Chapitre 7

Algorithmes de routage

7.1. Introduction

Nous avons vu dans le chapitre consacré au routage, le rôle joué par les tables de routage dans la fonction de commutation. Il est essentiel de comprendre que ces tables doivent exister préalablement à la commutation d'un message. Elles ne sont pas modifiées par la fonction de commutation. Il faut maintenant définir comment les remplir. Plusieurs routes sont en général possibles pour atteindre un même destinataire. Une solution manuelle est tout à fait acceptable et même recommandée pour de petits réseaux, ou bien pour des routes qui changent très rarement. Par contre, pour de grands réseaux, il faut un algorithme automatique de choix entre ces routes.

L'objectif de ce paragraphe est de décrire quelques-unes des politiques de routage utilisées pour l'interconnexion des réseaux. L'algorithme de routage doit trouver le « meilleur » chemin pour aller d'un point source A à une destination B. Nous mettons « meilleur » entre guillemets pour plusieurs raisons.

Il existe plusieurs critères pour définir la notion de meilleur chemin (on dit aussi le plus court chemin) :

— le moins cher (argent),

— le plus rapide (en temps),

— le plus grand débit,

— le plus sûr,

— passer par le moins de nœuds possible,

— le plus court (en distance)…

Il n'y a aucune raison pour que ces différents critères aient le même optimum. Nous avons donc à rechercher un optimum multicritère. Or un tel problème est N-P complexe et n'a, en général, pas de solution exacte.

Il doit prendre en compte des critères d'optimisation globaux, c'est-à-dire qui ne concernent pas un flot d'information particulier, mais le trafic global écoulé par le réseau. L'exploitant du réseau souhaite que le terme « meilleur » soit compris au sens suivant :

— écoule le maximum de trafic,

— utilise équitablement les ressources,

— sert le plus de clients possible,

— garantit l'équité entre les clients,

— évite les congestions...

Le choix doit être fait en temps réel, c'est-à-dire que l'algorithme de choix (de routage) ne doit pas être « coûteux » au point par exemple d'augmenter démesurément le délai d'acheminement des messages ou de saturer le réseau. Donc, le meilleur chemin peut être celui que l'on a découvert le plus vite, même si cela n'optimise complètement aucun des critères précédents. La notion de coût ici recouvre à la fois les délais et la charge réseau induite par l'algorithme.

La situation d'un réseau change sans arrêt et il n'est pas possible d'avoir une vue globale exacte à un instant « t », encore moins du futur. Donc une décision prise à l'instant « t » peut être optimale mais ne l'est pas forcément à l'instant « t + $\acute{y}\tau$ ». Doit-on reconsidérer toute décision à chaque changement d'état ?

Le réseau peut être vu comme un graphe G = (V, N), où V est l'ensemble des voies et N l'ensemble des nœuds. Deux fonctions peuvent être spécifiées : D (débit ou capacité), V->$\Re^+$, définit le débit de la voie et C (coût), V->$\Re^+$, définit le coût de la voie. Soit un couplet (Γ_s, Γ_d) de nœuds de N, réaliser un routage entre un nœud de Γ_s et un nœud de Γ_d consiste à trouver un ensemble $\Delta \subseteq V$ tel qu'il existe une suite (V_1,V_2,...V_n) qui conduise du nœud source au nœud destination.

7.1.1. *Difficultés du routage*

Des algorithmes de calcul de route sont nécessaires. Ils peuvent être réalisés de manière centralisée ou distribuée. La politique de routage, mise en œuvre par l'algorithme de routage, doit éviter plusieurs problèmes :

— la congestion, phénomène qui se produit lorsqu'un trafic trop important converge vers un routeur ou une liaison qui ne peut l'écouler. Cette congestion locale risque alors de se propager vers les routeurs adjacents et de conduire ainsi à un blocage du réseau global. On peut ici faire l'analogie avec ce que l'on connaît bien dans les réseaux routiers (les bouchons). Un réseau congestionné a un débit global faible, voire nul ;

— le contournement des éléments en panne, un routeur ou une liaison peuvent cesser de fonctionner. Il faut informer les autres routeurs que cette voie est indisponible et proposer des routes alternatives. Réciproquement, il faut récupérer l'usage de cette voie lorsque la réparation a été effectuée. Ces adaptations aux modifications dynamiques doivent être opérées suffisamment rapidement pour être efficaces et pas trop rapidement pour ne pas créer d'instabilité ;

— ne pas nécessiter que chaque routeur ait à connaître la topologie globale du réseau. L'interconnexion de réseaux (publics ou locaux) entre eux est souvent le fait d'accords entre plusieurs entités : sociétés, universités, administrations… pour construire un système de communication commun. L'adhésion est volontaire et

chacun souhaite rester maître chez lui. Il n'est donc pas, en général, souhaitable ou acceptable de devoir créer une entité supérieure pour administrer le réseau. On cherche pour satisfaire ce besoin à utiliser des algorithmes de routage qui se contentent d'utiliser les informations qu'ils peuvent obtenir de leurs seuls voisins immédiats. Chaque administrateur d'une entité adhérente au réseau n'a besoin de collaborer qu'avec les responsables des réseaux voisins auxquels il est raccordé. Cela n'empêche pas par ailleurs que la structure de connexion du réseau soit réalisée de manière hiérarchisée ;

— de chercher à atteindre des noms qui n'existent pas dans le réseau ou qui sont inaccessibles. Il est nécessaire que chaque routeur ait une connaissance non pas de tous les noms individuels existant dans le réseau, mais des espaces de noms qui peuvent être atteints en utilisant une voie de sortie du routeur, sans préjuger de l'existence de ce nom ni de ce que fera le routeur suivant. Pour chaque voie de sortie possible, le routeur connaît ou apprend dynamiquement l'espace de noms accessibles. Bien évidemment, cet espace de noms change au cours de la vie du réseau.

Enfin, la complexité de l'algorithme doit rester modérée de façon à pouvoir être installé dans les nœuds de commutation.

Du fait de cette variété de critères, il existe de nombreux algorithmes de routage. L'algorithme de routage du réseau d'un opérateur peut rester secret ou propriété de cet opérateur. En principe, les utilisateurs n'ont pas besoin de le connaître. Par contre, pour interconnecter des réseaux privés ou publics, il faut un algorithme de routage public et surtout un protocole public entre les nœuds pour gérer cet algorithme.

7.1.2. *Représentation matricielle du maillage*

Le réseau maillé décrit sur la figure 6.2. peut être représenté par une matrice source-destination (cf. figure 7.1.). Chaque case de la matrice indique l'existence ou la non-existence d'une voie directe entre une source et une destination. Les caractéristiques de la voie : débit, délai de propagation... sont fournies. Ici, les voies sont supposées bidirectionnelles, ce qui fait que la matrice est symétrique. Ce ne serait pas le cas pour les voies unidirectionnelles (sens unique). Trouver un chemin pour aller d'un nœud à un autre nœud consiste à trouver un chemin dans cette matrice.

Le maillage est une structure statique dans la mesure où l'on n'ajoute (ou ne retire définitivement) pas fréquemment de nouveaux nœuds ou de nouvelles voies entre les nœuds. Néanmoins, cela se produit et doit être introduit dans la matrice. Pour un nouveau nœud, il faut ajouter une nouvelle colonne et une nouvelle ligne. Par contre, la disparition temporaire d'une voie (panne, coupure) peut être représentée simplement en mettant la caractéristique débit de cette voie à zéro et le délai à la valeur infini. Il n'est pas nécessaire de modifier la matrice complète pour une panne temporaire.

Des algorithmes itératifs ou récursifs sont capables de trouver l'ensemble des chemins possibles entre tout couple de nœuds. On notera $C_{i,j} = \{c^1_{i,j}, c^2_{i,j}, \ldots c^n_{i,j}\}$ l'ensemble des chemins entre le nœud i et le nœud j, et $c^k_{i,j}$ le k^e chemin entre i et j

Source Dest.	1	2	3	4	5	6	7	8
1	-	V_3		V_2			V_1	
2	V_3	-	V_4			V_6	V_{11}	
3		V_4	-	V_5				V_9
4	V_2		V_5	-	V_8			
5				V_8	-	V_7		
6		V_6			V_7	-		V_{10}
7	V_1	V_{11}					-	
8			V_9			V_{10}		-

Figure 7.1. *Matrice représentant le maillage du réseau*

parmi les n possibles. Notons que cette définition tolère les cycles et le passage par le même nœud plusieurs fois. De ce fait $C_{i,j}$ peut être infini. Pour le rendre fini, il faut interdire les cycles dans les chemins (passage deux fois par une voie déjà parcourue).

Du fait de l'interdiction de repasser par une même voie, les nœuds qui n'ont que deux voies sont pour le routage, soit des points terminaux (ou d'entrée), soit des simples répéteurs. C'est le cas des nœuds 5 et 8 sur notre exemple. Nous les conservons car ils permettent d'atteindre des abonnés locaux et rendent l'exemple plus réaliste tout en lui conservant une certaine simplicité.

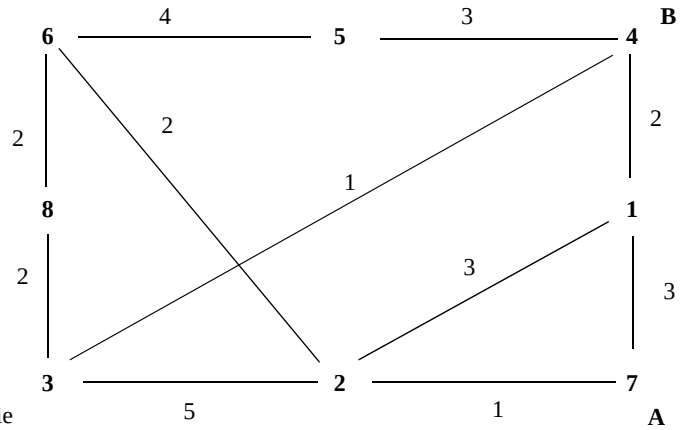


Figure 7.2. *Structure de notre réseau maillé exemple en supprimant la prise en compte de la topologie ; la valeur du coût pour chaque voie est notée*

La figure 7.2. montre le réseau maillé simplifié en le dépouillant de toute référence à une topologie réelle. On notera que cette représentation favorise l'idée que le chemin le plus court est en nombre de nœuds (ce qui est vrai uniquement si l'expression « plus court » considère le nombre de nœuds ou voies traversés ou parcourus). Nous utiliserons désormais plus volontiers cette représentation dépouillée.

7.2. Algorithme centralisé

7.2.1. *Routage dans un centre de gestion de routage*

La solution la plus simple à concevoir consiste à choisir un nœud particulier pour exécuter l'algorithme de routage. Ce nœud, appelé centre de gestion, CG, connaît la topologie du réseau et contient la matrice du maillage. Nous allons imaginer un cas extrême. Le CG seul peut décider du routage. Tous les nœuds savent envoyer des messages au CG. Pour bien comprendre les étapes et le travail, nous avons mis sur la figure 7.3, ce qui est fait pour établir un circuit virtuel entre A et B si le nœud 2 est le centre de gestion[1].

1 / Le demandeur envoie un message au CG, lui indiquant qu'il souhaite envoyer des messages de A vers B.

2 / Le CG exécute son algorithme de recherche de chemin.

3 / Le chemin étant choisi, le CG envoie alors aux nœuds du chemin l'ordre de réserver des ressources (buffer mémoire, numéro de voie logique entrante et sortante …) et de créer une entrée dans la table de routage[2] pour ce circuit virtuel. Le CG met à jour les ressources libres de chaque nœud dans sa matrice. C'est-à-dire qu'il retire du montant de ressources disponibles le nombre de buffers réservés, la fraction de bande passante allouée, etc, de manière à avoir une image à jour du réseau.

4 / Le CG envoie à A l'autorisation d'émettre ses données sur le circuit virtuel établi.

Cette solution est lourde, mais elle garantit que la matrice est à jour. Donc, le CG peut calculer à chaque fois les chemins disponibles. A l'inverse, lorsqu'un circuit virtuel est libéré, le CG doit être informé pour remettre la matrice dans un état cohérent, en restituant les ressources libérées. Si la fréquence des CV à établir est faible ou si les CV sont utilisés longtemps, une telle solution convient bien. La réservation de canaux satellites, pour la télévision par exemple, les liaisons spécialisées du service Transfix, entrent dans cette catégorie d'applications.

