'application.

La sauvegarde ne peut avoir lieu que sur un domaine qui n'a pas encore été démarré.

c. Menu Domaine

1) Description de l'interface de dump

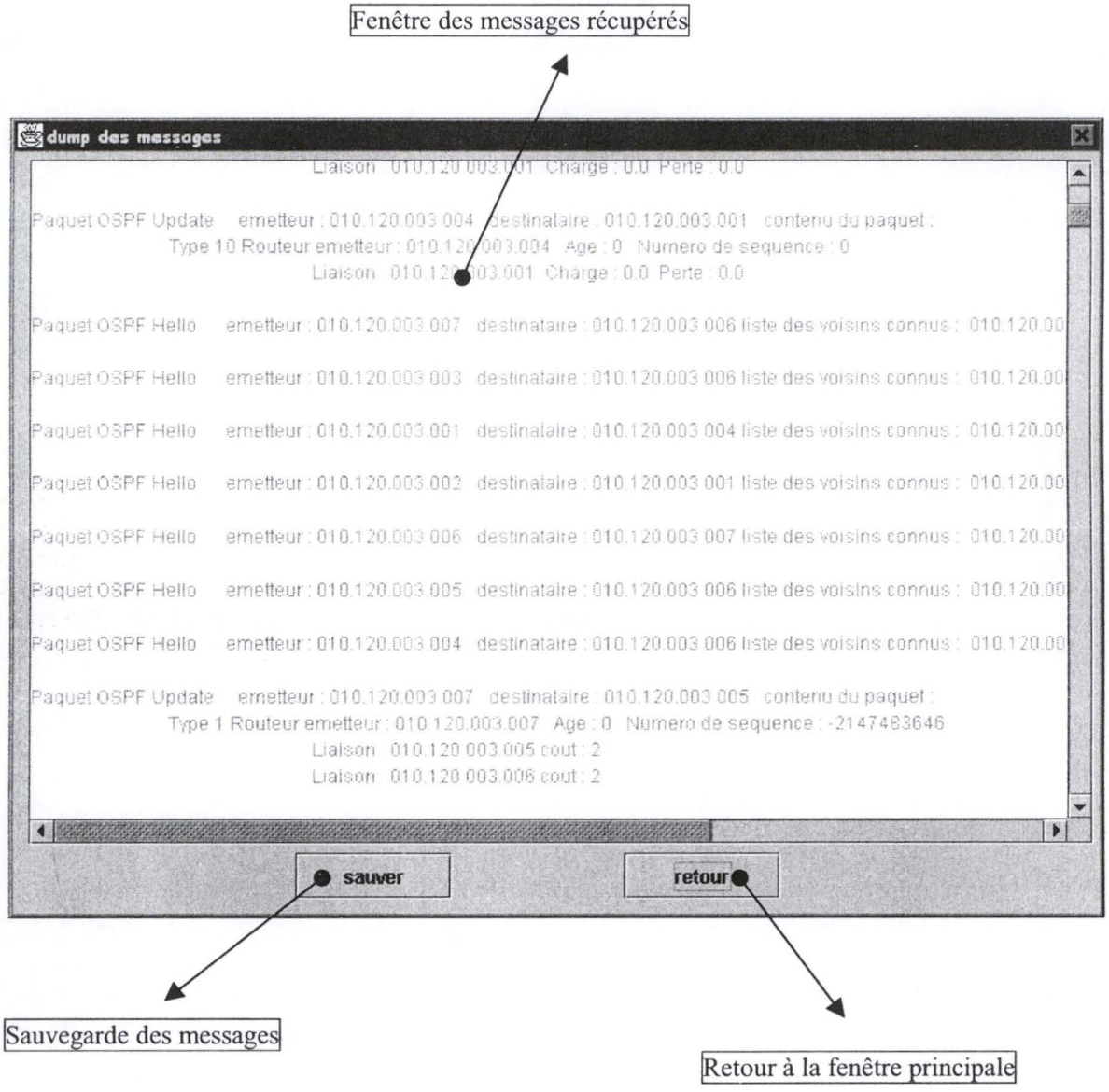


Figure 3 :Interface de dump des messages

L'interface de dump des messages (Figure 3) contient les éléments suivants :

- une zone de texte qui affiche tous les messages diffusés dans le domaine ;
- un bouton *sauver* qui ouvre une interface offrant la possibilité de sauvegarder tous les messages dans un fichier au format texte ;
- un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

2) Description de la structure des messages

Pour des facilités de compréhension, certaines parties des messages décrits ci-après ont été accentuées en italique ou en gras. Ces accentuations n'existent pas dans l'application.
Les parties en gras dans les messages représentent l'intitulé du champ. Le ou les éléments en italique qui les suivent représentent les données de ce champ.

- Message « Hello »

Paquet OSPF Hello emetteur : *010.120.003.001* **destinataire :** *010.120.003.002* **liste des voisins connus :** *010.120.003.004 010.120.003.005*

Ce message est un message Hello émis par le routeur *010.120.003.001* à destination du routeur *010.120.003.002* et qui contient la liste des voisins connus par le routeur *010.120.003.001*.

- Message de description de configuration du protocole d'inondation (LSA type 1)

Paquet OSPF Update emetteur : *010.120.003.006* **destinataire :** *010.120.003.007* **contenu du paquet :**
Type *1* **Routeur emetteur :** *010.120.003.001* **âge :** *0* **Numéro de séquence :** -2147483644 **Nombre de liaisons** : *3*
Liaison : *010.120.003.002* **poids :** *5*
Liaison : *010.120.003.004* **poids :** *5*
Liaison : *010.120.003.005* **poids :** *5*

Ce message est un message de description d'une configuration locale émis par le routeur *010.120.003.006* à destination du routeur *010.120.003.007*. Ce message contient un LSA de type 1 créé par le routeur *010.120.003.001*. Ce LSA a un âge de 0 et un numéro de séquence qui vaut –2147483644. Il contient la liste des liaisons actives dont le routeur *010.120.003.001* est à l'origine. Ces liaisons sont au nombre de 3. Chaque liaison est décrite par son destinataire et son poids (par exemple, **Liaison :** *010.120.003.002* **coût :** *5* indique qu'il existe une liaison entre *010.120.003.001* et *010.120.003.002* avec un poids de 5).

- Message de description de charge des liaisons du protocole d'inondation (LSA type 10 – OpaqueLSA)

Paquet OSPF Update emetteur : *010.120.003.006* **destinataire :** *010.120.003.007* **contenu du paquet :**
Type *10* **Routeur emetteur :** *010.120.003.001* **âge :** *0* **Numéro de séquence :** 5
Liaison : *010.120.003.002* **charge :** *40.0* **perte :** *0.0*

Ce message est un message de description d'une liaison émis par le routeur *010.120.003.006* à destination du routeur *010.120.003.007*. Ce message contient un LSA de type 10 créé par le routeur *010.120.003.001*. Ce LSA a un âge de 0 et un numéro de séquence qui vaut 5. Il contient la description de la charge d'une liaison active dont le routeur *010.120.003.001* est à l'origine. Cette liaison est décrite par son destinataire, sa charge et ses pertes exprimées en pour-cent (par exemple, **Liaison :** *010.120.003.002* **charge :** *40.0* **perte :** *0.0* indique qu'il existe une liaison entre *010.120.003.001* et *010.120.003.002* avec une charge de 40 % et un taux de perte de 0 %).
Lorsque le dump dépasse 5000 messages, les 1000 plus anciens sont automatiquement supprimés.

