Reroutage dans OSPF avec des chemins de secours

Miklos Molnar, Miled Tezeghdanti

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Reroutage dans OSPF avec des chemins de secours

Miklós Molnár, Miled Tezeghdanti

N˚4340

Décembre 2001

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4340--FR

Reroutage dans OSPF avec des chemins de secours

Miklós Molnár*, Miled Tezeghdanti†

Thème 1 — Réseaux et systèmes
Projet ARMOR

Rapport de recherche n ° 4340 — Décembre 2001 — 24 pages

Résumé : Le protocole de routage RIP (Routing Information Protocol) a été largement utilisé dans les réseaux IP. D'une part parce qu'il est le premier protocole de routage développé pour Internet et d'autre part parce qu'il très facile à mettre en ?uvre puisqu'il est basé sur l'algorithme de Bellman-Ford. Avec l'expansion de l'Internet, les limites du protocole RIP, telles que le diamètre du réseau qui ne doit pas dépasser 16 routeurs et le temps de convergence qui est relativement élevé, ont apparues. Le protocole OSPF (Open Shortest Path First) a été conçu plus tard au sein de l'IETF (Internet Engineering Task Force) pour substituer le protocole RIP. OSPF a permis de résoudre tous les problèmes constatés avec RIP et en particulier il a permis de réduire nettement le temps de convergence. Actuellement, ce temps de convergence est à l'ordre de la minute. Malheureusement, ce temps de convergence n'est pas tolérable pour certaines applications temps réel. Dans ce rapport, nous proposons un mécanisme complémentaire à OSPF qui permet de réduire encore plus le temps de convergence du protocole via le calcul préalable d'un chemin de secours disjoint du premier chemin utilisé pour chaque destination possible dans le réseau. L'algorithme TDSP " Two Disjoint Shortest Paths " que nous avons proposé pour calculer deux chemins disjoints en un seul passage a une complexité de $O(n^2)$.

Mots-clé : routage, reroutage rapide, OSPF, tolérance aux fautes.

(Abstract: pto)

* Miklos.Molnar@irisa.fr
† Miled.Tezeghdanti@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Rerouting in OSPF Networks with Standby Paths

Abstract: The RIP (Routing Information Protocol) was largely used in IP networks. This is explained by the fact that RIP was the first routing protocol developed for Internet and it was a simple protocol which is based on the Bellman-Ford algorithm. With the expansion of Internet, RIP shortcomings such as 16 hop limit and its high convergence time become to appear. Later, the IETF (Internet Engineering Task Force) specified the OSPF (Open Shortest Path First) protocol to substitute RIP protocol. OSPF resolved all RIP problems and particularly it reduces with a significant amount the convergence time. Nowadays, OSPF convergence time is about 1 minute. However, this convergence time is not low enough to be tolerated by real time applications. In this report, we propose complementary mechanism to OSPF which reduces much more the protocol convergence time by calculating in advance for each possible destination in the network a backup path which is disjoint from the primary used path. The TDSP "Two Disjoint Shortest Paths" algorithm that we had proposed calculates two disjoint paths in one pass with a complexity of $O(n^2)$.

Key-words: routing, fast rerouting, OSPF, fault tolerance.

1 Introduction

Pour éviter certains inconvénients du protocole RIP (Routing Information Protocol) et pour gérer des réseaux plus complexes, le protocole OSPF (Open Shortest Path First) est de plus en plus utilisé comme protocole de routage interne dans le monde IP. Dans OSPF, chaque routeur diffuse sur le réseau les informations décrivant sa topologie locale. Cette diffusion permet à chaque routeur d'avoir la totalité de la topologie réseau. Les routeurs appliquent l'algorithme de Dijkstra sur la base de données topologique pour calculer les routes vers toutes les destinations possibles. Lorsqu'un routeur observe une modification de sa topologie locale (panne d'un lien directement connecté), il diffuse sur le réseau sa nouvelle configuration topologique, ce qui permet aux autres routeurs de mettre à jour leurs bases de données topologiques et de modifier leurs tables de routage si nécessaire. Le fait de diffuser instantanément sur le réseau uniquement les changements de la topologie quand ils sont constatés permet au protocole OSPF d'avoir un temps de convergence relativement petit par rapport au protocole RIP. Dans RIP, il s'agit d'envoyer la totalité de la table de routage à chaque 30 secondes. Un routeur se trouvant à une distance d (en nombre de sauts) du routeur qui a constaté la panne d'un lien ne sera averti de ce changement qu'au bout de $30d$ secondes ! Certes, l'apport du protocole OSPF en ce qui concerne l'amélioration du temps de convergence est précieux, mais malheureusement cet apport n'est pas suffisant pour certaines applications temps réel. En fait, le temps de convergence du protocole OSPF suite à une panne simple dépasse la minute. C'est pour cette raison que nous proposons ici un mécanisme complémentaire à OSPF pour diminuer le temps de réaction du réseau en cas de pannes. Ce temps de réaction est dû à plusieurs facteurs. Parmi eux, on peut citer en particulier : le temps mis par un routeur OSPF pour décider qu'un routeur voisin est en panne, la durée minimale entre deux calculs successifs de la table de routage, et le temps nécessaire pour calculer la table de routage à partir de la base de données topologique. Dans cette étude, nous nous proposons de réduire le temps de convergence du protocole OSPF, en ramenant à zéro les deux derniers facteurs à savoir la durée minimale entre deux calculs successifs de la table de routage et le temps de calcul de la table de routage. Notre proposition est basée sur le calcul préalable d'un chemin de secours disjoint du premier chemin utilisé pour chaque destination possible dans le réseau. En fait, nous avons proposé un algorithme (une version de l'algorithme de Dijkstra) qui permet de calculer les deux chemins disjoints en un seul passage avec une complexité de $O(n^2)$.

Ce rapport est organisé de la façon suivante. Dans la première section, nous commençons par un bref survol de l'algorithme utilisé actuellement dans OSPF pour calculer les plus courts chemins. La section 3 soulève la question des pannes et analyse les différentes possibilités pour assurer les communications malgré les défaillances. Les éléments importants de notre proposition se trouvent dans les sections 4 et 6. La section 4 présente l'algorithme qui calcule deux chemins disjoints pour chaque couple *(source, destination)* du réseau. Cet algorithme construit les chemins de la même façon que l'algorithme SPF utilisé actuellement dans OSPF. Différents cas d'utilisation des chemins calculés par cet algorithme sont analysés dans la section 5. La section 6 indique la préparation des tables de routage et le méchanisme de reroutage en cas de panne.

2 Protocole de routage à état de liens

Le protocole OSPF appartient à la famille des protocoles de routage à état de liens [3, 4]. À la différence de la famille de vecteur de distance basée sur l'algorithme de Bellman-Ford (protocole RIP), la famille à état de liens est basée sur l'algorithme de Dijkstra. En effet, les routeurs échangent entre eux les informations sur la topologie du réseau et exécutent l'algorithme de Dijkstra sur cette base de données topologique pour calculer les routes vers les destinations possibles du réseau. Les routes sélectionnées sont celles qui possèdent un coût minimal. Le coût d'une route donnée est la somme des métriques des liens qui la composent. Cette métrique peut être différente d'un réseau à l'autre. Par exemple, une métrique (un coût approché de l'utilisation d'une ligne) peut être établie de la façon suivante [1] :

$$C_{ij} = 108/bandepassante(enbit/s).$$

Dans un domaine d'OSPF, les routeurs possèdent les mêmes connaissances sur le réseau (sur le domaine). Pour simplifier dans la suite, on ne modélise que les routeurs du domaine. De cette façon, le réseau correspond à un graphe valué par des valeurs positives. Nous supposons qu'il est connexe mais qu'il n'est pas forcement métrique.