---

1. Nous nous appuyons essentiellement sur des tables de routage, technique la plus fréquemment utilisée. Mais cela n'est pas impératif : une liste de nœuds pourrait être aussi fournie au site demandeur et incluse dans chaque message.

2. Pour la table associée à la voie d'entrée, une entrée contenant : LCN -> voie de sortie, LCN (cf. figure 6.12.).

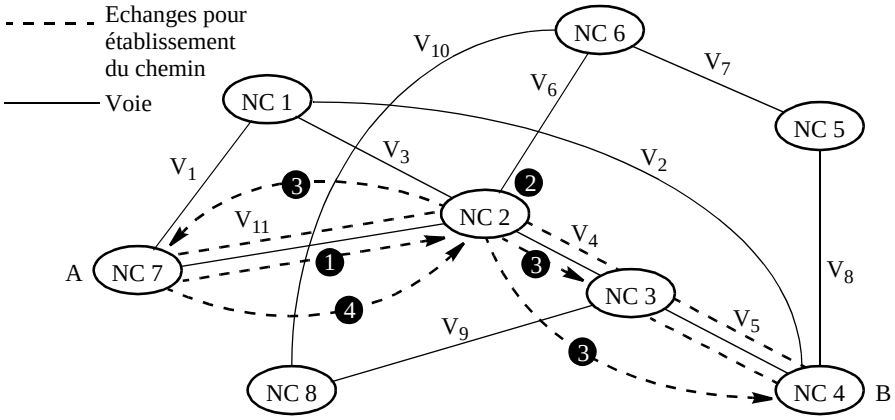


Figure 7.3. *Exemple de routage avec un centre de gestion responsable de toutes les décisions de routage*

Cette solution est conçue pour les CV dont des ressources sont réservées. Pour des datagrammes, cette solution est sans intérêt. On n'imagine pas passer par ces quatre étapes pour envoyer un seul message. Une solution possible serait de supprimer l'étape 3 et d'envoyer le chemin trouvé au demandeur. Ce chemin sera introduit dans l'en-tête réseau des messages et utilisé par les nœuds pour router ce message. Néanmoins, cette solution reste très coûteuse pour l'envoi d'un message unique, à moins de garder en mémoire (cacher en mémoire) chez le demandeur les chemins trouvés.

7.2.2. *Algorithme du plus court chemin*

Voyons maintenant l'algorithme qui peut être mis en œuvre à l'étape 2. Supposons qu'une fonction f(V) : $V \rightarrow \Re^+$ permette d'évaluer pour chaque voie, V, le coût pour passer entre deux nœuds adjacents ou voisins (les nœuds sont adjacents car v est une voie directe). Cette fonction prend en compte les informations connues sur la voie (débit, taux de réservation, taux d'utilisation réel, nombre de buffers libres…). Il faut que l'algorithme trouve le meilleur ou plus court chemin entre source A et destination B. La solution la plus évidente consiste à évaluer :

$C_{ab} = \text{Min} [F(C^1_{ab}), F(C^2_{ab}), \ldots, F(C^n_{ab})]$

où

$$\mathbf{F}(\mathbf{C}^{i}_{a,b}) = \sum_{i=1}^{p} f(\mathbf{V}v_i)$$

où … V_1, … V_p sont les p voies du chemin C^i_{AB}.

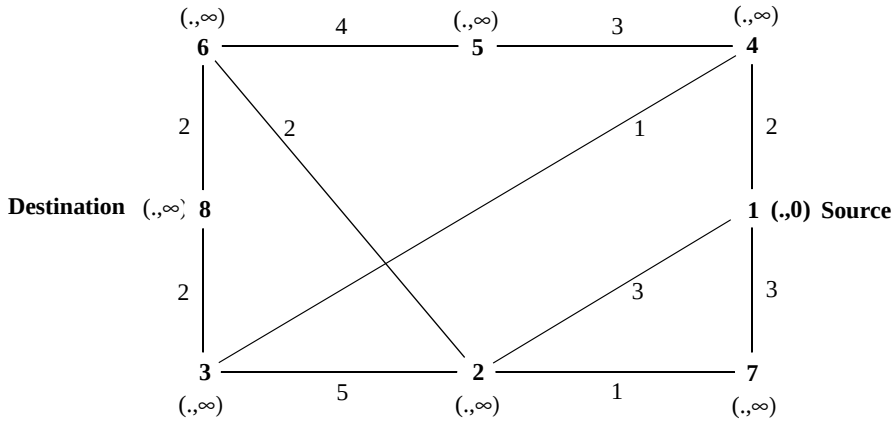


Figure 7.4. *Etat initial*

Cette solution n'est pas acceptable car trop coûteuse. L'algorithme du chemin minimum permet, à partir d'un nœud quelconque — dans notre exemple ce sera le nœud 1 (cf. figure 7.4.) — de trouver le chemin minimum entre le nœud source et chaque nœud destination. Il faudra exécuter l'algorithme pour chaque nœud destination possible et chaque source possible. Cet algorithme est dû à Dijkstra (1959).

L'idée de l'algorithme est simple : on ne peut atteindre une destination qu'en passant par l'une des voies qui y conduit. On va donc partir du point de départ et aller vers tous les voisins en calculant le coût du trajet. Puis, depuis chacun de ces points, on va calculer le coût minimum pour aller vers les voisins immédiats (nœuds dotés d'une voie directe).

Pour cet algorithme, seuls les nœuds connectés à plus de deux voies ont besoin d'être inclus. Ainsi, les nœuds 5, 7 et 8 devraient disparaître car ils ne permettent aucun choix de routage. La voie 6-4 aura alors un coût de 7, la voie 6-3 un coût de 4. La voie 1-2 passant par 7 aura un coût de 4 : elle duplique la voie directe notée sur la figure. Nous gardons ces nœuds pour conserver le maillage initial. Cela permet aussi de traiter le cas d'une voie 8-5 que l'on considère comme temporairement rompue donc de coût infini.

L'algorithme impose que l'on renumérote les nœuds dans un ordre quelconque, sauf pour le nœud origine, qui sera 1, et le nœud destination, qui sera N. Pour éviter au lecteur de reprendre la numérotation des nœuds, nous avons pris comme exemple, sur la figure 7.4, 1 comme nœud origine et 8 comme nœud destination. Il faudra reprendre la numérotation pour tout autre couple source-destination. A chaque nœud on affecte une étiquette (O, numéro du nœud prédécesseur, $C_{1,i}$ coût ou distance depuis l'origine que l'on notera C_i pour simplifier puisque l'origine est toujours le nœud 1) qui est initialisée à (.,0) pour le nœud origine et (.,α) pour tous les autres nœuds. Le

« . » signifie que le prédécesseur n'est pas connu. Le coût infini suggère que les nœuds ne sont pas connectés. Bien sûr, le coût pour rester sur place est nul. A l'état initial, le coût pour aller vers tout autre nœud est inconnu.

Destination Source	1	2	3	4	5	6	7	8
1	-	3		2			3	
2	3	-	5			2	1	
3		5	-	1				2
4	2		1	-	3			
5				3	-	4		
6		2			4	-		2
7	3	1					-	
8			2			2		-

Figure 7.5. *Matrice des coûts ou distance sur les voies reliant deux nœuds adjacents*

La figure 7.5. donne la matrice M des coûts sur les voies, déduits directement de la figure figure 7.4. Appelons m_{ij} l'entrée colonne i ligne j de cette matrice ; m_{ij} n'est autre que f(v) où « v » est la voie entre i et j. L'algorithme itère pour chaque nœud i la recherche des distances. Il est décrit ci-après :

Faire pour i = 1 à N-1

Faire pour j = 1 à N # calcul du coût pour aller vers les voisins en passant par i #

Si m_{ij} existe $\wedge$ [$C_i + m_{ij} < C_j$] alors :

• $C_j = C_i + m_{ij}$

• $O_j = i$

fsi

fin_faire

fin_faire

Faire pour j = 1 à N # regarde s'il existait un coût moindre pour venir en i à partir des voisins #

Si m_{Nj} existe $\wedge$ $[C_j + m_{Nj} < C_N]$ alors :

• $C_N = C_j + m_{ij}$

• $O_N = j$

fsi

fin_faire

La figure 7.6. montre les 8 itérations successives. A la fin de l'algorithme, le chemin le plus court est marqué, ainsi que son coût. Sur la figure, on obtient que le coût minimum pour aller de 1 à 8 est de 5 et qu'il faut passer par les nœuds 3 et 4. Pour trouver ce chemin, on prend la marque en 8 qui indique le prédécesseur, ici 3. A la marque en 3 le prédécesseur est 4. En 4, on trouve 1 comme prédécesseur. Il se peut qu'il y ait plusieurs chemins de même coût (sur l'exemple, c'est le cas du chemin aller de 3 vers 7, le coût est de 6 en passant par 2 ou par 4 et 1). Dans ce cas, l'algorithme donne un seul des deux chemins selon son ordre d'exécution[1].

L'algorithme construit un graphe de routage, ou arbre de routage dont le sommet est l'origine, de plus courts chemins entre la source et al destination. Cet arbre permet de choisir le chemin le plus court et d'éliminer les chemins multiple.

Reprenons la façon dont le centre de gestion va traiter les demandes. Une demande de circuit virtuel est faite entre 1 et 8.

— Le CG exécute l'algorithme du plus court chemin, décrit précédemment.

— Il affecte le chemin trouvé au circuit virtuel, CV, et modifie le coût sur chaque voie en mettant à jour les paramètres des voies et nœuds utilisés.

Ainsi, après ce calcul, il se peut que les nouvelles valeurs des coûts sur chaque voie soient augmentées selon une fonction $f'(C_{CV})$. Supposons que l'augmentation de coût soit de 1, $f'(C_{CV}) = 1$ sur chaque voie composant le CV, sur chaque voie après l'établissement d'un CV.

Ainsi les nouveaux coûts sont :

— $m_{1.4} = 3$

— $m_{4.3} = 2$

— $m_{3.8} = 3$

Nous laissons au lecteur le soin de trouver le prochain chemin de plus faible coût. Vous verrez ainsi que le chemin optimal ait été obtenu à la dernière itération dans l'exemple de la figure 7.6. est dû au hasard.