3) Faire le Dump

Le menu *Domaine* propose un checkbox *Faire le dump*. Lorsque le checkBox est sélectionné, tous les messages générés dans le domaine sont capturés. Lorsqu'il est déselectionné, aucun message n'est recupéré. Par défaut, le checkBox est déselectionné. L'attention est attirée sur le fait que la quantité de messages générés est relativement importante. Aussi, dès que le nombre de messages stockés atteint 5000, les 1000 plus anciens sont automatiquement supprimés.

4) Gestion de la vitesse

L'application offre la possibilité d'augmenter la vitesse de déroulement du domaine. Cette accélération peut affecter le domaine à deux niveaux :

1) *Vitesse OSPF* : augmenter la fréquence de rafraîchissement des enregistrements dans les bases de données des routeurs ;
2) *Vitesse OMP* : diminuer d'un facteur l'influence du temps dans les algorithmes de OMP.

L'accélération par défaut est nulle (1 X).

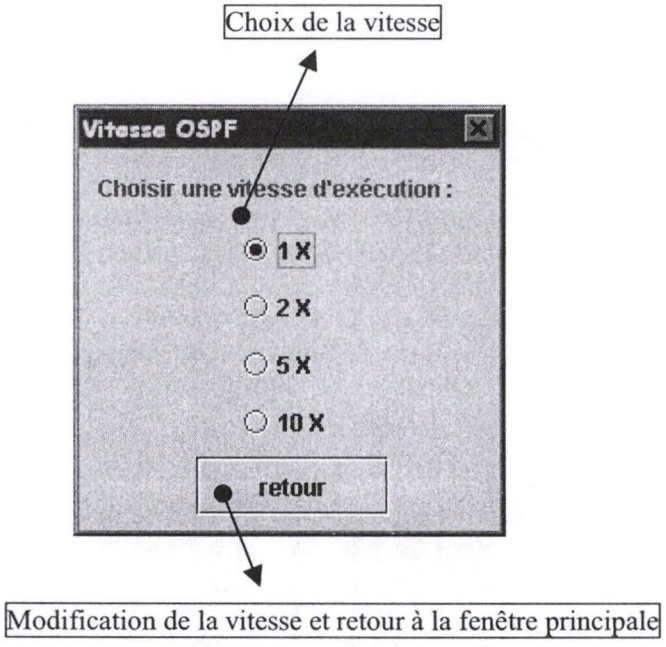


Figure 4 :Interface de gestion de la vitesse

La figure 4 donne l'interface de gestion de la vitesse pour OSPF. Cette interface est identique à celle utilisée pour la gestion de la vitesse de OMP. Elle se compose de 2 éléments :

1) une zone *choix de la vitesse* qui permet de déterminer parmi 4 possibilités la vitesse que l'on désire donner au domaine ;
2) un bouton *retour* qui permet de valider la modification éventuelle de vitesse et de retourner à la fenêtre principale.

d. Menu Routeur

Ce menu est responsable de la gestion des routeurs dans le domaine.

1) Créer Routeur

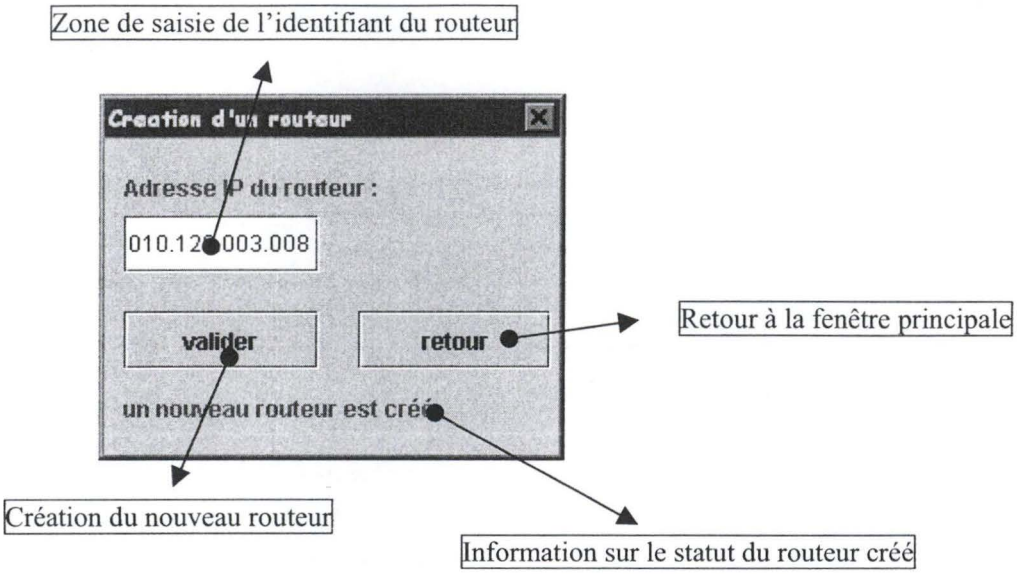


Figure 5 :Interface de création d'un routeur

Cette fenêtre (figure 5) permet de créer un nouveau routeur. Elle se compose de 4 éléments :

1) un champ de saisie pour l'identifiant du routeur. Cet identifiant doit être une adresse IP valide ;
2) un bouton *valider* qui a pour effet de créer un nouveau routeur. Ce routeur est positionné sur la carte du domaine et la couleur de son identifiant est rouge indiquant qu'il n'est pas démarré ;
3) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application ;
4) une ligne d'informations sur le statut du routeur créé et des messages d'erreur.

2) Démarrer Routeur

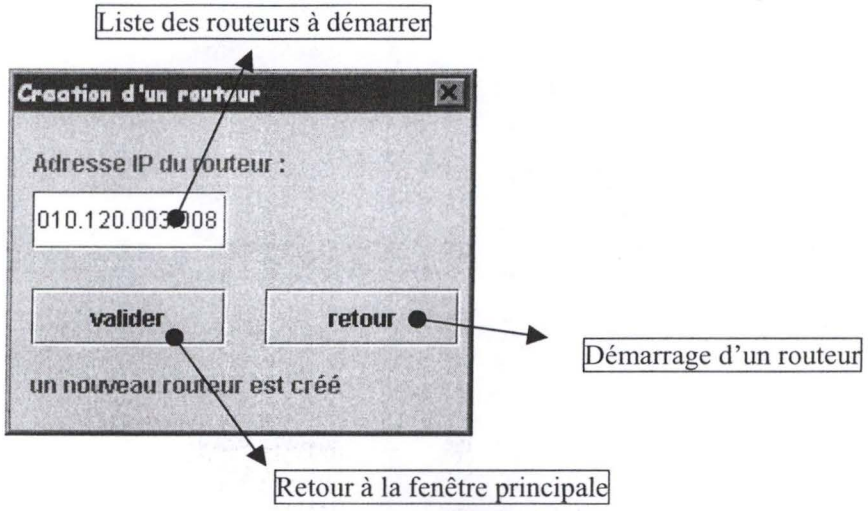


Figure 6 :Interface de démarrage d'un routeur

Cette fenêtre (figure 6) permet de démarrer un routeur déjà créé. Elle ne peut être utilisée que lorsque le domaine est déjà démarré. Elle se compose de 3 éléments :