L'algorithme SPF calcule les distances d'un nœud de départ par rapport aux autres nœuds (destinations) du réseau. Il s'agit de l'algorithme bien connu de Dijkstra (cf. par exemple [2]). Dans les calculs, une destination peut être représentée par un triplet :

Nœud `{nom,`
`coût (distance du nœud de départ),`
`sortie (successeur du nœud de départ vers la destination) }`

Pour les calculs, on évaluera trois ensembles de *Nœuds* :

- PATH - qui contiendra les premiers plus courts chemins acceptés ;
- TENT - qui contiendra les candidats *Nœuds* non encore justifiés selon leur coût ;
- ADJ - qui contiendra les nœuds adjacents du nœud dernièrement sélectionné.

Supposons qu'un routeur A veut calculer les plus courts chemins vers les autres nœuds du réseau. L'algorithme de Dijkstra est basé sur un fait très simple : si l'on connaît les meuilleurs chemins utilisant au plus k successeurs du nœud de départ (c'est l'ensemble TENT de l'algorithme), et si l'on sélectionne le chemin le plus court de cet ensemble, on obtient un plus court chemin pour aller du nœud de départ à l'autre extrémité du chemin. Les plus courts chemins ainsi sélectionnés sont cumulés dans l'ensemble PATH.

Les grandes lignes de l'algorithme peuvent être résumées de la façon suivante :

Procédure SPF (Nœud A)

```
/* Initialisation */
Ensemble de Nœuds PATH = ∅;
Nœud NœudCourant = A ;
Ensemble de Nœuds TENT = adjacent(A,∅) ;
```

```
 /* Construction */
Tent que TENT n'est pas vide faire :
      Nœud NœudAncien = NœudCourant ;
      NœudCourant = distmin(TENT) ;
      PATH = PATH ⋃ {NœudCourant} ;
      TENT = TENT \ {NœudCourant} ;
      Ensemble de Nœuds ADJ = adjacent(NœudCourant,NœudAncien) ;
      TENT = TENT ⋃ filtre(ADJ,PATH,TENT) ;
fait ;
fin.
```

Dans cette description algorithmique, nous proposons la décomposition du traitement des nœuds en utilisant les méthodes :

adjacent(N,M) - qui retourne l'ensemble des nœuds adjacents de N privé de M;

distmin(TENT) - qui sélectionne le nœud de l'ensemble TENT qui possède la distance minimale du nœud de départ ;

filtre(ADJ,PATH,TENT) - qui retourne le sous-ensemble de ADJ contenant les nœuds qui ne sont pas encore présents dans PATH ni dans TENT avec un meilleur coût.

Un exemple numérique illustrant le fonctionnement de l'algorithme SPF se trouve dans [1].

La création des tables de routage à partir des plus courts chemins calculés par l'algorithme SPF est très simple. Si, dans chaque routeur rencontré, on dirige les messages vers le successeur sur le plus court chemin connu par le routeur, alors on réalise un acheminement de coût minimal. Cette propriété peut être facilement prouvée.

Propriété 2.1 *Pour réaliser le routage des messages par un plus court chemin d'une source s vers une destination t, dans chaque nœud rencontré, il suffit de transmettre les messages au successeur du nœud sur le chemin le plus court localement calculé vers t. Le chemin final ainsi emprunté peut être différent du chemin calculé par s mais il est aussi un des plus courts chemins de s à t.*

Preuve. Admettons que chaque nœud du réseau connaît un plus court chemin et de cette façon le successeur immédiat sur le chemin pour chaque destination. Soit s la source et t la destination d'une communication. Soit s_1 le successeur de s se trouvant sur le plus court chemin de s à t calculé dans s et soit s_2 le successeur de s_1 pour aller à t selon le plus court chemin calculé dans s_1. Supposons que dans chacun des nœuds atteints pour aller de s à t, les messages de s sont dirigés vers le successeur du plus court chemin localement calculé. De cette façon, on parcourt le chemin $(s,s_1,s_2,...,s_m,t)$. Par définition, le chemin élémentaire (s_m,t) est un plus court chemin de s_m à t. Supposons que le chemin (s_{m_1},s_m,t) appartenant à $(s,s_1,s_2,...,s_m,t)$ n'est pas un plus court chemin entre s_{m-1} et t. Selon sa définition, s_m se trouve sur un plus court chemin de s_{m-1} à t. Supposons, qu'il existe alors un plus court chemin (s_{m_1},s_m^*,t) qui est plus favorable que le chemin (s_{m_1},s_m,t). Cela est possible, ssi le

chemin (s_m^*,t) est plus court que le chemin élémentaire (s_m,t). La contradiction est triviale. Par induction, le chemin $(s,s_1,s_2,...,s_m,t)$ utilisé ne peut être qu'un plus court chemin de s à t. ■

Dans le cas de l'utilisation de l'OSPF, les tables de routage sont créées selon les successeurs immédiats sur les plus courts chemins calculés localement pour aller aux destinations potentielles.

3 Le problème du reroutage

Le protocole OSPF évite les problèmes liés à la diffusion des tables de routage de taille importante qu'on peut observer dans le cas d'utilisation de RIP. Les routeurs possèdent la même image du réseau et peuvent traiter immédiatement des requêtes sur la topologie du réseau. Mais dans le cas d'une panne, la réaction du système peut être relativement lente. Pour assurer les communications malgré la panne, le réseau doit transmettre certaines communications par des routes changées ; cette réaction produit un délai qui vient de deux facteurs :

- de la détection de pannes,
- de la modification des tables de routage.

Ici, nous supposons qu'un mécanisme nésessaire pour informer les routeurs d'une éventuelle panne existe et ne peut pas être modifié.

Pour agir aux pannes après leur indication, plusieurs solutions peuvent être envisagées.

Au moment de la détection de la panne d'un lien, on peut modifier la topologie et les routeurs peuvent recalculer les premiers plus courts chemins a l'aide de l'algorithme SPF. Rappelons ici, que la complexité de l'algorithme de Dijsktra est en $O(n^2)$. Dans ce cas, la détection de la panne est suivie par des calculs relativement coûteux et par la réécriture des tables de routage ; ce qui n 'est pas très favorable pour le temps de réponse.

Une autre solution peut être la modification des tables de routage selon des itinéraires supplémentaires calculés et stockés pour les cas où un des liens du plus court chemin utilisé habituellement tombe en panne. Nous proposons l'étude de faisabilité de cette deuxième procédure. Pour pouvoir composer une solution algorithmique, donnons les critères pour la sélection d'un deuxième itinéraire :

- Le chemin doit être disjoint du premier plus court chemin, afin qu'on puisse l'utiliser quel que soit le lien en panne.
- Parmi les chemins qui sont disjoints du premier plus court chemin, ce chemin peut être le premier plus court chemin. De cette façon, on maintient l'idée de OSPF pour choisir toujours le meilleur chemin parmi les possibilités connues.

Naturellement, la métrique permettant de comparer les chemins peut être une métrique additive quelconque acceptée par OSPF.

Dans la suite, nous présentons un algorithme simple qui calcule les premiers plus courts chemins disjoints pour chaque destination potentielle des messages.