1. Pour s'en convaincre, le lecteur exécutera l'algorithme en inversant les numéros des nœuds 2 et 3 ainsi que 7 et 8.

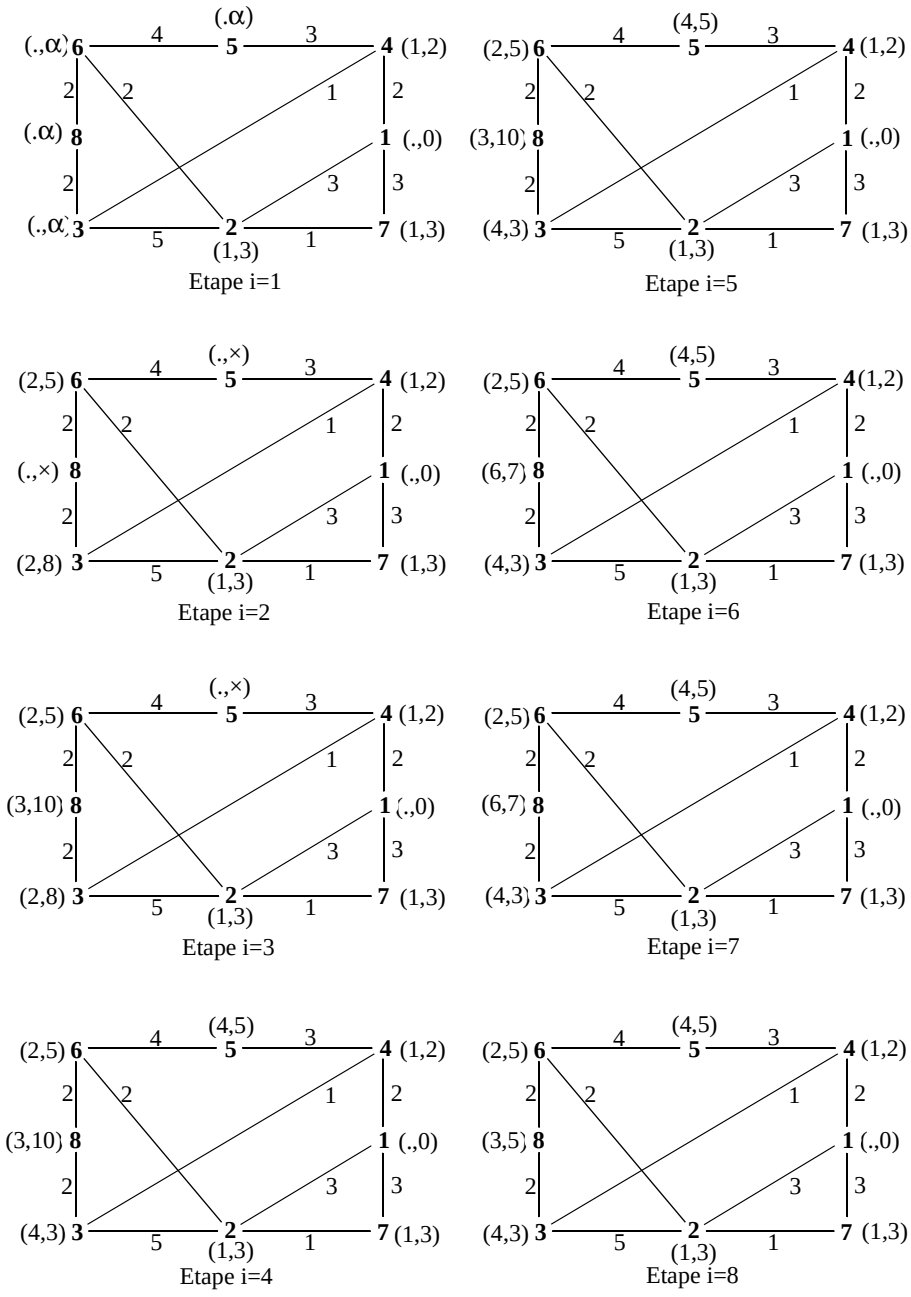


Figure 7.6. *Itérations successives*

7.2.2.1. *Algorithme de Dijkstra*

Une version optimisée de l'algorithme de Dijkstra pour la recherche du meilleur chemin peut être décrite en utilisant deux listes :

— PATH, une liste constituée de triplets (i Nom du nœud, O_i numéro du nœud prédécesseur, C_j coût du chemin). Elle donne le coût associé au chemin optimal pour atteindre chaque destination du réseau à partir du routeur, ainsi que le noeud de sortie (next hop) associé.

— TENT, une liste de même structure que PATH mais organisée par ordre de coût du chemin croissants. Le nom TENT est issu de tentative, ce qui signifie que les chemins indiqués dans TENT sont seulement les meilleurs chemins possibles. Le premier nœud de la liste est utilisé pour rechercher calculer le coût des nœuds voisins. Après quoi il passe dans PATH.

— Soit m_{ij} le coût pour aller de i vers j.

Voici un résumé de l'algorithme de Dijkstra optimisé pour un routeur origine R :

— 1 / Placer R à la racine d'un arbre. Ceci est fait en mettant (i = numéro associé à R, 0, 0) dans la base de données PATH.

— 2 / Faire pour j = 1 à N # calcul du coût pour aller vers les voisins en passant par i #

Si m_{ij} existe $\wedge$ $[C_i + m_{ij} < C_j]$ alors :

• $C_j = C_i + m_{ij}$

• $O_j = i$

Placer le triplet (j, O_j , C_j) à sa place, par ordre croissant de coût puis par ordre de numéro de station i, dans la liste ordonnée TENT

fsi

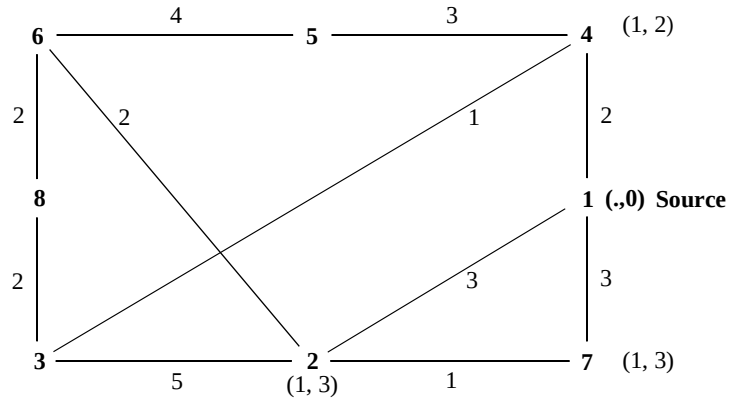
fin_faire

— 3 / Si TENT est vide, l'algorithme est terminé. Autrement, le premier triplet (i Nom du nœud, O_i numéro du nœud prédécesseur, C_j coût du chemin) dans TENT à le coût minimum. Déplacer ce triplet dans PATH et aller à l'étape 2 avec la valeur i de ce nœud.

En exécutant l'algorithme pour R=1 on obtient les étapes suivantes.

L'algorithme affecte aux nœuds voisins de 1 le coût de traversée des voies qui y conduisent puisque au départ les coûts sont infinis (cf. figure 7.7.). Il sont mis dans la liste TENT par ordre de coût et de noms. Le nœud 4 est le premier nœud de la liste TENT. C'est donc le meilleurs chemin pour aller en 4. Il va être traité par la suite

Sur la figure 7.8. l'algorithme affecte aux nœuds voisins de 4 le coût de traversée des voies qui y conduisent plus le coût pour arriver en 4 depuis 1 s'il est plus petit que le coût précédemment connu. C'est le cas pour les nœuds 2 et 4 puisque leurs coûts sont infinis, 4 n'est pas concerné car d'une part son coût est inférieur et de toute façon il est déjà dans PATH, donc son coût minimal est trouvé. Les nœuds 2 et 5 sont mis



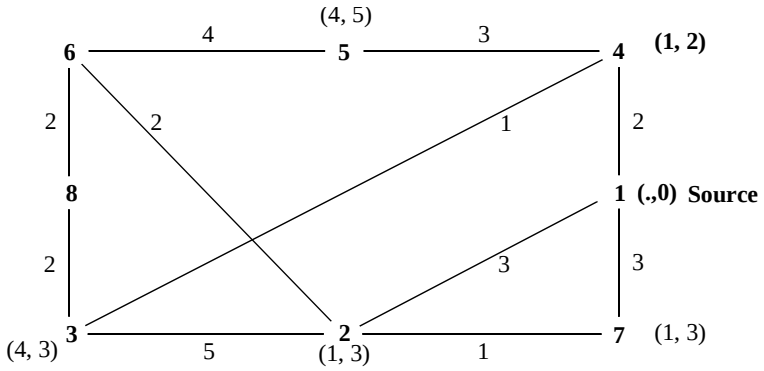
PATH = (1,0,0)

TENT= (4, 1, 2), (2, 1, 3), (7, 1, 3)

Figure 7.7. *Etape 1*

dans la liste TENT par ordre de coût et de noms. Le nœud 2 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 2 est donc trouvé. Il va être traité par la suite.

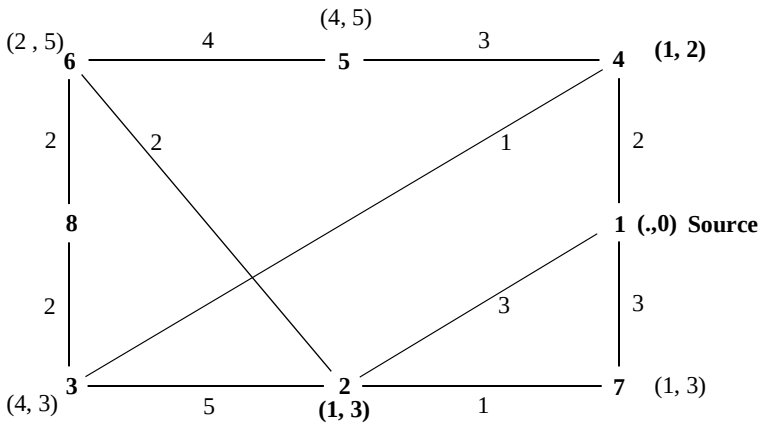
A l'étape 3 sur la figure 7.9. l'algorithme affecte aux nœuds voisins de 2 le coût de traversée des voies qui y conduisent plus le coût pour arriver au nœud 2 depuis 1 s'il est plus petit que le coût précédemment connu. C'est le cas pour les nœuds 6 et 3 puisque leurs coûts sont infinis, 1 n'est pas concerné car d'une part son coût est



PATH = (1,0,0), (4,2,1)

TENT= (2, 1, 3), (3, 4, 3), (7,1, 3), (5,4, 5),

Figure 7.8. *Etape 2*



PATH = (1,0,0), (4,2,1), (2, 1, 3)

TENT= (3, 4, 3), (7, 1, 3), (5,4, 5), (6, 2, 5),

Figure 7.9. *Etape 3*

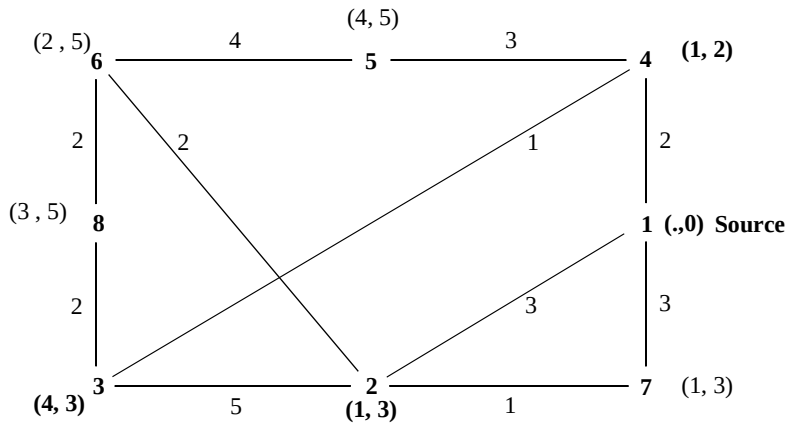
inférieur et de toute façon il est déjà dans PATH, donc son coût minimal est trouvé. Les nœuds 3et 6 sont mis dans la liste TENT par ordre de coût et de noms. Le nœud 3 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 3 est donc trouvé. Il va être mis dans PATH et traité par la suite.

A l'étape 4 sur la figure 7.10. les nœuds voisins de 3 sont 2, 4 et 8. 2 et 4 sont déjà dans PATH il ne sont pas concernés. La distance pour aller de 1 vers 8 en passant par 3 est 5. Dans ce cas l'ordre ne change pas dans la liste TENT. Le nœud 7 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 7 est donc trouvé. Il va être mis dans PATH et traité par la suite

A l'étape 5 sur la figure 7.11. les nœuds voisins de 7 sont 1 et 2 ils sont déjà dans PATH donc ils ne sont pas concernés. Le nœud 5 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 5 est donc trouvé. Il va être mis dans PATH et traité par la suite.

A l'étape 6 sur la figure 7.12. les nœuds voisins de 5 sont 4 et 6. 4 est déjà dans PATH il n'est pas concerné. La distance pour aller de 1 vers 6 en passant par 5 est de 5 + 3 (coût de la voie 1 vers 5). Ce coût est supérieur au coût de 5 connu actuellement dans TENT. Donc on ne change rien pour 6 dans la liste TENT. Le nœud 6 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 6 est donc trouvé. Il va être mis dans PATH et traité par la suite.