1) une liste déroulante dans laquelle on retrouve tous les routeurs créés mais non encore démarrés. Un routeur doit être choisi parmi cette liste pour être démarré ;
2) un bouton *valider* qui permet de démarrer le routeur choisi. Le routeur démarré voit alors la couleur de son nom passer du rouge au vert indiquant de ce fait qu'il est démarré ;
3) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

3) Supprimer un routeur

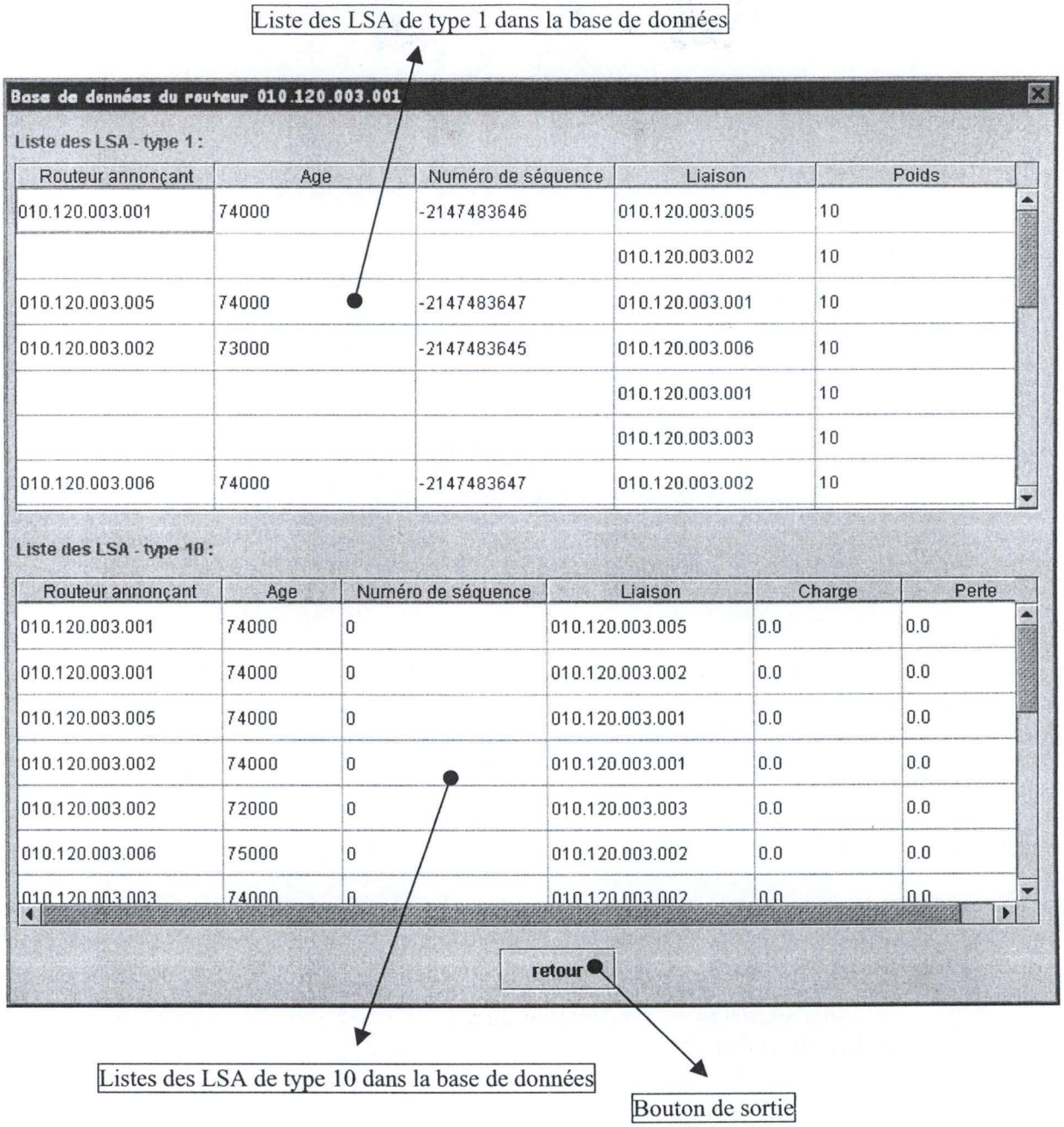
Cette fenêtre permet de supprimer un routeur dans le domaine. Son interface est semblable à celle de démarrage d'un routeur (figure 6) mais :

- la liste des routeurs reprend tous les routeurs du domaine ;
- le bouton *valider* cause la suppression du routeur choisi dans la liste. Ce routeur disparaît de la carte du domaine.

4) Base de Données

Une première fenêtre s'ouvre permettant de choisir le routeur pour lequel on désire afficher sa base de données locales. Cette interface est semblable à celle de démarrage d'un routeur (figure 6) mais :

- la liste des routeurs reprend tous les routeurs du domaine ;
- le bouton *valider* cause l'affichage de la fenêtre de la base de données du routeur choisi.



Routeur annonçant	Age	Numéro de séquence	Liaison	Poids
010.120.003.001	74000	-2147483646	010.120.003.005	10
			010.120.003.002	10
010.120.003.005	74000	-2147483647	010.120.003.001	10
010.120.003.002	73000	-2147483645	010.120.003.006	10
			010.120.003.001	10
			010.120.003.003	10
010.120.003.006	74000	-2147483647	010.120.003.002	10

Routeur annonçant	Age	Numéro de séquence	Liaison	Charge	Perte
010.120.003.001	74000	0	010.120.003.005	0.0	0.0
010.120.003.001	74000	0	010.120.003.002	0.0	0.0
010.120.003.005	74000	0	010.120.003.001	0.0	0.0
010.120.003.002	74000	0	010.120.003.001	0.0	0.0
010.120.003.002	72000	0	010.120.003.003	0.0	0.0
010.120.003.006	75000	0	010.120.003.002	0.0	0.0
010.120.003.003	74000	0	010.120.003.002	0.0	0.0

Figure 7 :Base de données d'un routeur

La fenêtre de base de données (figure 7) d'un routeur contient 3 zones :

1) la première reprend la liste des dernières versions des LSA de type 1 que le routeur a généré ou qu'il a reçu. Ce LSA peut être représenté sur plusieurs lignes. Un nouveau LSA commence lorsque les 3 premières colonnes d'une ligne ne sont pas vides. Les lignes dont les 3 premières colonnes sont vides dépendent de la ligne de niveau supérieur ayant les 3 premières colonnes remplies. Ainsi, dans notre exemple, les 2 premières lignes forment un LSA et la suivante en forme un autre. Chaque ligne est composée de 5 colonnes dont chacune décrit un élément particulier :

   - **Router annonçant** donne l'identifiant du routeur à l'origine de ce LSA ;

- **Age** donne l'âge de ce LSA dans la base de données. Tous LSA de type 1 dont l'âge dépasse 3.600.000 msec est automatiquement supprimé ;
- **Numéro de séquence** donne le numéro de séquence de ce LSA ;
- **Liaison** donne l'identifiant de la liaison décrite (une liaison est toujours identifiée par son origine et sa destination) ;
- **Poids** donne le poids affecté à la liaison décrite.