4 Algorithme TDSP pour calculer les deux premiers chemins disjoints

Pour assurer un mécanisme plus tolérant aux pannes, nous proposons un algorithme qui calcule un deuxième chemin (un chemin de secours) pour toutes les destinations possibles. Etant donné que n'importe quelle partie d'un chemin utilisé pour une communication donnée peut tomber en panne, on cherche un chemin de secours entièrement disjoint du chemin emprunté qui est calculé par l'algorithme SPF. Pour cela, nous supposons que le graphe du réseau est 2-connexe : il n'existe pas de coupe correspondant à un seul lien dans le domaine. La plupart des réseaux (par exemple les réseaux maillés) vérifie cette contrainte. Si cela n'est pas le cas, le réseau peut être décomposé en deux ou plusieurs domaines 2-connexes qui sont connectés entre eux par des liens critiques. Chaque communication portant sur plusieurs domaines 2-connexes doit passer par des liens critiques qui ne sont pas remplaçables. La panne d'un tel lien critique coupe le réseau en deux parties et la communication n'est plus possible entre les parties.

Afin d'offrir un chemin alternatif pour les cas de panne, une deuxième condition est nécessaire. Nous allons voir qu'l est indispensable que les chemins proposés par les routeurs ne contiennent pas de coupe séparant les extrémités du chemin en question. Dans nos calculs, nous supposons que la suppression d'un plus court chemin ne coupe pas le graphe en deux parties isolées et qu'après la suppression du chemin les extrémités de la connexion ne se trouvent pas dans deux parties différentes (cf. encore les remarques à la fin de la section).

Remarque : Pour calculer un chemin de secours vers chacune des destinations, il existe une solution simple. Après le calcul des premiers plus courts chemins, on peut modifier l'image du réseau : on peut supprimer un par un les plus courts chemins calculés. Pour chaque topologie ainsi modifiée, un nouveau calcul (éventuellement avec l'algorithme SPF) donne un nouveau plus court chemin pour la destination concernée par la suppression. Naturellement, ce nouveau chemin est disjoint du chemin supprimé. Afin de calculer un chemin alternatif pour chaque destination du réseau, il faut modifier le réseau et relancer le calcul autant de fois que destinations (nœuds) existent. Si n indique le nombre de nœuds dans le réseau, la complexité de l'algorithme SPF est à $O(n^2)$. Sans compter le coût des modifications nécessaires de la topologie, la complexité de cette solution relève à $O(n^3)$.

Dans la suite, nous proposons une solution qui calculent les deux chemins disjoints en un seul passage et avec une compléxité plus favorable.

4.1 Algorithme de base

L'algorithme proposé est une extension de l'algorithme de Link State : la même technique de calcul (une version modifiée de l'algorithme de Dijkstra) peut être utilisée dans les routeurs. Pour représenter un nœud et son chemin depuis le nœud de départ dans l'algorithme, on utilisera la structure modifiée suivante :

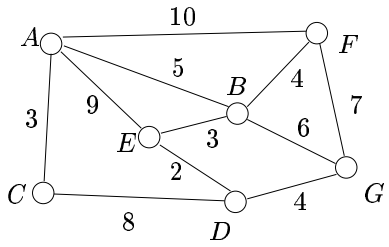

FIG. 1 – *Le réseau illustrant les exemples*

Nœud {nom,
coût (distance du nœud de départ),
chemin (liste des nœuds parcourus à partir du nœud de départ) }

Admettons que le routeur A du réseau illustré par la figure 1 calcule les chemins par l'algorithme ci-dessous. Dans ce cas, une représentation du nœud G corresponde à la structure : $G(14,BED)$ - ce qui indique que le nœud G est à 14 unités de A via le chemin $ABEDG$.

Pour chaque destination, l'algorithme TDSP (Two Disjoint Shortest Paths) proposé calcule les deux premiers plus courts chemins disjoints. Plus précisemment, il calcule le premier plus court chemin de secours disjoint du plus court chemin. L'algorithme d'origine de Dijkstra est un algorithme de fixation d'étiquettes : une fois une étiquette (un plus court chemin) est trouvée pour un nœud, cette étiquette ne change plus. Pour trouver deux chemins disjoints, on peut suivre l'algorithme de Dijksta mais avec l'objectif qu'on veut trouver deux étiquettes disjointes pour chaque nœud. Voici la squelette de l'algorithme modifié :

Procédure TDSP (Nœud A)

```
 /* Initialisation */
 Ensemble de Nœuds PATH = ∅;
 Nœud NœudCourant = A ;
 Ensemble de Nœuds TENT = adjacent(A,0) ;
 /* Construction */
Tant que TENT n'est pas vide faire :
      Nœud NœudAncien = NœudCourant ;
      NœudCourant = distmin(TENT) ;
      PATH = PATH ⋃ {NœudCourant} ;
      TENT = TENT \ {NœudCourant} ;
      Ensemble de Nœuds ADJ = adjacent(NœudCourant,NœudAncien) ;
      TENT = TENT ⋃ filtre(ADJ,PATH,TENT) ;
      TENT = filtre2(PATH,TENT) ;
fait ;
fin.
```

Pour associer deux étiquettes à chaque destination, certaines méthodes appelées dans cette procédure fonctionnent différemment que dans l'algorithme SPF :

adjacent(N,M) - pas de changement ; elle retourne toujours l'ensemble des nœuds adjacents de N privé de M ;

distmin(TENT) - sélectionne ici aussi le nœud de $TENT$ possédant la distance minimale du nœud de départ (en cas d'égalité des longueurs, dans un premier temps, on choisit par hasard le nœud retourné) ;

filtre(ADJ,PATH,TENT) - cette méthode maintient l'ensemble des étiquettes déjà trouvées mais pas définitivement associées aux nœuds. Ces étiquettes doivent être gardées d'une itération à l'autre dans l'ensemble $PATH$. Certaines étiquettes déjà découvertes par l'algorithme peuvent être jugées inintéressantes pour la suite. Les étiquettes déjà fixées dans $PATH$ sont choisies selon le critère d'optimalité de l'algorithme et ne changent plus. Le filtrage des étiquettes inutiles de l'union $TENT \bigcup ADJ$ est assuré par la méthode *filtre* en faisant des éliminations successives :

- **a)** si un nœud a déjà deux étiquettes dans $PATH$, toutes autres étiquettes de ce nœud doivent être supprimées ;
- **b)** si une étiquette de l'union $TENT \bigcup ADJ$ concerne un nœud présent dans $PATH$ et si les chemins associés aux deux occurences possédent une partie commune, alors l'étiquette de l'union $PATH \bigcup TENT$ doit être éliminée ;
- **c)** si l'union $PATH \bigcup TENT \bigcup ADJ$ contient trois étiquettes concernant un même nœud mais avec des chemins disjoints, alors l'occurrence de distance maximale des trois doit être retirée (en cas d'égalité des distances, une étiquette parmi les moins bonnes sera naturellement retirée) ;
- **d)** si après les filtrages a), b) et c), $TENT \bigcup ADJ$ contient encore deux étiquettes pour un même nœud et les chemins associés aux deux possèdent une partie commune, alors l'occurrence de distance maximale de deux doit être éliminée.

La nécessité du filtrage c) peut être illustrée à l'aide de l'exemple suivant. Admettons que $TENT$ contient déjà deux étiquettes d'un nœud (avec des chemins d'accès différents bien sûr) quand la troisième étiquette du nœud est proposée et ajoutée à partir de ADJ. Cette troisime version passe à travers des filtrages de type a) et b) si la distance de la nouvelle version est plus favorable. Par exemple : supposons que $TENT$ contient les structures $G(6)$ et $G(11,F)$ quand on calcule les nœuds adjacents de D. Ces derniers contiennent la représentation $G(9,ED)$. Naturellement pour la suite, il faut garder $G(6)$ et $G(9,ED)$ en éliminant $G(11,F)$. La condition d) du filtrage peut être elle aussi facilement justifiée.