A l'étape 7 sur la figure 7.13. les nœuds voisins de 6 sont 5 et 8. 5 est déjà dans PATH il n'est pas concerné. La distance pour aller de 1 vers 8 en passant par 6 est de 5 + 2 = 7. Ce coût est supérieur au coût de 5 connu actuellement dans TENT. Donc on ne change rien pour 8 dans la liste TENT. Le nœud 8 est le premier nœud de la liste



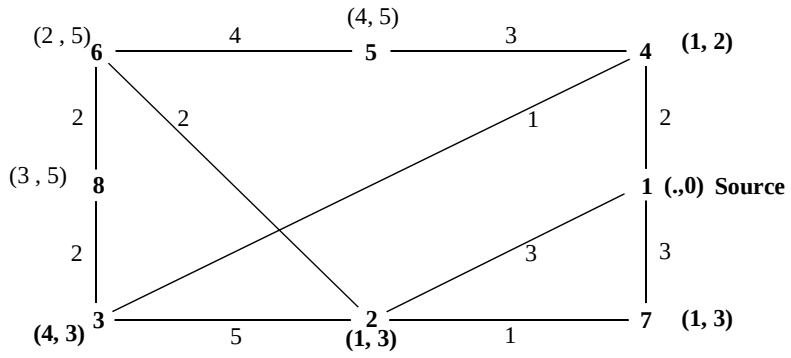
PATH = (1,0,0), (4,2,1), (2, 1, 3), (3, 4, 3)

TENT= (7, 1, 3), (5, 4, 5), (6, 2, 5), (8, 3 , 5)

Figure 7.10. *Etape 4*

TENT. Le nœud 8 est le premier nœud de la liste TENT. Le meilleurs chemin pour aller en 8 est donc trouvé. Il va être mis dans PATH et traité par la suite.

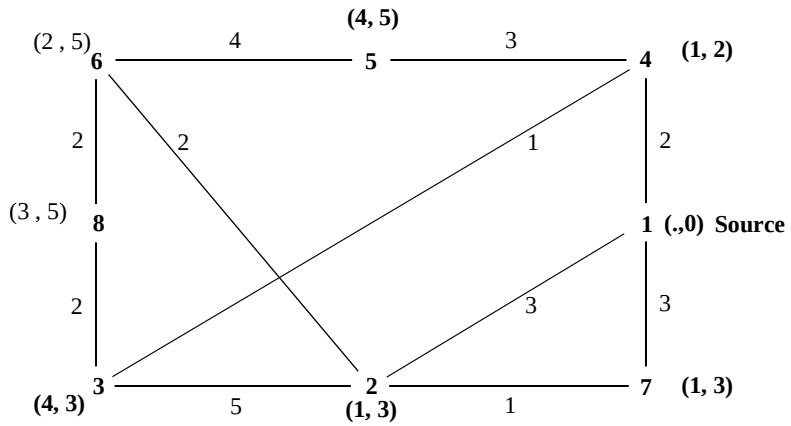
A l'étape 8 sur la figure 7.14. le processus terminé car la liste TENT est vide. La liste PATH contient une entrée pour chaque nœud du réseau.



PATH = (1,0,0), (4,2,1), (2, 1, 3), (3, 4, 3), (7, 1, 3)

TENT= (5, 4, 5), (6, 2, 5), (8, 3 , 5)

Figure 7.11. *Etape 5*



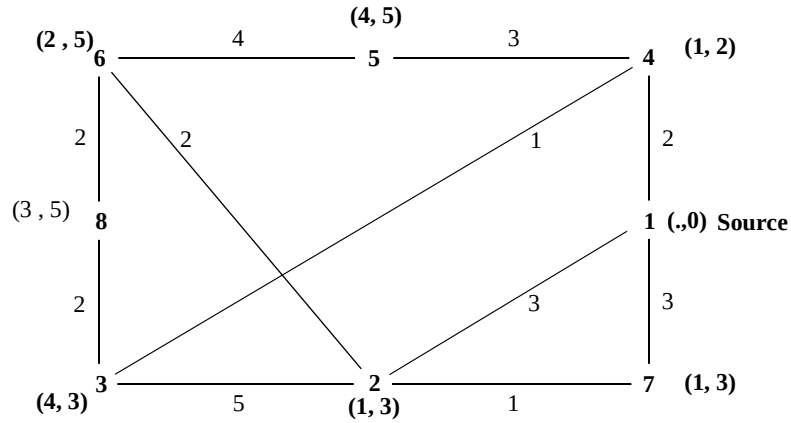
PATH = (1,0,0), (4,2,1), (2, 1, 3), (3, 4, 3), (7, 1, 3), (5, 4, 5)

TENT= (6, 2, 5), (8, 3 , 5)

Figure 7.12. *Etape 6*

7.2.2.2. *Conclusion sur le routage centralisé*

La solution du centre de gestion a de nombreux inconvénients. Si le nœud qui le supporte tombe en panne, aucune nouvelle communication n'est possible. Un trafic



PATH = (1,0,0), (4,2,1), (2, 1, 3), (3, 4, 3), (7, 1, 3), (5, 4, 5), (6, 2, 5),

TENT= (8, 3 , 5)

Figure 7.13. *Etape 7*

important va converger vers le CG si la fréquence des demandes de connexion est élevée. Il faut en outre une forte puissance de traitement dans ce nœud.

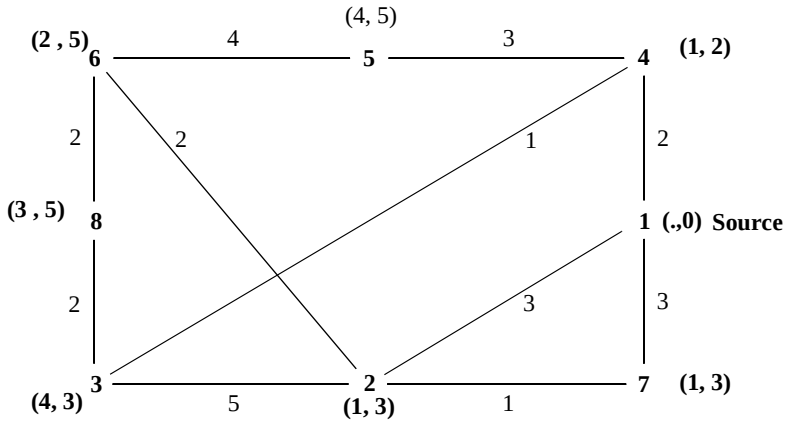
Elle a quelques avantages, outre le fait d'expliquer comment l'on cherche un chemin minimum. Le CG possède toutes les informations. Il est capable de contrôler le trafic et donc d'empêcher les congestions. Il permet de disposer d'une image globale et donc de la fournir aux administrateurs. Il adapte son routage de manière la plus fine possible aux changements d'état du réseau.

Par contre, cette solution ne peut faire qu'une allocation statique de ressources et ne sait pas utiliser les capacités de transmission demandées par les utilisateurs, mais non utilisées en pratique. Enfin, en cas de panne d'un nœud, il faut que le CG s'en aperçoive et modifie sa matrice d'état. Il doit alors scruter tous les nœuds périodiquement pour vérifier qu'ils sont toujours actifs.

Cette remarque sert de transition pour présenter une deuxième solution à routage centralisé, basée sur la distribution des tables de routage.

7.2.3. *Routage centralisé et distribution des vecteurs de routage*

La vulnérabilité du réseau est directement fonction de celle du centre de gestion. Les coûts pour interroger le CG sont inacceptables pour des communications de courte durée. La solution est inadaptée au routage des datagrammes. En outre, l'ouverture d'une nouvelle communication modifie en général trop peu l'état du réseau pour justifier le calcul d'un nouveau chemin.



PATH = (1,0,0), (4,2,1), (2, 1, 3), (3, 4, 3), (7, 1, 3), (5, 4, 5), (6, 2, 5), (8, 3 , 5)

TENT=

Figure 7.14. *Etape 8*

Destination Source	1	2	3	4	5	6	7	8
1	-	2	4	4	4	2	7	4
2	1	-	3	1	6	6	7	6
3	4	2	-	4	4	8	2/4	8
4	1	1	3	-	5	3	1	3
5	4	6	4	4	-	6	4	4/6
6	2	2	8	8	5	-	2	8
7	1	2	2	1	2	2	-	2
8	3	6	3	3	6/3	6	6	-

Nous avons noté i/j lorsque deux chemins ont le même coût. Le vecteur sera composé de la première valeur seulement (une au hasard est trouvée par l'algorithme).

Figure 7.15. *Matrice des plus courts chemins ou table de routage obtenue par le centre de gestion du routage pour notre réseau exemple et la matrice des coûts de la figure 7.5*

L'idée consiste donc à faire le calcul de tous les plus courts chemins dans le CG pour un état donné du réseau. Le résultat sera une matrice des plus courts chemins, aussi appelée table de routage. Chaque ligne de la matrice correspond à un nœud, et sur chaque colonne on trouve le successeur (c'est-à-dire le nœud adjacent) où l'on enverra le message pour atteindre la destination. La figure 7.15. montre un exemple d'une telle matrice pour notre exemple de cours.

On appelle vecteur de routage, V^i, du nœud i la ligne de cette matrice correspondant au nœud ; ce vecteur est distribué à chaque nœud du réseau afin de constituer sa table de routage (cf. figure 7.16.). Maintenant, le routage peut être fait en chaque nœud. A chaque fois qu'un nœud reçoit un message, l'adresse lui indique le nœud final destinataire, « j ». L'entrée $V^i(j)$ donne le nœud où doit être réexpédié ce message. Cette solution est parfaitement adaptée aux datagrammes. Chaque datagramme (lettre, message, paquet...) contient une adresse destination ; chaque nœud dispose d'un vecteur lui indiquant où aller immédiatement après pour atteindre cette destination.

Réaliser un circuit virtuel n'est guère plus complexe. Un paquet d'appel utilise le principe de routage du datagramme pour trouver sa route. En chaque nœud traversé, il réserve des ressources et associe à la voie d'entrée une voie de sortie. Une fois ce travail fait, tous les autres messages suivront le chemin établi par le paquet d'appel et n'auront plus besoin de porter l'adresse de destination.

Cette solution fonctionne bien et de grands réseaux l'utilisent. Elle pose toutefois plusieurs problèmes. Les routes ne sont pas optimales *ad vitam aeternam*. Il y a lieu de les recalculer périodiquement pour s'adapter à l'évolution du trafic. Pour cela, les

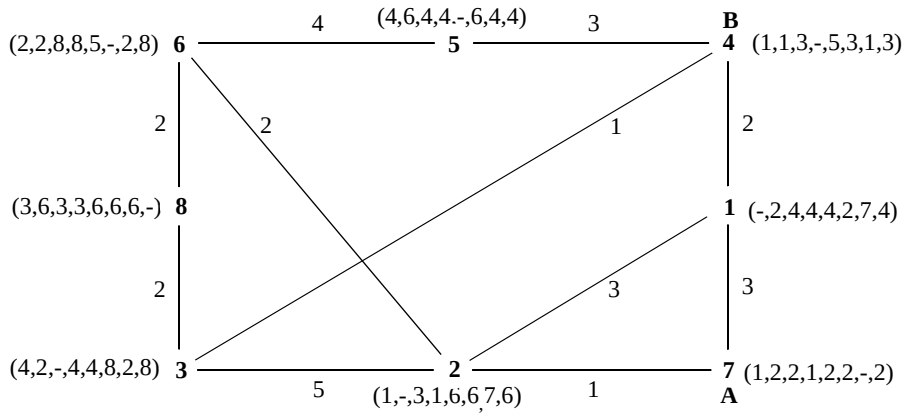


Figure 7.16. *Vecteurs de routage, issus de la matrice de la figure 7.15. en chaque nœud du réseau*

nœuds envoient périodiquement leurs états (nombre de connexions ouvertes, taux d'occupation des voies, taux d'utilisation du CPU, taille des files d'attente devant les voies) au CG. La fréquence d'envoi de ces messages dépend de l'estimation faite de la probabilité de changement de l'état global du réseau. Si la fréquence d'envoi est élevée la charge du réseau et du CG due à ces messages peut devenir intolérable. Imaginez un réseau avec 1 000 nœuds : le CG reçoit alors plusieurs fois par seconde 1 000 messages d'état des nœuds.