2) la seconde reprend la liste des dernières versions des LSA de type 10 que le routeur a généré ou qu'il a reçu. On a un LSA différent par ligne dans la base de données. Chaque ligne est composée de 5 colonnes dont chacune décrit un élément particulier :

- **Router annonçant** donne l'identifiant du routeur à l'origine de ce LSA ;
- **Age** donne l'âge de ce LSA dans la base de données. Tous LSA de type10 dont l'âge dépasse 3.600.000 msec est automatiquement supprimé ;
- Numéro de séquence donne le numéro de séquence de ce LSA (ce numéro de séquence commence à 0 pour les LSA de type 10 dans le seul but de pouvoir les distinguer plus facilement sur l'interface) ;
- **Liaison** donne l'identifiant de la liaison décrite (une liaison est toujours identifiée par son origine et sa destination) ;
- **Charge** donne la charge en pour-cent de la liaison décrite ;
- **Perte** donne la perte en pour-cent de la liaison décrite.

3) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

5) Table de routage

Une première fenêtre (figure 6) s'ouvre permettant de choisir le routeur pour lequel on désire afficher sa base de données locales. Cette interface est semblable à celle de démarrage d'un routeur mais :

- la liste des routeurs reprend tous les routeurs du domaine ;
- le bouton *valider* cause l'affichage de la fenêtre de la base de données du routeur choisi ;

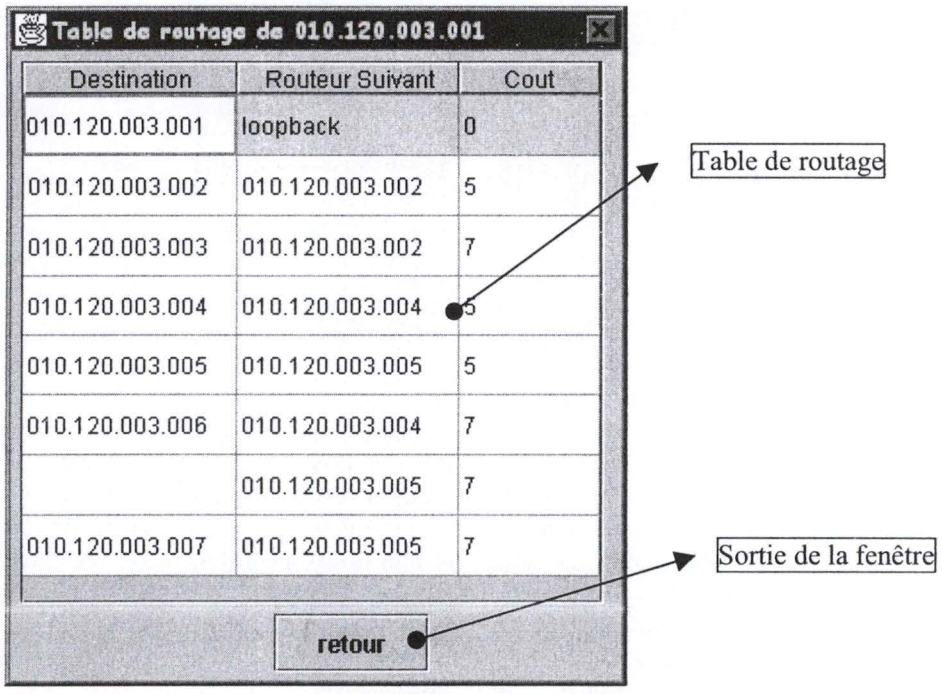


Figure 8 :Table de routage

La table de routage d'un routeur (figure 8) reprend deux zones :

1) la table de routage proprement dite. Celle-ci décrit dans chacune de ses lignes un chemin vers une destination. Une ligne se compose de 3 champs à savoir :

   - **Destination** qui représente la destination du chemin ;
   - **Routeur Suivant** qui indique la première liaison à parcourir sur le chemin vers la destination. La valeur *loopback* indique qu'on est déjà à la destination recherchée. La valeur *null* indique que la destination n'est pas joignable ;
   - **Cout** indique le coût du chemin considéré.

   Une destination peut avoir plusieurs chemins de coût minimal. Dans ce cas, chaque chemin possible sera représenté par une ligne dans la table mais seul le premier chemin aura une valeur dans sa première colonne. Les lignes qui n'ont pas de valeur dans leur première colonne représenteront les autres chemins possibles vers cette destination (il se raccorde à la première ligne qui a une valeur dans sa première colonne). Par exemple, dans la figure 8, il existe deux chemins pour joindre *010.120.003.006*, le premier passant par *010.120.003.004* et le second par *010.120.003.005*.

2) le bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

e. *Menu Liaison*

Ce menu est responsable de la gestion des liaisons dans le domaine.