Nous illustrons le déroulement de l'algorithme TDSP proposé par l'évolution des ensembles mentionnés. Au titre d'exemple, suivons les calculs des deux premiers chemins disjoints pour la base de données du routeur A de la figure 1.

PATH	TENT	ADJ
0	F(10),B(5),E(9),**C(3)**	D(11,C)
C(3) C(3), B(5)	F(10),**B(5)**,E(9),D(11,C) F(10),E(9),D(11,C), **E(8,B)**, F(9,B),G(11,B)	E(8,B),F(9,B),G(11,B) D(10,BE)
C(3),B(5),E(8,B)	F(10),**E(9)**,D(11,C),F(9,B), G(11,B),D(10,BE)	*D(11,E)*,B(12,E) (règle c)
C(3), B(5),E(8,B),E(9)	F(10),D(11,C),**F(9,B)**, G(11,B),D(10,BE),B(12,E)	*G(16,FB)* (règle d)
C(3), B(5),E(8,B),E(9), F(9,B)	**F(10)**,D(11,C),G(11,B), D(10,BE),B(12,E)	*B(14,F)*,G(17,F) (règle c)
C(3),B(5),E(8,B),E(9), F(9,B),F(10)	D(11,C),G(11,B),**D(10,BE)**, B(12,E),G(17,F)	C(18,BED),*G(14,BED)* (règle c)
C(3),B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE)	**D(11,C)**,G(11,B),B(12,E), *G(17,F)*,C(18,BED)	*E(13,CD)*,G(15,CD) (règle a) (règle c)
C(3),B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE), D(11,C)	**G(11,B)**,B(12,E),C(18,BED), G(15,CD)	*D(15,BG),F(19,BG)* (règle a)
C(3),B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE), D(11,C),G(11,B)	**B(12,E)**,C(18,BED),G(15,CD)	*F(16,EB),G(18,EB)* (règle a) (règle b)
C(3),B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE), D(11,C),G(11,B),B(12,E)	C(18,BED),**G(15,CD)**	*B(21,CDG), F(22,CDG)* (règle a)
C(3),B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE), D(11,C),G(11,B),B(12,E), C(15,CD)	**C(18,BED)**	
C(3), B(5),E(8,B),E(9), F(9,B),F(10),D(10,BE), D(11,C),G(11,B),B(12,E), C(18,BED),G(15,CD)		

Si chaque routeur calcule les deux premiers chemins disjoints pour chaque destination présente dans le réseau illustré sur la figure 1, on obtient les chemins comme ci-dessous :

	A	B	C	D	E	F	G
A		B(5) B(12,E)	C(3) C(18,BED)	D(10,BE) D(11,C)	E(8,B) E(9)	F(9,B) F(10)	G(11,B) G(15,CD)
B	A(5) A(12,E)		C(8,A) C(13,ED)	D(5,E) D(10,G)	E(3) E(12,GD)	F(4) F(13,G)	G(6) G(9,ED)
C	A(3) A(18,DEB)	B(8,A) B(13,DE)		D(8) D(13,ABE)	E(10,D) E(11,AB)	F(12,AB) F(19,DG)	G(12,D) G(14,AB)
D	A(10,EB) A(11,C)	B(5,E) B(10,G)	C(8) C(13,EBA)		E(2) E(13,GB)	F(9,EB) F(11,G)	G(4) G(11,EB)
E	A(8,B) A(9)	B(3) B(12,DG)	C(10,D) C(11,BA)	D(2) D(13,BG)		F(7,B) F(13,DG)	G(6,D) G(9,B)
F	A(9,B) A(10)	B(4) B(13,G)	C(12,BA) C(19,GD)	D(9,BE) D(11,G)	E(7,B) E(13,GD)		G(7) G(10,B)
G	A(11,B) A(15,DC)	B(6) B(9,DE)	C(12,D) C(14,BA)	D(4) D(11,BE)	E(6,D) E(9,B)	F(7) F(10,B)	

4.2 Elimination des chemins contenants une coupe

Pour pouvoir utiliser l'algorithme TDSP, nous avons supposé que les premiers plus courts chemins ne contiennent pas de coupe et de cette façon on peut toujours calculer les deuxièmes chemins. Le lemme suivant illustre l'importance de cette condition.

Lemme 4.1 *Soit $G = (V,E,d)$ le graphe valué d'un réseau connexe (ou encore 2-connexe). Soient s et t deux nœuds de V et soit p un chemin de s à t contenant une coupe qui sépare s et t. Dans ce cas, il n'existe pas de chemin disjoint de p dans G entre s et t.*

Preuve. Si l'on enlêve les arêtes de p, on obtient un graphe avec au moins deux composants connexes : un des composants contient s et l'autre contient t. Chaque chemin entre s et t doit passer par au moins une arête qui appartient à la coupe qui sépare les deux parties. En conséquence : il n'existe pas de chemin entre s et t qui n'a pas d'arête commune avec p. ■

La figure 2 présente un réseau où le plus court chemin de s à t correspond à une coupe séparant les nœuds s et t. En conséquence dans ce réseau, il n'existe pas de chemin disjoint de ce chemin (s,a,b,t) pour aller de s à t. Si l'on veut assurer deux chemins disjoints de s à t pour améliorer la tolérance aux fautes du réseau, il faut abandonner l'objectif que le premier chemin sélectionné soit un plus court chemin.

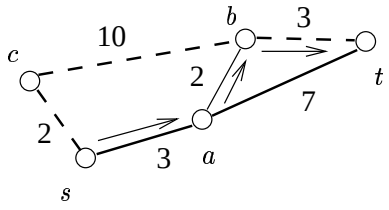

FIG. 2 – *Exemple d'un plus court chemin (S,T) coupant le réseau en deux*

Par définition dans un graphe 2-connexe, il existe toujours deux chemins disjoints entre deux sommets différents quelconques. Si, dans l'algorithme TDSP, on n'accepte pas des chemins contenant une coupe qui sépare les extrémités du chemin en question, alors on obtient deux chemins disjoints pour chaque destination mais les chemins ne sont pas forcement les plus courts. Avec cette modification, l'algorithme TDSP fournit toujours deux solutions par destination dans les réseaux 2-connexes. Pour obtenir cette version plus robuste de l'algorithme TDSP, on peut rajouter un filtrage supplémentaire à la méthode *filtre*:

filtre(ADJ,PATH,TENT) - cette méthode doit aussi appliquer la règle:

e) une étiquette d'un nœud dont le chemin associé contient une coupe séparant les extrémités du chemin doit être éliminé.

Si le premier chemin accepté par l'algorithme n'est pas un des plus courts chemins, la propriété 2.1 n'est plus vérifiée. Malgré cette perte, le routage est facilement réalisable et nous allons voir que la performance du réseau reste acceptable.

4.3 Suppression des oscillations

Si les chemins utilisés pour la création des tables de routage sont tous des plus courts chemins, la propriété 2.1 assure que les chemins réellement utilisés pour l'acheminement des messages sont aussi des plus courts chemins. De cette façon, ils sont exemptes de boucles (un plus court chemin ne contient pas de boucle). Dans le cas où certaines entrées des tables de routage sont initialisées par des chemins plus longs que les plus courts chemins, il faut examiner la possibilité des oscillations. Le lemme suivant nous assure que, pour l'acheminement de base des messages, les chemins fournis par la version modifié de l'algorithme TDSP ne cause pas d'oscillation.

