UMONS
Université de Mons

Réseaux

Projet 2 : Protocole LSP

Groupe n° 2
Calin Baptiste, matricule n° 010525
Jamar Frédéric, matricule n° 181699

Faculté
des Sciences

Réseaux – Projet 2
Implémentation du protocole LSP

Construction et exécution :

Notre implémentation s'exécute depuis la classe de démo "RoutingDemo". La partie qui concerne le dump de la FIB est faite dans chaque routeur car notre applicatif tourne sans cesse dans le cas où une modification est apportée dans la topologie. Les graphviz ne sont pas créés. L'applicatif est livré en mode non-verbeux. Il peut l'être en mettant à vrai le booléen « verbose » dans la classe LSPRoutingProtocol.

Approche utilisée

En premier lieu, la fonctionnalité de base à développer était la découverte des voisins pour le routeur. On a donc mis en place l'envoi de messages HELLO et ensuite géré la réception de ces derniers pour construire une liste de voisins pour chaque routeur.

En deuxième lieu, après la découverte des voisins par le routeur, il a été décidé de développer la partie du protocole LSP. Ayant la liste des voisins, nous pouvions alors commencer l'envoi de messages LSP et la gestion de la LSDB lors de la réception des LSPMessage dans chacun des routeurs.

En troisième lieu, il a été convenu d'afficher, dans une forme correcte, dans la console, les messages entrants et sortant du routeur avec leur contenu afin de détecter un éventuel problème dans les datagrammes.

En quatrième lieu, l'algorithme de Dijkstra devait être développé afin de calculer, sur base de la LSDB, les meilleures routes pour toute la topologie du réseau. Une fois l'algorithme terminé, le résultat devait être placé dans la FIB du routeur pour y être sauvegardé et utilisé.

En dernier lieu, un mode verbeux a été ajouté dans le cas où l'application nécessiterait un debugging. Le contenu des messages n'était plus qu'affiché dans le mode verbeux. Le même comportement a été appliqué pour les messages entrants.

Description des classes

RoutingDemo:

Cette classe permet de charger la topologie réseaux d'un fichier texte et un scheduler dans l'objet Network. Ensuite, dans l'objet Network, les routeurs instanciés reçoivent leur applicatif

LSPRoutingProtocol. Une fois les routeurs chargés de leur partie applicative, le scheduler est lancé.

LSPRoutingProtocol:

Cette classe permet au routeur de connaître toute la topologie du réseau en implémentant correctement les protocoles HELLO et LSP. Elle permet également de fournir à la FIB les meilleures routes calculées pour chaque point. Dans la méthode « start », un « AbstractTimer » est lancé pour chaque action avec un timer différent et paramétrable. Le routeur va tout d'abord commencer à envoyer des messages HELLO en broadcast et traiter les messages HELLO reçus dans la méthode héritée, « receive », pour connaître ses voisins et établir une liste de ces derniers. Par la suite, des messages LSP vont être envoyés en broadcast et traités en retour par la méthode héritée « receive ». Tant que la LSDB, base de données des LSP reçus, est vide, le calcul des chemins les plus courts ne commencera pas. Une fois des LSP sauvegardés dans la LSDB, la classe commandera le calcul des chemins les plus courts en y passant la LSDB. Une fois que le calcul est terminé, le résultat obtenu est placé dans la FIB.

LSPRoutingEntry:

Cette classe permet l'ajout d'un chemin calculé dans la FIB.

HelloMessage:

Cette classe permet la manipulation d'un message Hello reçu ou à envoyer. Un message Hello est structuré sous la forme d'un champ routeur, d'un autre champ désignant l'adresse IP utilisée par ce routeur et, enfin, d'un dernier champ désignant l'interface utilisée par ce routeur.

LSPMessage:

Cette classe permet la manipulation d'un message LSP structuré sous la forme d'un champ « routerIP » désignant l'adresse IP du routeur manipulant l'objet, d'un deuxième champ « numSeq » représentant le numéro de séquence du message, d'un troisième champ sous forme de dictionnaire ayant pour clé une adresse IP et pour valeur un entier désignant le cout vers cette adresse IP.

Link:

Cette classe permet de définir, pour l'algorithme de Dijkstra, un lien/chemin entre deux routeurs/points. Le lien a une source, une destination étant des objets de type « IPAddress », un objet « IPInterfaceAdapter » et enfin un coût sous forme d'entier.

Point:

Cette classe permet de définir, pour l'algorithme de Dijkstra, un routeur/point possédant une adresse IP comme identifiant, un coût total depuis une source donnée, une liste des liens/chemins, avec leur coût, enregistrés pour arriver à la destination de façon la plus courte, un booléen pour connaître si ce point, lors du calcul, a déjà été visité ou non, un deuxième

booléen pour savoir si ce point étant la source et enfin un ou plusieurs liens attachés à ce point (fournis par la LSDB).

Graph:

Cette classe permet de faire une représentation mathématique de la topologie et d'en extraire, pour chaque point, le plus court chemin en manipulant les objets « Link » et « Point » avec l'algorithme de Dijkstra.