1) Créer Liaison

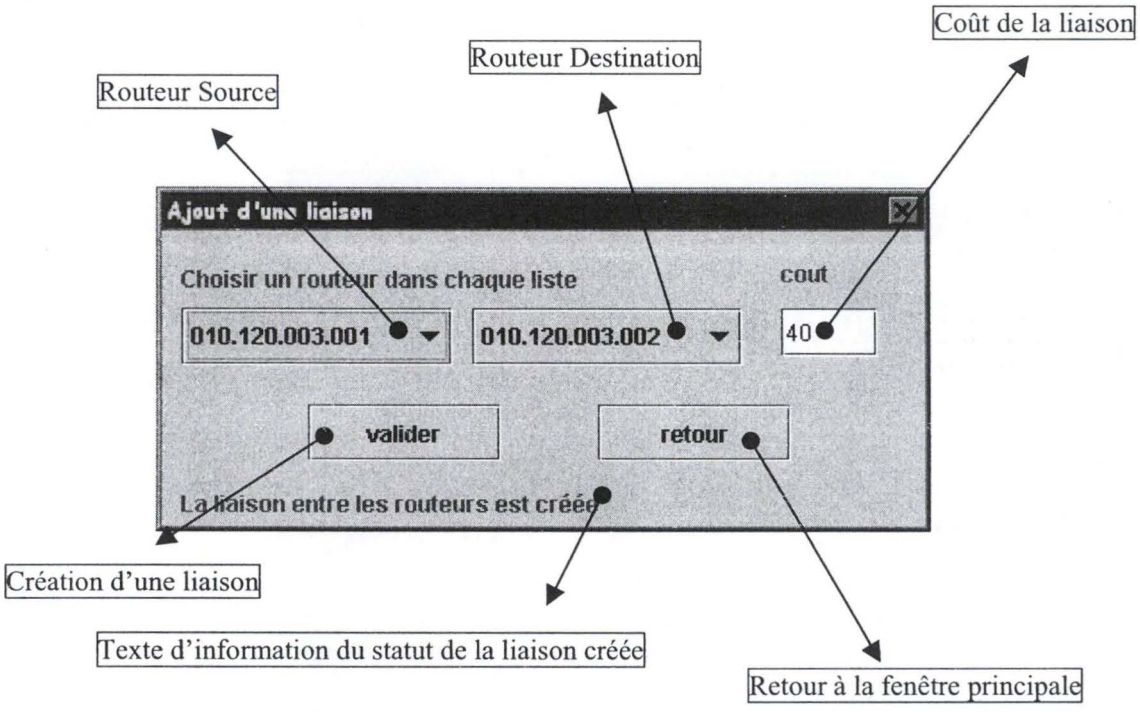


Figure 9 :Interface de création d'une liaison

Cette fenêtre (figure 9) permet de créer une nouvelle liaison entre deux routeurs. Elle se compose de 6 éléments :

1) une liste déroulante dans laquelle on choisit le routeur source de la liaison ;
2) une liste déroulante dans laquelle on choisit le routeur destination de la liaison ;
3) une zone de texte dans laquelle on indique le coût de la liaison. Celui-ci doit être supérieur ou égal à 0 ;
4) un bouton *valider* qui permet de créer la liaison. La création de la liaison entraîne la création d'un segment sur la carte du domaine entre le routeur source et le routeur destination. Cette liaison prend une couleur grise indiquant qu'elle est actuellement inactive. Si l'option *liaison bidirectionnelle* est choisie[42], alors une liaison est également créée entre le routeur destination et le routeur source avec le même poids. Il est à noter qu'une liaison ne peut être créée qu'entre deux routeurs différents et qu'une liaison orientée entre deux routeurs doit être unique ;
5) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application ;
6) une ligne de texte qui donne des informations sur le statut de la liaison créée et des messages d'erreur.

2) Supprimer une liaison

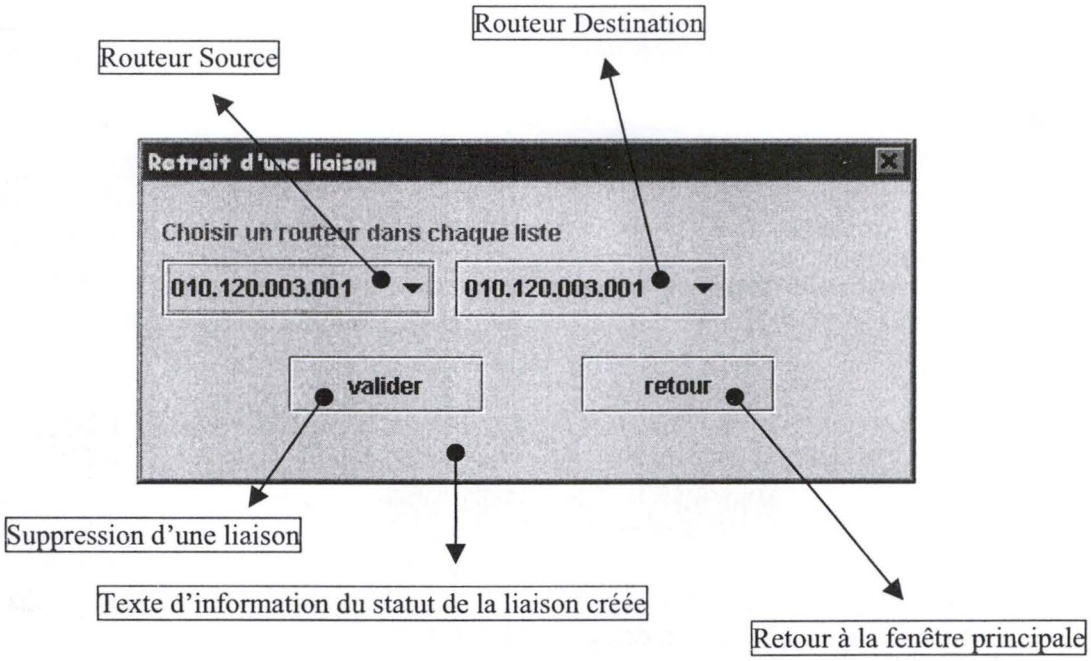


Figure 10 : Interface de suppression d'une liaison

Cette fenêtre (figure 10) permet de supprimer une liaison existante entre deux routeurs. Elle se compose de 5 éléments :

1) une liste déroulante dans laquelle on choisit le routeur source de la liaison ;
2) une liste déroulante dans laquelle on choisit le routeur destination de la liaison ;
3) un bouton *valider* qui permet de supprimer la liaison. La représentation de la liaison est supprimée de la carte du domaine si la liaison était unidirectionnelle. Si la liaison était bidirectionnelle, alors, son vis-à-vis (c'est-à-dire la liaison entre le routeur destination et le routeur source) est mis en gris (on le rend inactif) ;
4) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application ;

[42] Voir Option du menu Liaison

5) une ligne de texte qui donne des informations sur le statut de la liaison supprimée et des messages d'erreur.