Lemme 4.2 *Si les deux premiers chemins disjoints sont calculés par l'algorithme TDSP modifié et les tables de routage sont initialisées selon les premiers chemins proposés, alors il n'y a pas des oscillations au niveau de l'acheminement des messages.*

Preuve. Soit p_1 le plus court chemin pour aller de s à t. Supposons que ce chemin contient une coupe qui sépare s et t. Soit p_2 le premier chemin exempte de coupe de s à t et soit v le successeur de s sur p_2. Supposons qu'il y a des oscillations en réalisant

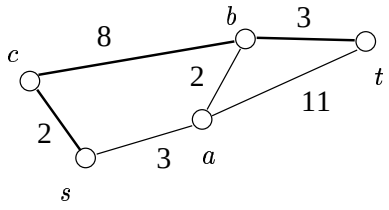

FIG. 3 – *Le plus court chemin* (s,t) *coupant le réseau en deux*

l'acheminement des messages de s à destination de t selon les chemins de base calculés à l'aide de l'algorithme TDSP. Prenons le chemin utilisé de v à t qui a comme premier nœud traversé le nœud s. Supposons que ce chemin est un plus court chemin. Dans ce cas, il ne peut être que le chemin $\{(v,s)\} + p_1$ (p_1 étant le plus court de s À t). De l'autre côté, ce chemin doit être refusé par la version modifiée de TDSP (il contient une coupe qui est p_1). Supposons qu'il existe un chemin optimal $\{(v,s)\} + p_3$ où p_3 ne contient pas de coupe entre v et t. Il est optimal, ssi $d(\{(v,s)\} + p_3) < d(p_2 - \{(s,v)\})$. De cette façon : $d(p_3) < d(p_2)$ ce qui contredit à la définition de p_2 (il est le plus court chemin de s à t sans coupe). ■

Pour illustrer l'absence des oscillations, nous proposons l'exemple du réseau de la figure 3. Ici, le plus court chemin de s à t est le chemin (s,a,b,t) qui n'est pas accepté par l'algorithme TDSP modifié selon ci-dessus. Le premier chemin accepté de s à t est le chemin (s,c,b,t). De s à t, les messages seront routés dans un premier temps vers c. Dans c, le premier plus court chemin pour aller à t est le chemin (c,s,a,b,t), mais ce chemin ne peut pas être accepté non plus par le même algorithme TDSP à cause de la coupe (s,a,b,t). Le chemin accepté de c à t est le chemin (c,b,t) et les messages de s seront routés vers c puis vers b pour aller à t.

Malheureusement, si les routeurs calculent leurs bases de données indépendamment, des oscillations sont possibles à partir du moment qu'une panne se produit et les chemins de secours sont utilisés dans une partie du réseau. Pour illustrer le problème, prenons le réseau de la figure 4. Supposons que l'algorithme TDSP fournit les solutions suivantes pour aller de a et de s à t :

	t
a	t(5,scdt) t(7,ab)
s	t(4,d) t(7,ab)

Supposons que le lien (s,d) tombe en panne. Le premier chemin proposé par s est touché par cette panne mais le premier chemin proposé par a de a à t non. De cette façon, les messages de s à destination de t partent vers a mais a les renvoie vers s.

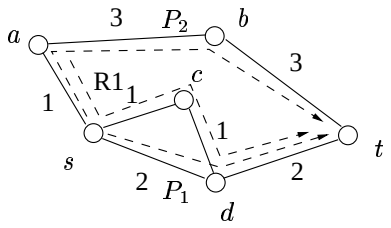

FIG. 4 – *Oscillations en cas de panne*

Le problème vient de l'indépendance des calculs des chemins dans les différents nœuds. Ici, il existe deux plus courts chemins pour aller de s à t. Le nœud s propose l'un des deux et a calcule un chemin qui contient l'autre. Si l'on pourrait assurer la cohérence des chemins calculés par les routeurs, alors on pourrait éliminer les boucles. On peut calculer des chemins différents pour un couple (*source*,*destination*) par l'algorithme TDSP, si plusieurs chemins de même longueur et de même propriété existent entre les nœuds en question. Parmi les candidats possibles, c'est la fonction *distmin* de l'algorithme TDSP qui choisit le meuilleur chemin pour le routage et pour le secours. Si, en cas d'égalité de longueur de plusieurs chemins, cette fonction choisit toujours le même chemin, on peut éviter les oscillations. Un même choix parmi les chemins de longueur égale peut être assurer si les nœuds (les successeurs des nœuds et tous les chemins) sont triés partout et si l'on choisi toujours le premier chemin en cas d'égalité. Pour trier des chemins, on peut utiliser les mots composés des adresses IP des routeurs traversés par le chemin. De cette façon, en cas d'égalité des chemins, la fonction *distmin* retourne toujours la même solution.

Sur notre exemple : supposons que l'adresse IP de d est inférieure à celle de c. Quand s calcule son premier chemin à d, il compare les deux chemins (s,d) et (s,c,d). La comparaison "mnémonique" favorise le premier. Quand à a, il doit aussi comparer les chemins (a,s,d) et (a,s,c,d). Pour la même raison, ce nœud choisit aussi le chemin (a,s,d). En conséquence : il n'y a pas de deux passages différents entre s et d.

4.4 Propriétés des chemins calculés

Soit s une source et t une destination comme le montre la figure 5. Soit P_1 le plus court chemin et P_2 le premier chemin disjoint de P_1 entre s et t (pour simplifier, admettons que P_2 existe). Appelons v_1 et v_2 les successeurs de s en passant par P_1 et P_2 (en utilisant l'arête p_{11} ou p_{21}) respectivement. Désignons par P_{12} et P_{22} les chemins restants à partir de v_1 et de v_2 respectivement pour aller à t.

Propriété 4.3 *Le chemin P_{12} (appartenant au plus court chemin P_1 entre s et t) est un plus court chemin de v_1 à t.*

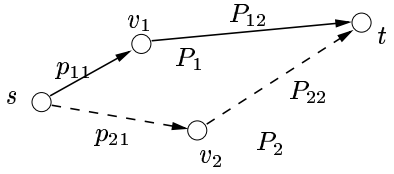

FIG. 5 – *Les deux premiers plus courts chemins disjoints de s à t*

Preuve. Supposons que P_{12} n'est pas un plus court chemin de v_1 à t. Soit P_{12}^* le plus court chemin. En passant par $\{p_{11}\} + P_{12}^*$, on obtient un chemin plus court que P_1, ce qui contredit à la définition de P_1. ■

Propriété 4.4 *Si le chemin P_{22} n'est pasun plus court chemin pour aller de v_2 à t, alors il existe un plus court chemin P_3 de v_2 à t, tel que $(P_3 \bigcap P_1)$ n'est pas vide.*

Preuve. Supposons que le plus court chemin de v_2 à t n'est pas le chemin P_{22} mais un chemin P_3. Naturellement, $d(P_3) < d(P_{22})$. Supposons que P_3 est disjoint de P_1. Dans ce cas, le chemin $\{p_{12}\} + P_3$ est un chemin disjoint de P_1 et il est plus court que P_2, ce qui contredit la définition de P_2 (celui dernier est le premier chemin disjoint de P_1). ■

4.5 Complexité de l'algorithme

En utilisant un filtrage particulier, l'algorithme TDSP cherche les deux premières étiquettes qu'on peut associer aux nœuds du réseau. L'algorithme se termine, quand l'ensemble $TENT$ devient vide : quand les deux étiquettes sont déjà trouvées pour chaque nœud. De cette façon, la boucle principale de l'algorithme est exécutée $2n$ fois.