Le CG calcule une nouvelle matrice de routage. Si celle-ci est significativement différente de la précédente, il diffuse les nouveaux vecteurs aux nœuds du réseau. Par « significativement différente », il faut comprendre que l'on souhaite éviter de modifier le routage pour un nouvel état global du réseau qui n'est pas fiable. Ce qui aurait pour conséquence de revenir à l'état antérieur au cycle suivant. On cherche à établir une stabilité par un mécanisme d'hystérésis. Il faut que le coût ait changé fortement pour diffuser des vecteurs modifiés (nouveaux vecteurs).

La figure 7.16. montre le réseau avec les vecteurs dans les nœuds. Le vecteur ne peut contenir que les nœuds adjacents, puisque seuls ceux-ci sont reliés par une voie directe. Chaque nœud calcule en continu les coûts pour aller au nœud adjacent. Il les envoie périodiquement au CG.

Sur la figure 7.17, le coût des voies change. La nouvelle matrice des coûts est donnée sur la figure 7.18. Les nouveaux vecteurs sont alors diffusés. Ils n'arrivent pas au même instant dans tous les nœuds du réseau. Pendant la phase de diffusion, les vecteurs de routage ne sont pas cohérents. Ils correspondent à des matrices différentes, ce qui génère un trafic important vers le nœud CG.

La réception des vecteurs de routage n'est pas simultanée. Ainsi, pendant la phase de diffusion, des vecteurs peuvent être incohérents (correspondre à deux matrices

différentes) et introduire des boucles dans le routage. C'est ce qui se produit sur la figure 7.19. qui décrit une étape intermédiaire dans la diffusion de la matrice. Le nœud 5 route les message à destination de 8 par le nœud 4, avec l'ancien vecteur, alors que le nœud 4 route en passant par 5, avec le nouveau vecteur. Une telle situation se corrigera lorsque le nouveau vecteur arrivera au nœud 5.

Destination Source	1	2	3	4	5	6	7	8
1	-	2	**2**	4	4	2	7	**2**
2	1	-	3	1	6	6	7	6
3	**2**	2	-	4	**8**	8	**2**	8
4	1	1	3	-	5	**5**	1	**5**
5	4	6	4	4	-	6	4	**6**
6	2	2	8	**5**	5	-	2	8
7	1	2	2	1	2	2	-	2
8	6	6	3	**6**	**6**	6	6	-

Figure 7.18. *Matrice des plus courts chemins obtenue par le centre de gestion du routage. Pour l'état du réseau de la figure 6.2. Les modifications sont en gras*

De telles incohérences sont inévitables. Il est impossible en réseau de faire basculer au même instant tous les nœuds du vecteur « i » au vecteur « i + 1 ». La notion d'heure globale cohérente est illusoire parce que les messages mettent un temps variable pour se propager dans le réseau. Il est aussi très complexe de vouloir garantir que tous les nœuds ont bien reçu le nouveau vecteur. En effet, des messages peuvent se perdre, quelles que soient les précautions prises. On appelle « routage cohérent » le fait que tous les nœuds du réseau aient un vecteur de routage correspondant à une

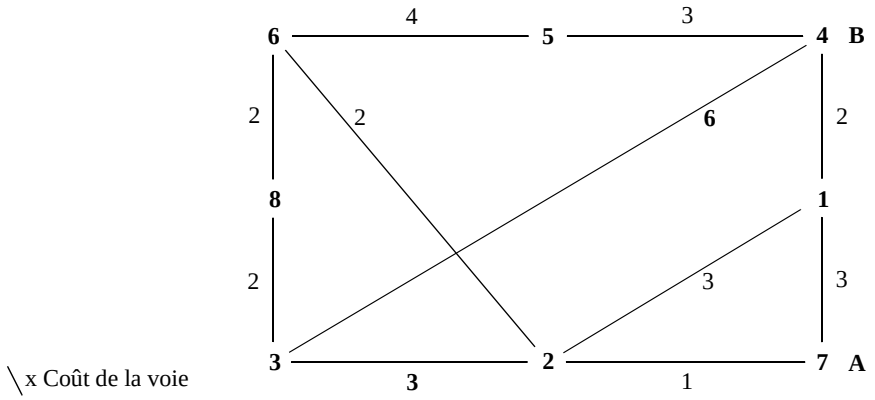


Figure 7.17. *Le coût de $m_{3,4}$ passe de 1 à 6 et de $m_{3,2}$ passe de 5 à 3*

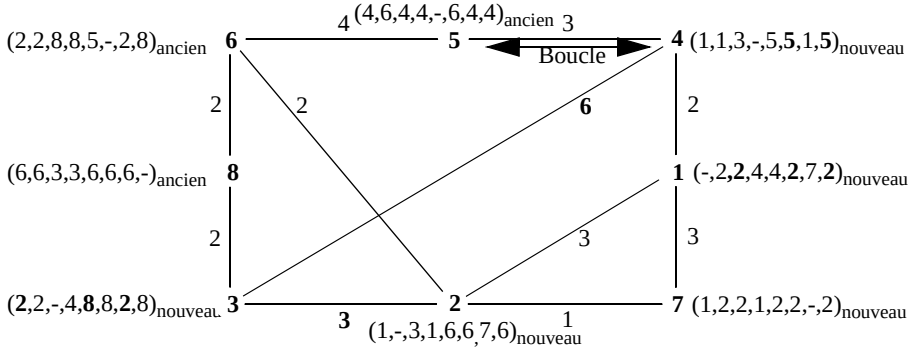
même matrice calculée par le CG. Quelles que soient les précautions prises, on aura toujours pendant la phase de diffusion des états de routage incohérents.

Les conséquences de ces états incohérents sont que certains messages vont subir des délais d'acheminement anormalement importants par rapport à la moyenne, du fait de boucles dans le routage. Nous laissons à votre réflexion les conséquences pour un circuit virtuel, CV, le fait que son paquet d'appel subisse une telle situation. Tous les messages ultérieurs suivront les même méandres que le paquet d'appel.

Cet algorithme reste centralisé : si le centre de gestion tombe en panne, le réseau ne sait plus s'adapter aux modifications de trafic. Néanmoins, contrairement à la solution précédente, les nœuds sauront poursuivre le routage. Cette solution peut donc tolérer une absence temporaire du CG, pourvu qu'il n'y ait pas de modification significative du trafic pendant cette période. Par contre, cette solution repose sur l'envoi au CG d'informations sur l'état des nœuds. La période d'envoi est très faible, la mise à jour du routage sera fine, c'est-à-dire proche des changements d'état du réseau. Par contre, la charge réseau provoquée par ces messages est d'autant plus élevée que la fréquence d'envoi l'est. Si la fréquence est trop faible, on risque de laisser se développer des congestions avant d'avoir à calculer une nouvelle matrice de routage. Une fois le réseau congestionné, il sera difficile de transmettre même ces messages d'état, et donc de revenir à un état de routage satisfaisant. C'est le cas pour une ambulance, les pompiers ou la police tentant de remonter un bouchon pour en résoudre la cause.

7.3. Algorithmes distribués

Les problèmes évoqués à propos des algorithmes centralisés montrent qu'une solution répartie devrait apporter des avantages majeurs. Une solution répartie n'aura plus de CG et donc sera plus robuste. En évitant le centre de gestion, elle permet à des



L'état global ne correspond pas encore à la matrice de routage décrit sur la figure 7.18. Les risques de boucles existent pour le nœud 4 qui envoie vers 5 pour aller en 8, et le nœud 5 qui continue à envoyer vers 4.

Figure 7.19. *Etat des vecteurs de routage pendant une phase de changement.*

organismes indépendants de construire un réseau fédératif sans avoir à créer une structure chargée de gérer le réseau. Nous allons décrire ici deux principes utilisés pour ces algorithmes distribués :

— la patate chaude ou routage par saut qui sont traités dans le paragraphe 7.3.1. routage adaptatif,

— l'inondation ou routage par la source qui sont traités dans le paragraphe 7.3.2. sur l'inondation .

La patate chaude est basée sur cette idée simple : si chaque nœud se débarrasse « au plus vite » des messages qu'il reçoit et qui ne lui sont pas destinés, alors le message a de bonnes chances d'atteindre vite sa destination. Chaque routeur va procéder un peu comme lorsque, étant en voiture sur un boulevard avec plusieurs files, on arrive à un feu rouge. Parmi les files organisées qui sont devant nous, on choisit, parmi celles qui nous dirigent vers notre destination, celle qui a le moins de voitures, en espérant ainsi passer plus vite. Bien sûr, cela ne nous garantit pas contre un prédécesseur endormi, très lent ou une file bloquée en aval. On prend une décision sur une information locale immédiate qui ne saurait tenir compte de l'état ultérieur du chemin.

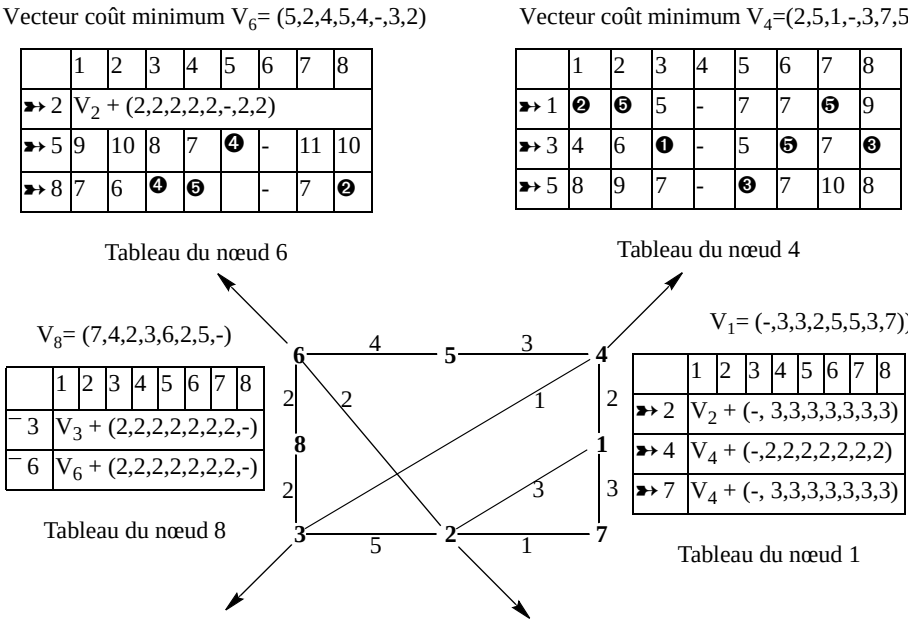
L'inondation consiste à tester effectivement tous les chemins : c'est une forme extrême du principe précédent. Lorsqu'un choix se présente, le message est répliqué et envoyé dans toutes les directions. On est sûr ainsi que le chemin le plus rapide sera trouvé par le premier message qui arrive à destination. Imaginez que vous disposez d'autant d'estafettes que vous le désirez. Vous les envoyez expérimenter chaque chemin possible. Chaque estafette collecte des informations sur l'état du chemin qu'elle a suivi. A l'arrivée, il suffira de choisir en fonction des critères retenus et des informations obtenues le meilleur chemin parmi les chemins trouvés. Ainsi l'estafette qui arrive la première a sûrement trouvé le chemin le plus rapide[1]. On l'utilisera pour les messages ultérieurs.

Dans les algorithmes de routage distribués, le calcul des routes est donc partagé entre les noeuds du réseau grâce à l'information de routage qu'ils échangent entre eux. Cette solution est beaucoup plus robuste car elle évite de faire passer systématiquement les communications par un noeud spécifique qui peut paralyser le réseau en cas de panne.

7.3.1. *Routage adaptatif*

Cette technique est une amélioration du principe de la patate chaude. Il consiste à combiner cette idée avec des vecteurs de routage statiques. En chaque nœud, un vecteur de routage indique par où l'on peut, ou l'on accepte, de passer pour atteindre une destination quelconque. Sur notre exemple de réseau maillé, on peut atteindre

1. Sous réserve que ses successeurs aient la même conduite. Imaginez le cas d'une estafette en moto et d'un 35-tonnes qui suit la route indiquée. L'idée ramenée à un réseau correspond à un petit message estafette et de gros messages de données. Cela soulève le choix des paramètres à collecter.