3) Modifier Coût Liaison

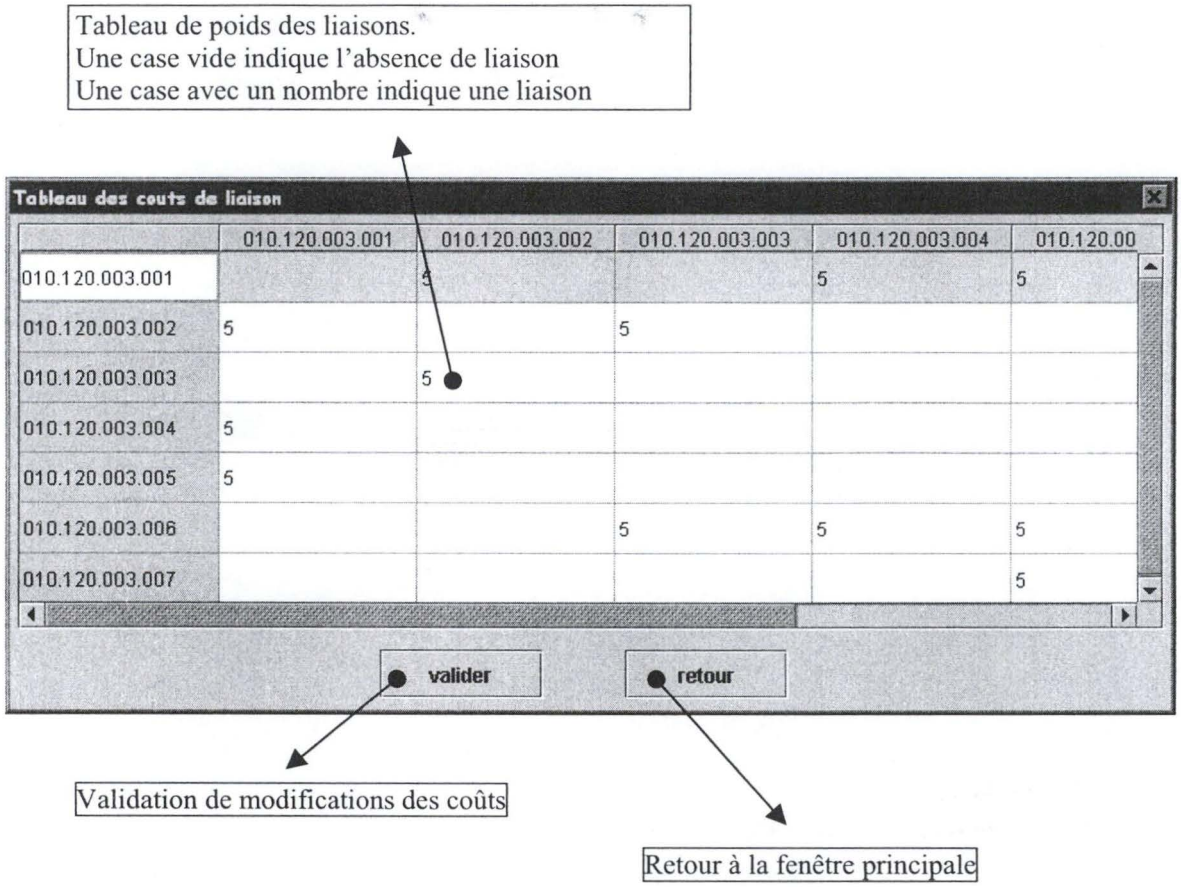


Figure 11 :Interface de modification des coûts de liaison

Cette fenêtre (figure 11) permet de visualiser le poids affecté à une liaison et éventuellement de changer celui-ci. La fenêtre est composée de trois éléments :

1) un tableau indiquant le poids des liaisons. Une case vide de coordonnées (X,Y) indique qu'il n'existe pas de liaison orientée entre le routeur X et le routeur Y. Une case de coordonnées (X,Y) avec un nombre indique par contre l'existence d'une liaison entre le routeur X et le routeur Y; le nombre mentionne alors le poids associé à cette liaison. Toute case contenant un nombre peut être modifiée avec un autre nombre ayant une valeur entière supérieure ou égale à zéro. Les cases vides ne peuvent pas être modifiées ;
2) un bouton *valider* qui permet d'encoder les modifications éventuelles apportées dans le tableau ;
3) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

4) Bidirectionnalité

Le sous-menu *option* permet de modifier la directionnalité d'une liaison lors de sa création. Lorsque le checkBox est sélectionné, la création d'une liaison entraîne automatiquement la création de son vis-à-vis en sens contraire et avec le même poids. Par contre, si le checkBox n'est pas sélectionné, la liaison n'est créée que dans le sens demandé.

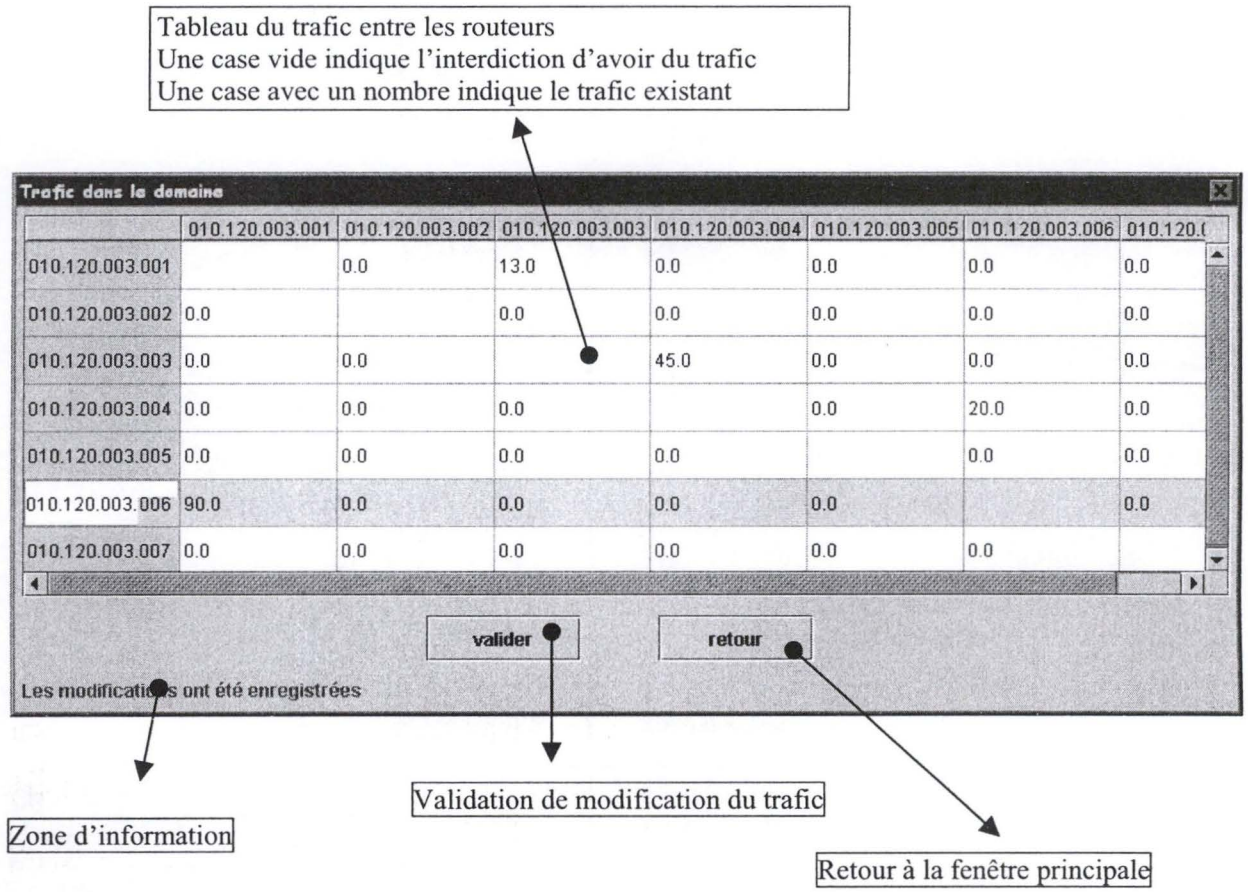
f. OMP

Ce menu est responsable de la gestion et de l'affichage des éléments propres à OMP.