Soit d la valeur maximale des degrés des nœuds dans le graphe. Au cours des calculs, la cardinalité des ensembles $PATH$ et $TENT$ reste en $O(n)$. De cette façon :

- La sélection du nœud de distance minimale par la fonction *shortest* est en $O(n)$ (cf. [2]).
- Le nombre des nœuds adjacents du nœud traité est borné par d.
- Les filtrages *filtre* et *filtre2* sont en $O(dn)$.

La complexité du corps du boucle est ainsi en $O(n + d + dn)$.

Si le graphe est peu dense, d est relativement petit. Dans ce cas, la complexité de l'algorithme TDSP est en $O(n^2)$.

5 Etudes comparatives des politiques de routage/reroutage

En cas de panne, la réaction des routeurs du réseau dotés par un chemin de secours pour les différentes destinations peut être basée sur des politiques différentes. Dans cette section, nous proposons une étude comparative de plusieurs stratégies de reroutage des messages

concernés. Dans un premier temps, nous proposons l'anayse des coûts des différentes solutions en régime "stationnaire" ; dans les comparaisons ci-dessous, on fait abstraction des phénomènes de la transition du régime de base à la solution de secours.

On va examiner deux cas de panne dans le réseau présenté sur la figure 1. Pour simplifier, supposons qu'un seul lien tombe en panne à la fois. Supposons ensuite que les deux chemins disjoints sont calculés dans chaque nœud et pour chaque destination.

Nous proposons la comparaison de trois gestions différentes :

Première stratégie : Le premier gestionnaire veut que les messages émis par les sources suivent exactement le chemin prévu par la source. (C'est possible par exemple, si la source insère le chemin à suivre dans l'entête des messages.) Au moment de l'avertissement de la panne, les sources vérifient si le lien en panne appartient ou non aux différents chemins proposés en premier pour le routage. Si oui, elles changent immédiatement les chemins touchés par la panne et prennent les chemins de secours calculés préalablement vers les mêmes destinations. De cette façon, les sources insèrent un chemin de secours dans l'entête si celui est nécessaire à cause de la panne.

Deuxième stratégie : En cas de panne d'un lien, seules les extrémités du lien en question changent leur table de routage. Elles remplacent le lien en panne par l'itinéraire calculé à l'aide de l'algorithme TDSP. Par exemple, si le lien $A - C$ tombe en panne, on utilisera le chemin A,B,E,D,C et les passages $C - A$ seront assurés par le chemin C,D,E,B,A. La table de routage des autres routeurs ne change pas. Par exemple, les messages de B à C seront dirigés d'abord vers A puis vers l'itinéraire A,B,E,D,C .

Troisième stratégie : Quand l'avertissement de la panne arrive dans les routeurs, chaque routeur change sa table de routage, si nécessaire. Plus précisément : une sortie de la table de routage change, si le premier chemin utilisé pour le routage des messages contient le lien en panne. Dans ce cas, la nouvelle sortie indiquera le successeur du routeur vers la destination en utilisant le chemin de secours. L'acheminement des messages est assuré indépendamment dans les routeurs selon le contenu de leur table de routage.

Pour faciliter la comparaison des coûts, supposons qu'il existe une communication entre chaque paire de nœuds dans le réseau de la figure 1 et que la quantité de message est identique pour chaque communication. Le coût de la transmission des messages soit proportionnel à la "longueur" des chemins parcourus.

5.1 Le lien A-C tombe en panne

Les longueurs et les coûts des chemins utilisés par la première solution en régime de base et en cas de panne sont les suivants:

Communication	**Routage de base**	**En cas de panne**
A,C	A,C : 3	A,B,E,D,C : 18
B,C	B,A,C : 8	B,E,D,C :13
C,A	C,A : 3	C,D,E,B,A : 18
C,B	C,A,B : 8	C,D,E,B : 13
C,F	C,A,B,F : 12	C,D,G,F : 19
F,C	F,B,A,C : 12	F,G,D,C : 19
	Total : 46	Total : 100

La deuxième stratégie donne les résultats suivant :

Communication	**Routage de base**	**En cas de panne**	
A,C	A,C : 3	A,B,E,D,C : 18	
B,C	B,A,C : 8	B,A,B,E,D,C :18	*
C,A	C,A : 3	C,D,E,B,A : 18	
C,B	C,A,B : 8	C,D,E,B : 13	**
C,F	C,A,B,F : 12	C,D,E,B,A,B,F : 27	*
F,C	F,B,A,C : 12	F,B,A,B,E,D,C : 27	*
	Total : 46	Total : 123	

- * Ici, on suppose que le gestionnaire du réseau n 'est pas capable de détecter la boucle causée
- ** Ici, le parcours est simplifié, parce que les messages arrivent à la destination.

En utilisant la troisième stratégie, on obtient les longueurs (les coûts) suivantes :

Communication	**Routage de base**	**En cas de panne**
A,C	A,C : 3	A,B,E,D,C : 18
B,C	B,A,C : 8	B,E,D,C :13
C,A	C,A : 3	C,D,E,B,A : 18
C,B	C,A,B : 8	C,D,E,B : 13
C,F	C,A,B,F : 12	C,D,E,B,F : 17
F,C	F,B,A,C : 12	F,G,D,C : 19
	Total : 46	Total : 98

5.2 Le lien E-B tombe en panne

Quand le lien E_B tombe en panne, la première stratégie implique les coûts suivants :

Communication	**Routage de base**	**En cas de panne**
A,D	A,B,E,D : 10	A,C,D : 11
A,E	A,B,E : 8	A,E : 9
B,D	B,E,D : 5	B,G,D : 10
B,E	B,E : 3	B,G,D,E : 12
D,A	D,E,B,A : 10	D,C,A : 11
D,B	D,E,B : 5	D,G,B : 10
D,F	D,E,B,F : 9	D,G,F : 11
E,A	E,B,A : 8	E,A : 9
E,B	E,B : 3	E,D,G,B : 12
E,F	E,B,F : 7	E,D,G,F : 13
F,D	F,B,E,D : 9	F,G,D : 11
F,E	F,B,E : 7	F,G,D,E : 13
	Total : 84	Total : 132

Deuxième stratégie : on remplace uniquement le lien en panne par le deuxième chemin disjoint proposé par la table de routage. Ici, à la place du lien $E-B$, on utilisera le chemin E,D,G,B et les passages $B-E$ seront redirigés vers le chemin B,G,D,E.

Communication	**Routage de base**	**En cas de panne**	
A,D	A,B,E,D : 10	A,B,G,D : 15	**
A,E	A,B,E : 8	A,B,G,D,E : 17	
B,D	B,E,D : 5	B,G,D : 10	**
B,E	B,E : 3	B,G,D,E : 12	
D,A	D,E,B,A : 10	D,E,D,G,B,A : 19	*
D,B	D,E,B : 5	D,E,D,G,B : 14	*
D,F	D,E,B,F : 9	D,E,D,G,B,F : 16	*
E,A	E,B,A : 8	E,D,G,B,A : 17	
E,B	E,B : 3	E,D,G,B : 12	
E,F	E,B,F : 7	E,D,G,B,F : 16	
F,D	F,B,E,D : 9	F,B,G,D : 14	**
F,E	F,B,E : 7	F,B,G,D,E : 16	
	Total : 84	Total : 178	

Comme dans le cas précédant,

- * Ici, on suppose que le gestionnaire du réseau n 'est pas capable de détecter la boucle causée
- ** Ici, le parcours est simplifié, parce que les messages arrivent à la destination.