	1	2	3	4	5	6	7	8
➳ 2	V_2 + (2,2,2,2,2,-,2,2)							
➳ 5	9	10	8	7	❹	-	11	10
➳ 8	7	6	❹	❺		-	7	❷

	1	2	3	4	5	6	7	8
➳ 1	❷	❺	5	-	7	7	❺	9
➳ 3	4	6	❶	-	5	❺	7	❸
➳ 5	8	9	7	-	❸	7	10	8

	1	2	3	4	5	6	7	8
¯ 3	V_3 + (2,2,2,2,2,2,2,-)							
¯ 6	V_6 + (2,2,2,2,2,2,2,-)							

	1	2	3	4	5	6	7	8
➳ 2	V_2 + (-, 3,3,3,3,3,3,3)							
➳ 4	V_4 + (-,2,2,2,2,2,2,2)							
➳ 7	V_4 + (-, 3,3,3,3,3,3,3)							

	1	2	3	4	5	6	7	8
¯ 2	8	❺	-	10	11	7	6	9
¯ 4	❸	6	-	þ	❹	6	❻	4
¯ 8	7	6	-	5	8	❹	7	❷

	1	2	3	4	5	6	7	8
➳ 7	4	-	7	6	8	4	❶	6
➳ 1	❸	-	6	❺	8	8	6	8
➳ 6	7	-	6	9	❻	❷	5	❹
➳ 3	8	-	❺	6	9	9	11	7

Les valeurs minimales qui composent le vecteur de choix du routage sont mises sur fond noir. Chaque nœud a une table donnant le vecteur coût pour aller en i en sortant par une de ses voies locales (noté ¯ 7 pour voie locale). A tout instant il émet sur la voie locale la moins chère pour aller en i.

Figure 7.20. *Etat des matrices de routage correspondant à l'état initial de notre exemple*

toutes les destinations à partir de tous les nœuds. Il y aura donc des routes que l'on souhaite privilégier. Pour cela, on attribue à chaque route possible un coût qui va être modifié dynamiquement. La représentation en chaque nœud prend la forme d'une matrice ayant une colonne par destination et une ligne par voie de sortie. Par exemple, le nœud 2 aura 4 lignes correspondant aux voies V_3, V_4, V_6, et V_{11}, alors que le nœud 8 n'a que deux lignes correspondant aux voies V_9 et V_{10}.

Sur la figure 7.20, nous avons représenté les matrices en chaque nœud correspondant au coût par voie de notre état initial, figure 7.4. La ligne indique le coût pour aller vers le site désiré en sortant par la voie locale correspondante. Le contenu des lignes n'est plus toujours le coût minimal comme précédemment. Pour calculer ce coût, il faut ajouter le coût d'un passage sur cette voie (forcer le passage par cette voie) au coût minimum du nœud atteint pour aller à la case correspondante. Ainsi, sur la figure 7.20, sur le nœud 2, le coût pour aller de 2 en 8 en passant par 7 est de [1 + coût minima de 7 en 8]. Le coût minima de 7 en 8 passe par les nœuds 2 et 6. Ce routage conduirait à une boucle s'il était réellement utilisé.

La fonction de routage consiste, pour une destination donnée, à prendre dans la colonne du nœud destinataire la voie de sortie la moins chère. Par exemple, à partir du nœud 2 sur la figure 7.20, la voie de sortie pour aller en 5 sera d'aller vers 6. Le vecteur des coûts est (8, 8, 6, 9) ; 8 en passant par le nœud adjacent 7, 8 en passant par le nœud adjacent 1, 6 en passant par le nœud adjacent 6, et enfin 9 en passant par le nœud adjacent 3. Sur la figure 7.20, le vecteur des coûts minimaux apparaît en cercles noirs. Il indique la voie de sortie pour atteindre une destination. Ainsi en 2 le vecteur des coûts est (3,-,5,5,6,2,1,4) et le vecteur de routage que l'on en déduit est (1,-,3,1,6,6,7,6) où chaque entrée désigne la voie, par le numéro du nœud adjacent, à utiliser.

La prise en compte des coûts, autrement dit l'évaluation de la fonction de coût, peut être faite de différentes manières. Par exemple, en introduisant dans certains messages des informations permettant de calculer le coût sur chaque voie ou chaque nœud, ou bien en mesurant le temps écoulé entre l'envoi d'une donnée et son acquittement (rtt, *round trip transmission*). Elle peut être faite en forçant le routage des messages. Nous allons décrire ici en algorithme de mise à jour de ces tables, dont le principe est utilisé dans de nombreux réseaux. Chaque nœud mesure les coûts sur ses voies de sortie propres (les liaisons avec les nœuds adjacents).

Chaque nœud doit être capable de détecter un changement dans le coût pour atteindre un nœud adjacent. Le coût pour atteindre un nœud adjacent en panne sera noté comme infini. On notera « + x » une augmentation de coût de « x », « -x » une réduction du coût, ∞ une voie ou un nœud adjacent en panne, x le coût d'une voie ou d'un nœud état de marche avec un coût x.

— a/ Si le coût change sur une voie allant vers un nœud adjacent, tous les coûts sur cette ligne de la matrice sont modifiés. S'il s'agit d'une augmentation, on accroît toute la ligne de ce coût (+x) ; s'il s'agit d'une diminution, on diminue toute la ligne (-x). S'il s'agit d'une panne, toute la ligne passe à ∞, et, réciproquement, s'il s'agit d'un retour à l'état actif, toute la ligne est mise à x (ce qui n'est pas la meilleure stratégie mais simplifie les explications).

— b/ Le nœud après un changement de coût va regarder si son vecteur des coûts minimaux est modifié et pour quelles voies de sortie. S'il n'est pas modifié, le nœud ne fait rien. S'il est modifié, le nœud envoie à tous ses voisins un message indiquant : « modification de coût (+x, -x, ∞ ou x) pour aller en "j" en passant par "moi" (nœud i) », pour chaque modification.

— c/ Chaque voisin (nœud adjacent), quand il reçoit ce message, prend en compte la/les modification(s) annoncée(s). Il va donc pour l'entrée ligne i (nœud origine), colonne j (destination finale) de sa matrice, effectuer l'ajout (si +x est reçu), la réduction (si -x est reçu), mise à x ou ∞ dans les deux autres cas. Une fois cette opération effectuée le nœud exécute l'opération « b » précédemment décrite.

L'algorithme converge en un nombre fini d'itérations si les modifications du réseau ne sont pas trop rapides. Le résultat n'est pas l'obtention des routes minimales globales, mais un état approché. Pour obtenir les routes minimales globalement il faudrait indiquer tous les changements de coût. Il faut vérifier dans ce cas que les vecteurs minimaux sont bien équivalents dans cette solution et dans la précédente. On n'utilise pas cette approche car elle risquerait de conduire à un échange important de messages à chaque changement de coût local, même s'il ne change pas la décision de routage. En conséquence, l'algorithme divergerait et le réseau serait saturé.

Nous allons illustrer cet algorithme sur les modifications de coût décrites sur la figure 7.17. On supposera que le coût augmente d'abord sur la voie 3 vers 4, puis plus tard diminue sur la voie 3 vers 2.

Les nœuds 3, 2 et 4 observent localement, indépendamment l'un de l'autre, une modification de coût sur leur voie de sortie. Ainsi, le nœud 3 va observer une augmentation de coût de 5 unités sur sa voie de sortie vers 4. Il ajoute ce coût sur la matrice de coût sortant vers 4. Ce qui lui donne la nouvelle matrice décrite sur la figure 7.21.

	1	2	3	4	5	6	7	8
➳ 2	8	❺	-	10	11	7	❻	9
➳ 4	8	11	-	6	9	11	11	9
➳ 8	❼	6	-	❺	❽	Õ	7	❷

Figure 7.21. *Etat de la matrice du nœud 3 après prise en compte de l'augmentation de coût de la voie 3 ->4*

L'ancien vecteur coût minimum V_3=(3,5,-,1,4,4,6,2) devient V_3=(7,5,-,5,8,4,6,2). Les coûts minimaux pour aller en 1, 4 et 5 sont augmentés chacun de 4. Il va envoyer un message à ses voisins en les informant que le coût pour aller en 1, 4 et 5 en passant par 3 est augmenté de 4. Il le fait car il a modifié pour ces destinations son choix de routage. Pour aller en 7, le routage n'est plus le même : on passera désormais par la voie 3->2 bien que le coût reste le même. Aucune information n'est fournie aux voisins.

Les voisins de 3 sont les nœuds 2, 4 et 8. Ils vont ajouter 4 à l'entrée correspondante de leur matrice, ce qui est décrit sur la figure 7.22.

Dans ce cas, le vecteur de coût minimal n'est pas modifié. Le nœud 2 ne fera rien. Les décisions de routage de 2 ne sont pas modifiées.

	1	2	3	4	5	6	7	8
➳ 7	4	-	7	❻	8	4	❶	6
➳ 1	❸	-	6	5	8	8	6	8
➳ 6	7	-	6	9	❻	❷	5	❹
➳ 3	12 (8+4)	-	❺	10 (6+4)	13 (9+4)	9	11	7

Figure 7.22. *Etat de la matrice du nœud 2 après réception des messages issus de 3*

Par contre, le nœud 4 voit lui aussi l'augmentation de coût de sa voie vers 3 et il opère la même action que le nœud 3, comme le montre la figure 7.23.

	1	2	3	4	5	6	7	8
➳ 1	❷	❺	❺	-	7	❼	❺	9
➳ 3	9	11	6	-	10	10	12	8
➳ 5	8	9	7	-	❸	7	10	❽

Figure 7.23. *Etat de la matrice du nœud 4 après prise en compte de l'augmentation de coût sur sa voie vers 3*

Vecteur coût minimum V_4= (2,5,1,-,3,7,5,3) devient V_4= (2,5,5,-,3,7,5,8), donc il y a augmentation de coût de 4 pour aller en 3, et de 5 pour aller en 8. Ses décisions de routage sont donc modifiées uniquement pour aller en 3. Il diffuse cette information à ses voisins, les nœuds 1, 3 et 5. Ceux-ci vont augmenter de 4 leur coût pour aller 3 en passant par 4.

En faisant les calculs, on verra que les décisions de routage en 1 ne sont pas modifiées. En effet, jusqu'a présent, 1 envoyait vers 4 les messages à destination de 3, car le coût connu était de 3. Le nouveau coût annoncé par ce chemin devient 7 (3 + 4 augmentation annoncée). Or cette valeur reste inférieure au coût en passant par 2 ou 7. Il n'annoncera donc pas ce changement de coût à ses voisins.

Nous laissons le soin au lecteur de poursuivre l'algorithme jusqu'à son terme en prenant en compte la réduction de coût sur la voie 3 vers 2. Il verra alors que les décisions de routage sont modifiées.

Un des avantages majeurs du routage adaptatif est que chaque nœud n'a besoin de connaître et surveiller que ses voisins. En outre, en cas d'une modification locale du trafic, seul le voisinage concerné est modifié — ce que vous avez pu vérifier sur l'exemple précédent.

Cet algorithme, ou tout au moins des variantes, est très utilisé pour la réalisation de réseaux d'entreprise, basé sur l'interconnexion de réseaux privés (locaux).

Cette famille d'algorithme est plus adaptée au routage de datagramme qu'à l'établissement de circuits virtuels. Il est en effet impossible de garantir pendant l'établissement du CV qu'aucun nœud traversé n'est en train de mettre à jour ses tables, créant ainsi une potentialité d'instabilité et de boucle. Néanmoins, en ajoutant quelques mécanismes pour réduire les boucles éventuelles, il est aussi adapté que les vecteurs de routage, puisqu'il aboutit au même résultat pour cette fonction.

Les algorithmes de types « *Link state* » ou *« Distant vector* » font partie de la famille des algorithmes adaptatifs. Ils sont entre autre très utilisés dans l'Internet [11].