1) Trafic

Ce sous-menu gère le trafic généré par les routeurs dans le domaine. A partir de ce trafic, il sera possible de calculer la charge de chacune des liaisons du domaine en utilisant la technique ECMP de OSPF ou en se basant sur la répartition proposée par OMP.

a) Adapter le trafic



	010.120.003.001	010.120.003.002	010.120.003.003	010.120.003.004	010.120.003.005	010.120.003.006	010.120.(
010.120.003.001		0.0	13.0	0.0	0.0	0.0	0.0
010.120.003.002	0.0		0.0	0.0	0.0	0.0	0.0
010.120.003.003	0.0	0.0		45.0	0.0	0.0	0.0
010.120.003.004	0.0	0.0	0.0		0.0	20.0	0.0
010.120.003.005	0.0	0.0	0.0	0.0		0.0	0.0
010.120.003.006	90.0	0.0	0.0	0.0	0.0		0.0
010.120.003.007	0.0	0.0	0.0	0.0	0.0	0.0	

Figure 12 :Interface d'adaptation du trafic dans le réseau

Cette fenêtre (figure 12) permet de visualiser et de modifier le trafic généré par un routeur du domaine à destination d'un autre routeur de ce même domaine. La fenêtre est composée de quatre éléments :

1) un tableau indiquant le trafic. Une case de coordonnées (X,Y) indique le trafic généré par le routeur X à destination du routeur Y. A l'origine, toutes les cases ont pour valeur 0.0. L'utilisateur peut modifier cette valeur en lui affectant une nouvelle valeur comprise entre 0.0 et 100.0 (cette valeur correspond au pourcentage de trafic par rapport au trafic maximal qu'une source peut générer). La somme du trafic généré par un routeur ne peut jamais excéder 100.0 (en d'autres mots, la somme des trafics sur une ligne doit valoir 100.0 au maximum). Un routeur ne peut pas générer du trafic vers lui-même ; c'est la raison pour laquelle les cases représentant ce trafic sont vides ;

2) un bouton *valider* qui permet d'encoder les modifications éventuelles apportées dans le tableau ;
3) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application ;
4) une ligne de texte qui donne des informations sur le statut de la liaison supprimée et des messages d'erreur.

b) Ouvrir et sauver du trafic

- Le menu *Ouvrir* permet de récupérer une matrice de trafic sauvegardée. Si la matrice n'a pas les mêmes dimensions que le domaine actuel, alors une boîte de dialogue s'ouvre demandant s'il faut faire une adaptation automatique. En cas de réponse positive, la charge du domaine est adaptée comme suit :
    - si la matrice est plus grande, alors, les éléments qui dépassent à droite et en bas ne sont pas pris en compte ;
    - si la matrice est plus petite, alors, les éléments qui ne sont pas pris en compte par cette matrice sont mis à zéro.
- Le menu *Sauvegarder* permet de placer dans un fichier la matrice actuelle du trafic.

2) Charge du domaine

a) Classique

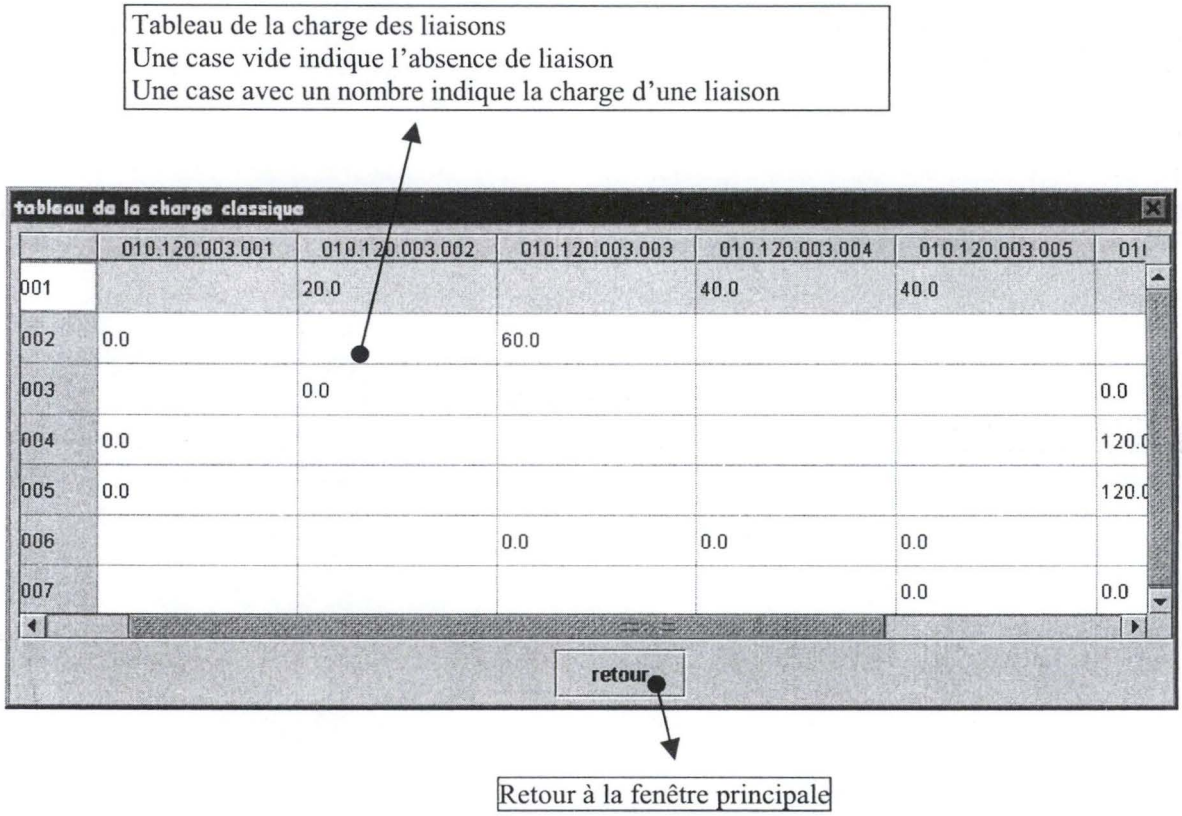


Figure 13 :Représentation de la charge classique du domaine

Cette fenêtre (figure 13) décrit la charge supportée par chacune des liaisons du domaine lorsqu'on n'utilise que les chemins donnés par OSPF. On utilise ECMP (Equal Cost Multi Path) pour répartir la charge de manière équitable lorsque plusieurs chemins sont disponibles pour rejoindre une destination.

La fenêtre comprend 2 éléments :

1) un tableau indiquant la charge sur les liaisons. Une case de coordonnées (X,Y) avec un nombre indique la charge sur la liaison dont le routeur X est la source et dont la destination est le routeur Y. Une case vide indique l'absence de liaison entre les deux routeurs. Une charge est exprimée en pourcentage d'occupation par un nombre supérieur ou égal à 0.0. Si ce nombre dépasse 100.0, alors le reste (c'est à dire le nombre auquel on retire 100.0) correspond au pourcentage de trafic perdu. Il est à noter qu'une liaison qui perd une partie de son trafic voit cette perte répercutée sur les chemins en aval.

   Par exemple, supposons le chemin A – B – C avec une charge de 150.0 pour la liaison entre A et B et une charge de 60.0 pour la liaison entre B et C. Si A génère 60.0 à destination de B, comme 1/3 des paquets sont perdus sur la liaison entre A et B, le routeur B ne recevra que 40.0 qu'il transmettra intégralement au routeur C. Par contre si la liaison entre A et B vaut 80.0, dans ce cas, B recevra 60.0 venant de A ; comme les 20.0 perdus précédemment existent en B, ils pourront être communiqués à C ce qui fera augmenter la charge de la liaison entre B et C de 20.0 lui donnant en résultat une charge de 80.0.