Troisième stratégie :

Communication	**Routage de base**	**En cas de panne**
A,D	A,B,E,D : 10	A,C,D : 11
A,E	A,B,E : 8	A,E : 9
B,D	B,E,D : 5	B,G,D : 10
B,E	B,E : 3	B,G,D,E : 12
D,A	D,E,B,A : 10	D,C,A : 11
D,B	D,E,B : 5	D,G,B : 10
D,F	D,E,B,F : 9	D,G,F : 11
E,A	E,B,A : 8	E,A : 9
E,B	E,B : 3	E,D,G,B : 12
E,F	E,B,F : 7	E,D,G,F : 13
F,D	F,B,E,D : 9	F,G,D : 11
F,E	F,B,E : 7	F,G,D,E : 13
	Total : 84	Total : 132

Cette fois-ci, la première et la troisième stratégie donnent le même résultat.

Déjà cette petite étude montre que les chemins disjoints calculés par l'algorithme TDSP peuvent servir le reroutage rapide des messages en cas de panne : les surcoûts restent maîtrisables dans les trois cas présentés. Cependant, on peut également constater que les itinéraires calculés pour remplacer un seul lien (la stratégie 2) sont moins intéressants du point de vue coût et organisation. La première stratégie nécessite le stockage du chemin à parcourir dans l'entête des messages et l'analyse de l'entête à chaque routeur ; ceux qui rendent la première solution difficile et lourde. C'est pourquoi dans la suite, nous retenons l'idée du reroutage basé sur les chemins disjoints calculés pour chaque couple *(source, destination)* où chauque routeur décide seul s'il change sa table de routage ou non.

6 Reroutage basé sur les chemins de secours calculés en avance

Afin d'améliorer la tolérance aux fautes des réseaux utilisants OSPF, nous proposons l'application de la troisième politique. Cette politique suppose que les bases de données des routeurs contiennent un chemin de secours disjoint du chemin utilisé pour le routage normal des messages. Dans la méthode proposée, les routeurs agissent indépendamment, lors de l'arrivé d'un message indiquant la panne, ils changent leur table de routage selon le contenu de leur propre base de données et remplacent la sortie utilisée pour une destination si nécessaire.

Les avantages de cette proposition et de l'application de l'algorithme TDSP pour assurer un deuxième chemin disjoint pour les communications sont les suivantes :

- en recevant un avertissement d'une panne, les routeurs ne doivent pas recalculer les chemins utilisés pour modifier les tables de routage ;

– le calcul d'un deuxième chemin disjoint n'augmente pas considérablement le coût de la préparation de la base de données des routeurs (cf. l'anayse de la complexité de l'algorithme).

En contrepartie, la méthode proposée nécessite des bases de données plus volumineuses dans les routeurs. Cette augmentation des bases de données, vu la taille des domaines typiques utilisants OSPF, ne remet pas en cause l'applicabilité de notre proposition.

6.1 Le mécanisme du reroutage

Initialement, les tables de routage créées selon les premiers chemins proposés. En cas de la panne d'un lien, les routeurs sont informés (plus ou moins rapidement) de la panne. Si la panne touche des chemins supposés utilisés par un routeur (selon le témoignage de la base de données du routeur), le routeur en question remplace les sorties vers les destinations par les successeurs des deuxièmes chemins de sa base de données. Dans la suite, le routeur transmet les messages vers chaque destination touchée par la panne selon le chemin de secours. Cette même politique est appliquée dans chaque nœud rencontré en passant vers la destination.

Dans la section 4, nous avons constaté que les propriétés des différents chemins proposés ne sont pas les mêmes ; il faut distinguer et analyser deux cas. Si les tables de routage ne contiennent que des entrées calculées selon un des plus courts chemins, l'acheminement final des messages se passe toujours par un plus court chemin. Le routage dans l'état normal du réseau est similaire à celui d'OSPF. C'est aussi le cas quand on calcule les chemins à l'aide de l'algorithme TDSP dans un réseau maillé où aucun plus court chemin ne correspond pas à une coupe. En gardant le même mécanisme de routage, le résultat est différent pour les cas où les sorties des tables de routage ne sont pas forcement calculées sur la base des plus courts chemins. Dans la suite, nous analysons la relation entre chemins proposés et chemins utilisés dans les deux cas.

6.1.1 Routage quand les premiers chemins sont les plus courts chemins

Dans l'état normal du réseau, les messages sont transmis selon les plus courts chemins calculés localement dans les nœuds. Pour un couple (source, destination), un exemple simple est illustré par la figure 6. En état de fonctionnement normal, les messages de la source s à la destination t parcourent un plus court chemin. Dans notre cas, il existe deux chemins avec la longueur minimale entre s et t : les chemins (s,a,e,f,t) et (s,a,b,c,d,t). Supposons que dans les routeurs, les nœuds adjacents sont toujours triés de la même façon : pour simplifier, l'ordre des nœuds est donné par l'ordre alphabétique de leurs noms. En conséquence, la même relation d'ordre existe sur l'ensemble des noms des chemins. La source s utilise le chemin (s,a,b,c,d,t) pour envoyer ses messages à t puisque ce mot précède le mot (s,a,e,f,t). Les messages de s sont dirigés vers le nœud a qui est le successeur immédiat de s sur le chemin. Etant donné que la relation d'ordre entre (a,b,c,d,t) et (a,e,f,t) donne exactement le même résultat que dans s, le routeur a envoie les messages reçus vers la sortie b, etc. Les messages suivent exactement le chemin prévu par s. Les deux chemins proposés par l'algorithme TDSP pour la source s sont indiqués sur la figure 6.a).

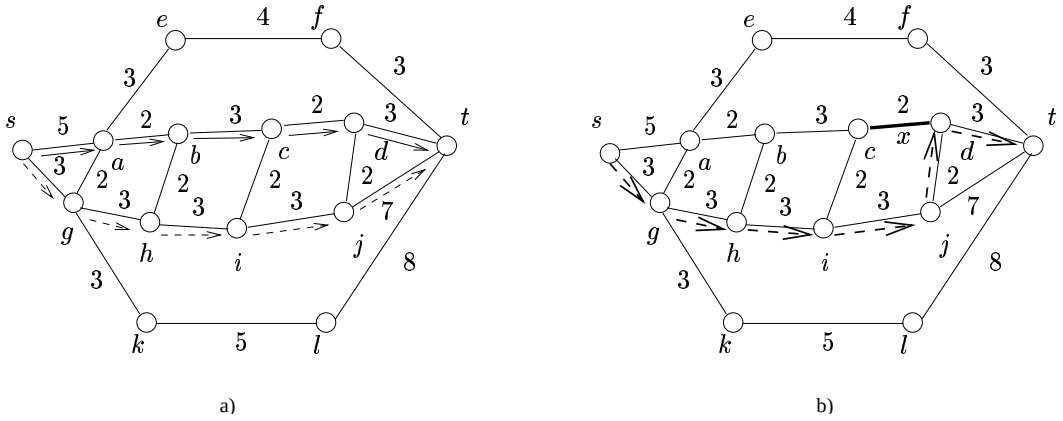

FIG. 6 – *Routage et reroutage de s à t*

Supposons que le lien $x = (c,d)$ tombe en panne (cf. la figure 6.b). Après avoir pris connaissance de cette panne, la source s (comme les autres routeurs du réseau) change sa table de routage et envoie les messages à t vers la sortie g en utilisant le chemin de secours disjoint (s,g,h,i,j,t). A partir de g, les messages à destination de t doivent emprunter le chemin de secours créé localement, puisque le plus court chemin (g,a,b,c,d,t) est aussi touché par la panne. Le chemin de secours de g à t est le chemin (g,h,i,j,d,t), donc les messages sont envoyés vers h puis vers i et vers j. Dans le routeur j, la situation change. Ici, le plus court chemin (j,d,t) n'est pas concerné par la panne de x ; ce chemin sert pour terminer l'acheminement des messages de s à t par la voie du chemin de secours.