7.3.1.1. *Algorithmes Distance Vector*

L'algorithme *Distance Vector* également appelé algorithme Bellman - Ford, est un algorithme pour lequel chaque noeud échange avec ses voisins sa table de routage, afin de mettre à jour son contenu, pour que celle-ci contienne la plus courte distance vers chaque destination. Pour cela chaque routeur exécutant cet algorithme possède en mémoire une table de routage comprenant une entrée par destination, dont le contenu comprend :

— la destination à atteindre,

— le coût pour atteindre cette destination (exprimé en fonction d'une métrique),

— le prochain voisin en direction de cette destination.

Cet algorithme est implanté dans les protocoles suivants : RIP (Routing Information Protocol), BGP (*Boarder Gateway Protocol*), IGRP (*Interior Gateway Routing Protocol*).

Les étapes de l'algorithme sont :

— chaque routeur est initialisé avec son identificateur propre, le coût des liens vers chacun des ses voisins et un coût nul pour lui-même.

— chaque routeur diffuse périodiquement vers chacun de ses voisins sa table de routage contenant les noeuds du réseau et le coût associé pour les atteindre.

— lorsqu'un routeur reçoit une nouvelle table de routage, il calcule son propre vecteur de distance en effectuant pour chaque entrée de la table les traitements suivants :

- si l'entrée n'est pas présente dans la table, le routeur l'ajoute dans sa propre table.
- si le coût rapporté par un des voisins ajouté au coût pour atteindre ce voisin est inférieur au coût déjà stocké, alors la table de routage est mise à jour avec le nouveau chemin.
- si le coût rapporté par un autre des voisins ajouté au coût pour atteindre ce voisin est supérieur au coût déjà stocké, alors l'entrée correspondante dans la table de routage reste inchangée, puisqu'on sélectionne le plus court chemin vers chaque destination.

Si un routeur découvre qu'un lien vers l'un de ses voisins est rompu, il met à jour l'entrée correspondante de sa table de routage (coût infini).

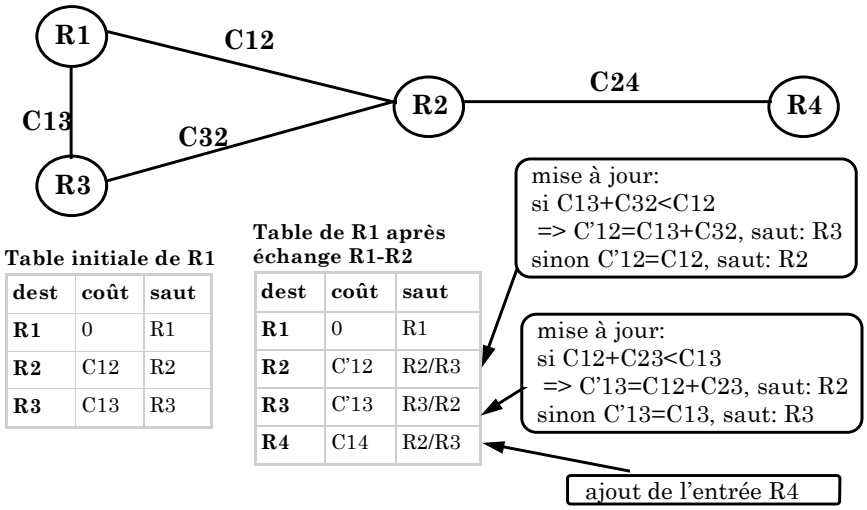


Figure 7.24. *Distance Vector sur un exemple de topologie*

Les tables de routage sont échangées périodiquement, afin de mettre à jour pour chaque routeur, le plus court chemin vers chacune des destinations. La figure 7.24. illustre en partie l'exécution de l'algorithme *distance vector* sur un exemple simple de topologie.

Le problème avec cette méthode est qu'il faut procéder à un certain nombre d'itérations avant que l'algorithme ne converge (c'est à dire jusqu'à ce que le contenu des tables de routage n'évolue plus). Ceci peut induire des boucles lorsque la longueur d'un chemin augmente (exemple : rupture d'un lien de communications) et entraîner une augmentation conséquente du délai de transmission, voire une perte des paquets. Ce problème connu sous le nom d'effet de comptage est illustré à partir de la topologie représentée sur la figure 7.25. Cet exemple considère trois noeuds de communication notés A, B et C connectés entre eux par des liens de coût unitaire.

Supposons que le lien entre les routeurs B et C soit rompu. En remarquant la rupture de cette liaison, le routeur B rejette la table de routage qu'il a précédemment reçue de la part du routeur C et recalcule son vecteur de distance. Malheureusement, le routeur B ne va pas conclure à ce moment que le routeur C est devenu inaccessible; il va établir que son coût vers le routeur C est égal à 3 en se basant sur le fait qu'il est voisin du routeur A (coût de 1 vers A) et que le routeur A lui a reporté qu'il est à une distance (coût) de 2 par rapport au routeur C. Puisque la table de routage du routeur B a changé, ce dernier va transmettre la table de routage modifiée à ses voisins encore en activité (ici le routeur A). Le routeur A puisqu'il a reçu une table de routage modifiée de son voisin B, va recalculer sa propre table et conclure que C est maintenant à une distance égale à 4. Les deux routeurs A et B vont continuer ce

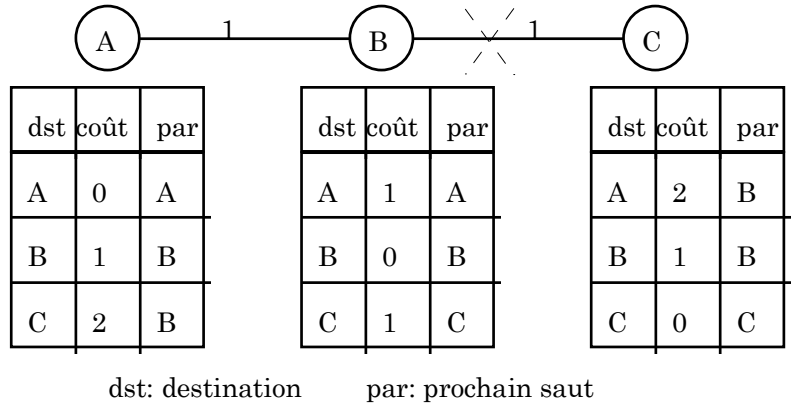


Figure 7.25. *Topologie sensible à l'effet de comptage*

processus et progressivement incrémenter leur distance par rapport au routeur C jusqu'à atteindre la distance maximale autorisée (les routeurs A et B détecteront enfin que la destination C est devenue inaccessible). On a alors la formation d'une boucle de routage induite par l'effet de comptage qui va considérablement freiner la convergence de l'algorithme.

Le diagramme temporel (cf. figure 7.26.) indique les mises à jour du coût effectuées par les routeurs A et B concernant la destination C à chaque itération de l'algorithme et met en évidence l'effet de comptage. Les temps (T1, T2, etc.) indiqués dans ce tableau correspondent aux temps de mises à jour des routeurs A et B après la rupture du lien B-C supposée s'être produite au temps T0.

Temps	T1	T2	T3	T4	T5
routeur A (distance A-C)	2		4		6
routeur B (distance B-C)		3		5	

Figure 7.26. *Diagramme temporel des échanges*

Il importe que l'algorithme ait un temps de convergence faible pour détecter des destinations devenues inaccessibles, c'est pourquoi il faut utiliser des mécanismes pour éviter la formation de boucles afin de garantir la cohérence entre la topologie réelle du réseau et celle perçue par l'algorithme de routage. Plusieurs techniques ont été mises en oeuvre pour accélérer la convergence de cet algorithme :

— le coût infini "fixé".
— l'horizon coupé,
— le protocole de coordination des noeuds,
— les mises à jour déclenchées,

7.3.1.1.1. Coût infini fixé

La première solution envisagée pour accélérer la convergence de l'algorithme *distance vector* est de fixer une valeur limite pour le coût infini, afin d'écourter les boucles et de détecter plus rapidement des destinations inaccessibles. En choisissant une valeur qui n'est pas trop élevée, l'effet de comptage sera plus limité car le coût maximal sera plus vite atteint, par conséquent l'algorithme convergera plus vite. On ne peut pas non plus prendre une valeur trop petite car si l'on veut implanter l'algorithme pour des réseaux de taille suffisamment importante, la valeur choisie doit être supérieure au diamètre du réseau. Pour le protocole RIP cette valeur a été fixée à 16.

7.3.1.1.2. Horizon Coupé

La philosophie de cette technique est qu'il est inutile pour un routeur d'indiquer l'accessibilité d'une route au voisin par lequel cette route a été apprise, puisque ce voisin n'a aucune raison de passer par un routeur plus éloigné pour atteindre la destination considérée. Les deux versions possibles de cette technique sont :

— *l'horizon coupé simple* qui consiste pour un routeur à ne pas émettre vers son voisin des mises à jour de sa table contenant l'entrée vers une destination, si c'est ce voisin qui l'a informé sur cette destination,

— *l'horizon coupé avec "poisoned reverse"* qui consiste pour un routeur à affecter un coût infini vers une destination, lorsqu'il envoie une mise à jour de sa table de routage vers le voisin qui l'a informé sur cette destination.

La technique de l'horizon coupé appliquée à l'exemple précédent permet d'éviter l'occurrence de la boucle de routage dans le cas de la figure 7.25. En effet, en appliquant la règle de l'horizon coupé, le routeur A n'annoncera pas au routeur B le chemin pour atteindre le routeur C puisqu'il a appris l'existence du routeur C par l'intermédiaire du routeur B. Ceci permet au routeur B d'éviter d'être trompé en pensant qu'il y a une route vers C par A.

Elle n'est cependant pas la panacée car le problème des boucles est difficile à résoudre quand les noeuds ne connaissent que leurs voisins et non pas la topologie complète du réseau. La technique de l'horizon coupé, qui tend à réduire ce phénomène, n'empêche pas une succession de routeurs de générer des boucles. C'est le cas pour l'exemple traité sur la figure 7.27.

Lorsque le lien entre B et D se rompt, le routeur B le détecte et écrit dans sa table un coût infini en direction de D car la technique de l'horizon coupé empêche les routeurs C et A d'informer le routeur B qu'ils sont à un coût de 2 par rapport à la destination D. Par contre les routeurs A et C pensent réciproquement que le meilleur chemin pour atteindre la destination D est de passer à travers l'autre et concluent que A peut atteindre D en passant par C avec un coût de 3. Il en résulte que le routeur B après réception de la table de routage en provenance de A va penser pouvoir atteindre le routeur D en passant par A avec un coût de 4. Le problème de boucle de routage dû à l'effet de comptage existe donc toujours dans ce cas.

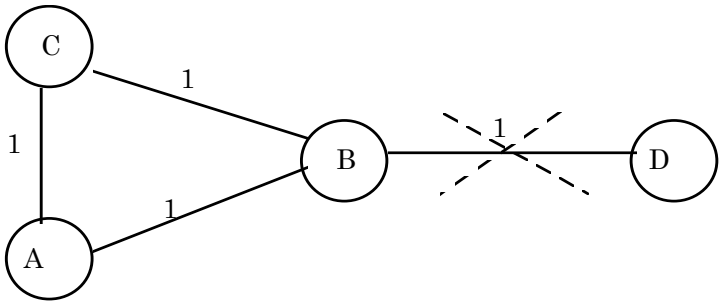


Figure 7.27. *Topologie incompatible avec l'horizon coupé*

Le diagramme temporel sur la figure 7.28. indique les mises à jour du coût effectuées par les routeurs A, B et C concernant la destination D à chaque itération de l'algorithme et met en évidence l'effet de comptage qui existe encore dans ce type de topologie. Les temps (T1, T2, etc.) indiqués sur la figure 7.28. correspondent aux temps de mises à jour des routeurs A, B et C après la rupture du lien B-D supposée s'être produite au temps T0.