2) un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

b) Charge pondérée

Cette fenêtre décrit la charge supportée par chacune des liaisons du domaine lorsque ce domaine utilise les chemins donnés par OMP. Dans ce cas, la charge sera répartie proportionnellement entre les différents chemins suivant l'allocation que OMP leur concède.

La structure et l'interprétation de la fenêtre sont identiques à celle donnant la charge évaluée de manière classique (figure 13).

3) Next Hop Structure

Une première fenêtre (figure 10) s'ouvre permettant de choisir le routeur pour lequel on désire lire la next Hop Structure et le routeur destination (dans la fenêtre résultat, on n'affiche pas la nextHopStructure complète d'un routeur, mais uniquement la ligne qui concerne une destination précise) . Cette première interface est identique à celle de la suppression d'une liaison.

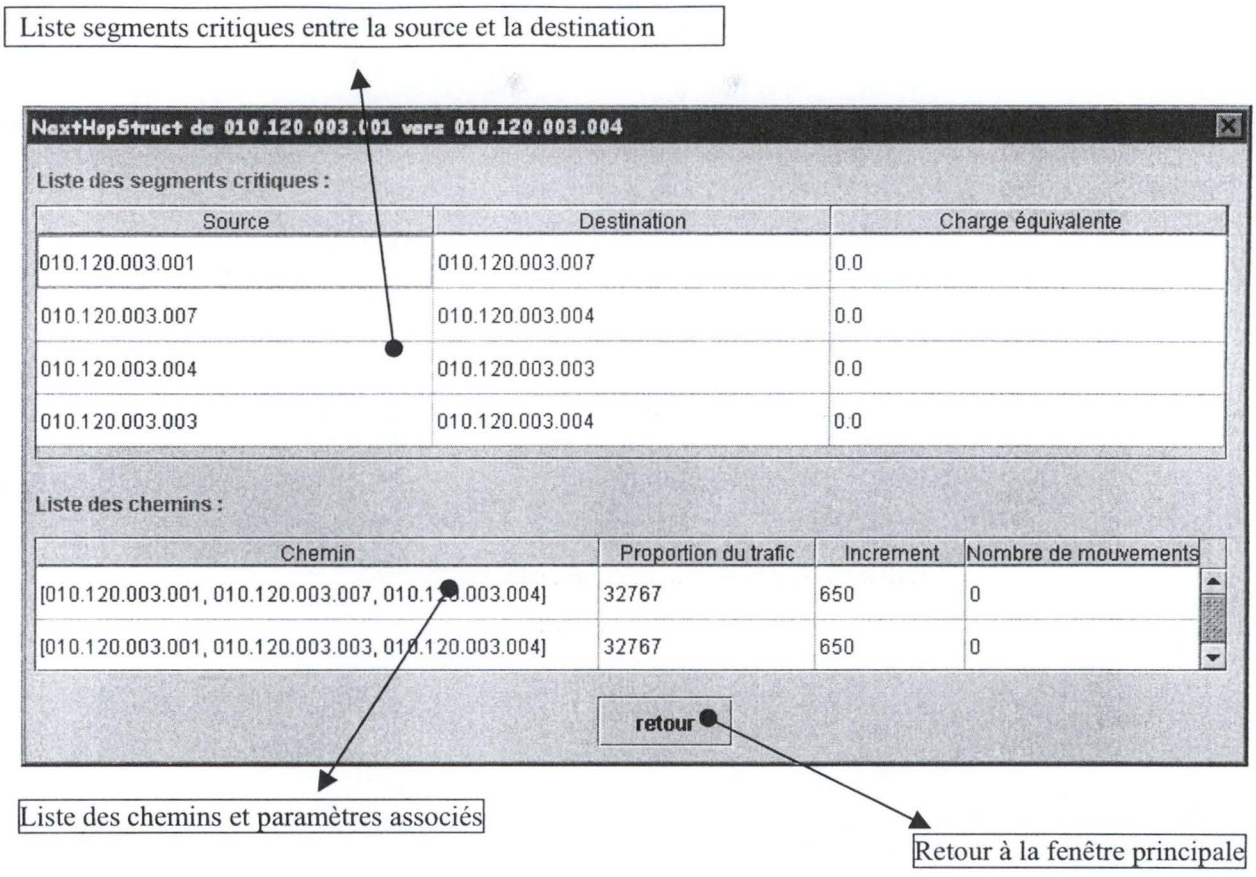


Figure 14 : Next Hop Structure d'une destination

Cette fenêtre (figure 14) se divise en 3 parties :

1) un premier tableau décrit la liste des segments critiques sur l'ensemble des chemins qui unissent le routeur source au routeur destination. On retrouve 3 champs à savoir :
   i) **Emetteur** qui donne le routeur source du segment critique ;
   ii) **Recepteur** qui donne le routeur destination du segment critique ;
   iii) **Charge Equivalente** qui indique la charge équivalente associée au segment critique (si on retrouve plusieurs segments, alors, tous auront la même charge équivalente).
2) un second tableau décrit tous les chemins qui permettent de joindre le routeur source et le routeur destination choisis. Chaque chemin est décrit par 4 champs :
   i) **Chemin** qui donne la description complète du chemin.
   ii) **Proportion du trafic** indique la part du trafic que ce chemin reçoit pour véhiculer les paquets de la source vers la destination. Ce nombre peut varier de 0 (cas où le chemin n'est pas utilisé) à 65535 (cas où ce chemin est le seul à être utilisé). La somme totale de toutes les proportions pour une NextHopStructure doit être égale à 65535[43] ;
   iii) **Increment** indique le trafic supplémentaire que peut recevoir un chemin lorsque le protocole décide de déplacer une proportion de trafic vers lui. Ce nombre est borné entre 65 et 65535 ;
   iv) **Nombre de mouvements** indique le nombre de fois que ce chemin a été successivement incrémenté. Ce nombre sera remis à zéro lorsqu'on recommence une nouvelle série d'accroissement.
3) Un bouton *retour* qui permet de fermer cette fenêtre et de retourner à la fenêtre principale de l'application.

[43] On peut tolérer 65534 en raison des arrondis de représentation.

Annexe 2 : Description du contenu de la disquette

La disquette fournie en annexe avec ce document contient le code source, le code exécutable et les commentaires techniques du simulateur OSPF.

Cette disquette se compose de 4 répertoires :

OSPFsource : qui contient le code source des différentes classes JAVA utilisées par le simulateur. Chacune des classes est commentée dans le détail, principalement en ce qui concerne les aspects techniques.

OSPFcode : qui contient le fichier exécutable. Le lecteur voudra bien se référer au manuel utilisateur quant à l'utilisation de ce fichier exécutable.

OSPFdoc : qui reprend la description complète sous forme de pages HTML des différentes classes ainsi que des champs et des méthodes qui les constituent. L'accès à cette information se fait en sélectionnant le fichier *index.html* dans ce répertoire.

exemple : qui contient quelques exemples de domaines directement utilisables par le simulateur.