Constants:

Cette classe permet une gestion plus aisée des logs pour un affichage plus clair.

Etat de l'implémentation finale

L'applicatif est terminé et fonctionnel. Chaque routeur est capable de connaître ses voisins, de fournir des LSP garnis et de calculer le chemin le plus court d'un point à un autre dans la topologie. L'applicatif affiche des logs cycliquement pour savoir ce qu'il se passe sur le routeur. On peut y voir :

- Les HelloMessages sortants ou entrants (entrants que si le mode verbeux est activé) :

```
INFO   : Sending HELLO on Router [R1] ...
INFO   : Sending LSP on Router [R1] ...

INFO   : Router [R2] 192.168.4.1 on eth0 SEND     [HELLO]->     src=192.168.1.2,
dst=255.255.255.255,   proto=1,   payload=[HELLO[R=Router   [R2];   FROM=192.168.4.1;
ETH=eth0]]
INFO   : Router [R1] 192.168.1.1 on eth2 RCVE     [HELLO]<-     src=191.168.255.2,
dst=255.255.255.255,   proto=1,   payload=[HELLO[R=Router   [R2];   FROM=192.168.4.1;
ETH=eth3]]
INFO   : Router [R1] 192.168.1.1 on eth2 RCVE     [LSP]  <-     src=191.168.255.2,
dst=255.255.255.255,    proto=1,    payload=[LSP[FROM=192.168.4.1    ;    NUMSEQ=0    ;
ADJROUTER={{192.168.1.1=1}}]]
INFO   : Router [R1] 192.168.1.1 on eth0 SEND     [LSP]  ->     src=192.168.0.1,
dst=255.255.255.255,    proto=1,    payload=[LSP[FROM=192.168.4.1    ;    NUMSEQ=0    ;
ADJROUTER={{192.168.1.1=1}}]]
```

- Le calcul des chemins les plus courts pour une source vers toutes les destinations :

```
INFO   : Router [R1] calculating best routes ...
INFO    : Router [R1] has best route from 192.168.5.1-->192.168.3.1 : 192.168.5.1-
>192.168.1.1->192.168.4.1->192.168.3.1 cost 9
INFO    : Router [R1] has best route from 192.168.5.1-->192.168.5.2 : 192.168.5.1-
>192.168.5.2 cost 10
INFO    : Router [R1] has best route from 192.168.5.2-->192.168.4.1 : 192.168.5.2-
>192.168.3.1->192.168.4.1 cost 8
INFO    : Router [R1] has best route from 192.168.5.2-->192.168.1.1 : 192.168.5.2-
>192.168.3.1->192.168.4.1->192.168.1.1 cost 9
INFO   : The shortest distance from IP 192.168.4.1 to IP 192.168.4.1 is 0 with []
INFO   : The shortest distance from IP 192.168.4.1 to IP 192.168.1.1 is 1 with [Link
from 192.168.4.1 to 192.168.1.1 eth1  cost 1]
```

```
INFO   : The shortest distance from IP 192.168.4.1 to IP 192.168.5.1 is 2 with [Link
from 192.168.4.1 to 192.168.1.1 eth1  cost 1, Link from 192.168.1.1 to 192.168.5.1
eth1  cost 1]
INFO   : The shortest distance from IP 192.168.4.1 to IP 192.168.3.1 is 7 with [Link
from 192.168.4.1 to 192.168.3.1 eth0  cost 7]
```

- D'autres traces sont à voir si le mode verbeux est activé.

Difficultés rencontrées

Tout d'abord, la première difficulté était la compréhension et la sauvegarde de la LSDB. En effet, il faut pour cela bien comprendre la structure de cette base de données afin d'avoir un objet cohérent pour calculer les meilleures routes.

Ensuite, la gestion des AbstractTimer n'était pas des plus anodine. Sa manipulation n'est pas très explicite.

Enfin, l'algorithme de Dijkstra, toujours vu en mathématiques mais jamais implémenté, nous a pris quelques jours. Pour une meilleure compréhension de cet algorithme et de son debugging, les objets ont été nommés comme en cours de mathématiques avec des liens et des points. Un premier jet consistait à calculer pour un point hardcodé les meilleures routes pour chaque point de la topologie. Ensuite, après avoir calculé et affiché le coût total pour chaque point à partir du point fixe hardcodé, le but du jeu était d'enregistrer les points visités avec le coût vers ce point dans une liste, appelée « listOfCostsFromSrc ». Une fois cette liste intègre et correcte, notre but était de calculer pour chaque point les meilleures routes pour tous les autres points et de renvoyer le résultat à la classe principale « LSPRoutingProtocol ».

Changement apporté dans le simulateur

Aucun changement fonctionnel n'a été apporté au simulateur de base. Néanmoins, dans la classe « Datagram », se trouvant dans le package « ip », un espace est manquant à la ligne 44 entre la « , » et « proto= ». Dans le cas où l'applicatif est en mode verbeux, cette ligne est affichée et pour raison esthétique a été corrigée.