En résumé : Les chemins de secours calculés et retenus par l'algorithme TDSP dans les routeurs et les chemins réellement utilisés en cas de panne peuvent être différents. Le phénomène est bien connu dans OSPF, il est observé en utilisant les plus courts chemins localement calculés.

6.1.2 Routage quand le premier chemin n'est pas un plus court chemin

Dans le cas où le premier plus court chemin d'une source à une destination correspond à une coupe, l'algorithme TDSP le refuse pour pouvoir trouver deux chemins disjoints. Il est possible que le premier chemin accepté pour le routage de base est plus long et que la propriété 2.1 n'est plus vérifié : le chemin proposé pour le routage de base n'est pas un plus court chemin. Le routeur successeur de la source sur ce chemin calcul aussi ses deux chemins indépendamment ; si le reste du chemin refusé dans la source n'est plus une coupe, il est possible que la route finalement appliquée corresponde au premier plus court chemin refusé par la source. Ainsi, le deuxième chemin proposé par TDSP pour la source et qui est sensé être disjoint du premier chemin *(source, destination)* n'est pas disjoint de la route réellement parcourue dans le cas du routage normal. On peut également constater que la différence entre

les chemins prévus dans la base de données des routeurs et les chemins réellement utilisés peut impliquer des fausses alertes.

Pour illustrer ces phénomènes, prenons le réseau de la figure 7. Si l'on enlève le premier plus court chemin de s à t du résau, on obtient deux composants connexes tels que s est séparé de t; l'algorithme TDSP n'accepte pas ce chemin pour la base de données de s. Le premier chemin accepté est le chemin (s,a,t) ce qui permet de calculer un chemin de secours (s,c,b,t). La matrice ci-après indique les routes proposées par l'algorithme TDSP pour chaque routeur.

	a	b	c	s	t
a		b(2) b(10,t)	c(5,s) c(12,b)	s(3) s(14,bc)	t(5,b) t(7)
b	a(2) a(10,t)		c(7,a,s) c(10)	s(5,a) s(12,c)	t(3) t(9,a)
c	a(5,s) a(12,b)	b(7,sa) b(10)		s(2) s(15,ba)	t(12,sa) t(13,b)
s	a(3) a(14,cb)	b(5,a) b(12,c)	c(2) c(15,ab)		t(10,a) t(15,cb)
t	a(5,b) a(7)	b(3) b(9,a)	c(12,as) c(13,b)	s(10,a) s(15,bc)	

Analysons les réactions du réseau pour acheminer/rediriger les messages de s à t dans plusieurs cas de figure.

Routage normale : Les messages partent de s vers a ce qui correspond au premier chemin (s,a,t) accepté par s. Le nœud a transmet les messages à t par le chemin (a,b,t) parce qu'il est le plus court et il ne contient pas de coupe pour le couple (a,t). Finalement, les messages de s utilisent le chemin (s,a,b,t) qui n'a pas été accepté pour la base de données de s.

Panne du lien (s,a) *:* Dans la source s, la table de routage change pour t. Les messages sont envoyés vers c selon le deuxième chemin calculé. Etant donné que la table de routage de c change aussi à cause de la même panne, les messages parcourent le chemin (s,c,b,t).

Panne du lien (a,b) *ou du lien* (b,t) *:* Pour la source, il n'y a pas de changement. Elle envoie les messages vers a. Le routeur a, sachant que le liens b,t n'est plus accessible, utilise le chemin de secours pour aller à t; c'est le lien (a,t). Les messages parcourent le chemin (s,a,t).

Panne du lien (a,t) *:* Malgré le fait que le chemin réellement utilisé pour transmettre les messages de s à t ne contient pas ce lien, la table de routage de s change pour la destination t. Les messages sont envoyés vers c selon le deuxième chemin calculé. Ici aussi, la table de routage de c change à cause de la panne; le chemin final correspond au chemin (s,c,b,t). On peut considérer ce cas comme une fausse alerte.

Nous soulignons de nouveau que dans les réseaux maillés la connexité est relativement élevée et cette situation est rare (celle-ci peut caractériser les petits réseaux ne contenant que quelques liens).

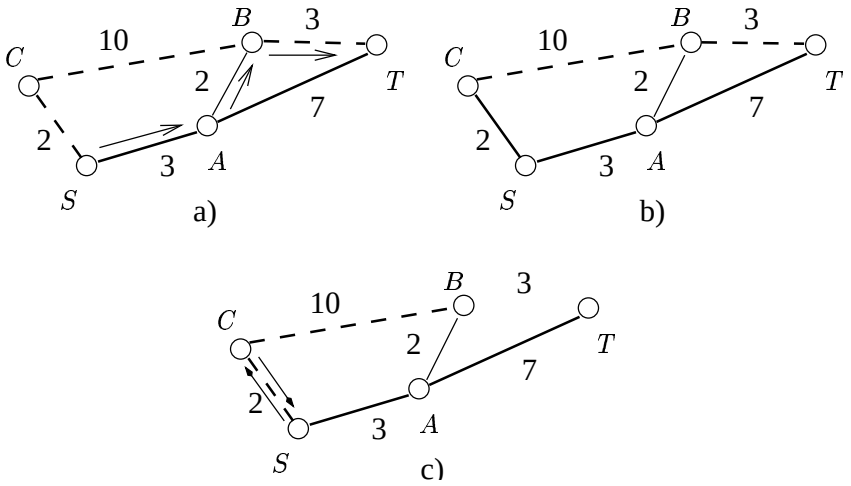

FIG. 7 – *Routage de s à t en utilisant les deux premiers chemins sans coupe*

7 Conclusion, perspectives

Dans ce rapport, nous avons proposé un mécanisme complémentaire à OSPF qui permet de réduire le temps de réaction du réseau suite à une panne. Ce temps de réaction dépend de plusieurs paramètres. Parmi eux, on peut citer en particulier : le temps mis par un routeur OSPF pour décider qu'un routeur voisin est en panne, la durée minimale entre deux calculs successifs de la table de routage, et le temps nécessaire pour calculer la table de routage à partir de la base de données topologique. Notre proposition permet, grâce à un calcul préalable d'un chemin de secours disjoint du premier chemin utilisé pour chaque destination possible dans le réseau, de ramener à zéro les deux derniers paramètres à savoir la durée minimale entre deux calculs successifs de la table de routage et le temps de calcul de la table de routage. Pour cela, nous avons proposé un algorithme qui permet de calculer deux chemins disjoints en un seul passage avec une complexité de $O(n^2)$. Notons que la présence des chemins alternatifs dans la table de routage permet d'augmenter la tolérance aux fautes au niveau des applications sensibles.

Dans ce rapport, nous n'avons pas traité le problème de détection des pannes qui mérite une attention particulière et qui sera traité ultérieurement.

Références

[1] L. Toutain, *"Réseaux locaux et Internet, des protocoles à l'interconnexion"*, Hermès, 1999.

[2] C. Prins, *"Algorithmes de graphes"*, Eyrolles, pp. 141-149, 1997.

[3] J. Moy, *"OSPF Version 2"*,Technical Report RFC 2328, IETF, 1998.

[4] R. W. Callon, *"Use of OSI IS-IS for routing in TCP/IP and dual environments"*,Technical Report RFC 1195, IETF, 1990

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399