Temps	T1	T2	T3	T4	T5	T5
routeur A (distance A-D)		3			6	
routeur B (distance B-D)			4			7
routeur C (distance C-D)	2			5		

Figure 7.28. *Diagramme temporel correspondant à al figure figure 7.27.*

7.3.1.1.3. Protocole de Coordination

Des algorithmes basés sur le *Distance Vector* et éliminant les problèmes de bouclage ont été conçus en forçant les noeuds à participer à un protocole de coordination. Ceci est réalisé en contrôlant l'ordre dans lequel les noeuds diffusent leurs messages de mise à jour et calculent leur routes, suite à un accroissement du coût d'un lien. Pour cela tous les noeuds qui se trouvent en amont du lien rompu dans le graphe de topologie de la destination sont progressivement mis dans un état d'attente leur permettant de mettre seulement à jour le coût de leurs entrées sans pouvoir changer le prochain saut vers la destination avant que l'augmentation du coût de la route ne soit propagé à tous les noeuds en amont du graphe de topologie. Un noeud reste dans cet état d'attente jusqu'à ce que tous les noeuds en amont du graphe aient mis à jour leur coûts reflétant l'augmentation du coût du lien. Ainsi en purgeant l'ancienne information, un noeud ne provoquera jamais une boucle en sélectionnant un noeud plus en amont dans le graphe de topologie. Cette approche requiert en contrepartie un plus grand nombre de messages car les noeuds coordonnent leurs

calculs de table de routage par échanges de messages de contrôle supplémentaires. De plus, elle n'apporte qu'un faible gain en vitesse de convergence.

7.3.1.1.4. Mises à jour déclenchées

Une autre manière d'accélérer la convergence du *Distance Vector* est de déclencher des mises à jour sur événements. La règle à appliquer est la suivante : à chaque fois qu'un routeur est informé du changement de métrique d'une route, il n'a pas à attendre l'instant d'une mise à jour régulière pour diffuser l'information : il la propage après expiration d'un *timer* de déclenchement de courte durée. Cela permet de prendre en compte des modifications de topologies relativement tôt, sans pour autant générer trop de trafic qui serait induit par une succession de mises à jour immédiates en phase d'instabilité (nombreux changements de topologie) du réseau.

En effet, durant la période entre deux mises à jour régulières, un nombre important de changements de topologie peuvent intervenir et entraîner une incohérence entre l'information de routage maintenue par l'algorithme et la topologie réelle du réseau. Cette incohérence peut causer des problèmes d'instabilité par la formation de boucles de routage perturbant la transmission des données, il convient donc d'effectuer des mises à jour déclenchées en choisissant une valeur appropriée pour le *timer* de déclenchement.

La valeur fixée pour les mises à jour déclenchées résulte d'un compromis entre la nécessité de maintenir la cohérence de l'information de routage et de sauvegarder le plus de bande passante pour le trafic de données. Il faut éviter d'avoir à transmettre une succession de mises à jour déclenchées, pour cela il est préférable d'attendre plusieurs changements internes au niveau d'un routeur lorsque ces changements sont rapprochés afin de ne transmettre qu'une seule mise à jour. Le choix de cette valeur dépend de la vitesse d'évolution des changements de topologie du réseau.

Malheureusement les mises à jour déclenchées et les mises à jour régulières peuvent se produire simultanément conduisant à des problèmes d'incohérence. Par exemple il est possible qu'après la réception par un routeur d'une mise à jour déclenchée, celui-ci reçoive une mise à jour régulière de la part d'un autre routeur n'ayant pas reçu cette mise à jour déclenchée. On a alors une mise à jour effectuée à partir d'une information obsolète. Cependant ce genre de situation est rare car les mises à jour déclenchées ont des *timers* nettement plus courts que la période de mise à jour régulière.

Ces deux dernières techniques (coût infini fixé et timer de déclenchement) sont exploitées dans le protocole RIP déployé dans le réseau Internet.

7.3.1.2. *Link State (LS)*

Contrairement à l'algorithme du *Distance Vector*, chaque routeur échange avec ses voisins la description des liens qu'il a avec ceux-ci. Cette information locale est relayée de proche en proche et permet à chaque routeur de se bâtir sa propre topologie du réseau, avec lui-même comme racine du graphe de topologie.

Avec les algorithmes basés sur le *Link State*, chaque noeud maintient toute la topologie du réseau, par acquisition du voisinage et diffusion des changements de topologie à tous les noeuds du réseau, pour calculer le plus court chemin vers chaque destination par l'algorithme de Dijkstra.

Cet algorithme est mis en oeuvre dans les protocoles IDRP (*Inter Domain Policy Routing*), IS-IS (*Intermediate Systems to* Intermediate *Systems*) et OSPF (*Open Shortest Path First*) déployé dans le réseau Internet.

L'algorithme *Link State* se déroule en 4 étapes qui sont les suivantes :

— la phase d'identification des voisins,

— la phase de construction du paquet d'état des liens à diffuser,

— la phase de propagation du paquet d'état des liens,

— la phase de calcul des routes suivant l'algorithme du plus court chemin de Dijkstra.

7.3.1.2.1. Identification des voisins

Lors de cette étape, chaque routeur s'identifie auprès de ses voisins en envoyant à chacun d'eux des paquets spécifiques indiquant sa présence. Les paquets reçus permettent à chaque routeur d'identifier les noeuds qui sont ses propres voisins.

7.3.1.2.2. Construction du paquet d'état des liens

Chaque routeur construit un paquet, dénommé paquet d'état des liens (LSP : *Link State Packet*) qui contient une liste de ses voisins et leurs coûts associés, puis émet ce paquet vers tous les noeuds du réseau :

— lorsqu'il découvre que :
  • il a un nouveau voisin,
  • le coût d'un lien vers un voisin a changé,
  • le lien vers un de ses voisins est rompu.

— périodiquement, période de rafraîchissement des tables de routage, afin de garantir la synchronisation des bases de données de routage maintenues au niveau de chaque noeud et renforcer ainsi la fiabilité de l'algorithme.

Initialement, l'algorithme *Link State* spécifiait de diffuser l'état des liens de tous les noeuds périodiquement. Le fait de limiter la diffusion des paquets d'états des liens aux seuls liens modifiés, c'est à dire aux changements de topologie a permis de réduire sensiblement le trafic généré par l'algorithme *Link State*.

7.3.1.2.3. Propagation du paquet d'état des liens

Lorsqu'un paquet d'état des liens est généré il est diffusé à tous les routeurs du réseau sans exception.

Cette étape est délicate car il est nécessaire que tous les routeurs au sein du réseau dispose de la même information de routage pour calculer leurs routes, sinon les routes risquent d'être incohérentes et peuvent entraîner de mauvaises décisions de routage. On doit donc s'assurer que cette diffusion est réalisée sans problème, c'est à dire que ce paquet est bien reçu par tous les routeurs. On doit également s'assurer que cette diffusion n'engendre pas de duplicata afin de ne pas gaspiller les ressources en bande passante du réseau au détriment des applications réseau des utilisateurs. Par conséquent si un paquet d'état des liens reçu sur un noeud est identique à un précédent stocké en mémoire et provenant de la même source, il est ignoré et le relayage de ce paquet vers les autres noeuds du réseau est stoppé.

Une des plus grosses difficulté à résoudre avec cette phase de propagation est de pallier au déséquencement des paquets qui peuvent intervenir dans le réseau. En effet, si un routeur R reçoit plusieurs paquets LSP d'une même source S, on ne peut pas garantir que l'ordre d'émission des paquets par la source S correspond à l'ordre de réception des paquets par le routeur R, puisque les paquets LSP peuvent emprunter des chemins différents pour atteindre la destination.

La solution adoptée face à ce problème est d'utiliser dans chaque paquet LSP un champ numéro de séquence. Le champ numéro de séquence est affecté par la source lors de la génération du paquet LSP, ce numéro étant incrémenté pour chaque nouveau paquet LSP créé afin de permettre aux autres noeuds d'identifier parmi les paquets LSPs reçus (en provenance d'une source donnée) celui qui est le plus récent. Ainsi lorsqu'un routeur R reçoit un paquet LSP généré par une source S, il accepte le LSP et le réécrit à la place de tout LSP en provenance de la source S si le numéro de séquence du LSP reçu est supérieur au numéro de LSP qu'il avait en mémoire.

Le champ numéro de séquence n'est pas suffisant pour assurer le bon fonctionnement de l'étape de propagation des paquets LSP. Les valeurs du champ numéro de séquence peuvent être corrompues par un routeur et atteindre une valeur proche du maximum de ce champ. Or l'algorithme doit pouvoir continuer à fonctionner correctement même si le numéro de séquence d'un LSP atteint la valeur maximale. Dans ce but, le champ durée de vie est exploité pour valider ou de rejeter les paquet LSP. Ce champ commence à une certaine valeur et est décrémenté par les routeurs pendant le stockage du LSP en mémoire. Lorsque ce champ est nul, le paquet LSP stocké en mémoire est rejeté et le prochain LSP reçu est accepté indépendamment du numéro de séquence.

Un nouveau paquet LSP reçu est forcément de durée de vie non nulle car tout paquet LSP de durée de vie nulle est rejeté et n'est par conséquent plus propagé au travers du réseau.

7.3.1.2.4. Calcul des routes

Lorsqu'un routeur reçoit un nouveau paquet LSP, il exploite l'information qu'il contient pour mettre à jour son information de routage afin de maintenir une vision complète de la topologie du réseau. Il peut alors effectuer le calcul des routes vers chaque destination pour construire sa propre table de routage. Cette étape de calcul des

routes vers chacune des destination repose sur l'algorithme du plus court chemin de Dijkstra.

— Remarques sur l'algorithme *Link State*

L'algorithme du *Link State* est caractérisé par un temps de convergence bien meilleur que celui du *Distance Vector*. En effet, le *Link State* n'engendre pas de boucles, sauf celles (de très courte durée) qui peuvent apparaître pendant le délai de propagation des messages de changement de topologie, pour atteindre tous les noeuds du réseau. Ceci le rend particulièrement efficace pour trouver un chemin de remplacement lorsqu'un noeud ou un lien disparaît. Cet algorithme particulièrement robuste, permet aussi le partage de trafic sur plusieurs liens de même coût pour atteindre une destination, contrairement au *Distance Vector*.

En revanche, Il est assez coûteux en temps de calcul du fait de l'algorithme de Dijkstra et en taille mémoire du fait du maintien de toute la topologie du réseau au niveau de chaque noeud. Cependant, son principal inconvénient est qu'il génère beaucoup de trafic lors de la diffusion des changements de topologie, puisque ceux-ci sont transmis à tous les noeuds du réseau, ce qui est pénalisant dans une configuration très dynamique.

Les critères qui permettent de comparer deux algorithmes de routage sont :

— Vitesse de convergence, c'est à dire le temps pris par l'algorithme pour construire de nouvelles tables de routage,

— Coût CPU d'exécution de l'algorithme local à chaque nœud :

• Dijkstra requiert une charge CPU proportionnelle au nombre de liens du réseau et au logarithme du nombre de noeuds dans le réseau (le logarithme caractérisant la complexité pour trouver le plus court chemin dans la structure des chemins candidats). Si l'on considère un réseau de N noeuds avec en moyenne k liens par noeud, cela donne un chiffre égal à : N*k*log(N).

• L'exécution de l'algorithme du *Distance Vector* nécessite de traiter k vecteurs de distance de N entrées où N est le nombre de noeuds du réseau et k le nombre moyen de liens par noeud. La charge CPU est donc proportionnelle à N*k.

— Coût mémoire, dans l'algorithme *Distance Vector*, chaque routeur a besoin de garder en mémoire les tables de routage de N entrées de ses k voisins. La mémoire requise est donc proportionnelle à : k*N. Dans le cas de l'algorithme *Link State*, chaque routeur a besoin de garder en mémoire N paquets LSP (un pour chaque noeud du réseau), chacun de ces LSP contenant k entrées.

— Bande Passante consommée par les messages protocolaires utilisés pour les échanges d'information.

7.3.2. *L'inondation*

L'objectif du routage est de trouver le trajet optimal. Au lieu de le faire par calcul, l'inondation le fait de manière expérimentale. Lorsqu'un nœud veut trouver le meilleur chemin pour aller vers un autre nœud, il envoie un paquet de recherche de

chemin à tous ses voisins. Le paquet contient son adresse (nom) et bien sûr l'adresse du destinataire.

Chaque nœud, lorsqu'il reçoit un paquet de recherche de chemin :

— ajoute son nom dans le corps du paquet et des informations de coût locales s'il n'est pas le destinataire. Puis il envoie une copie du paquet à tous ses voisins, à l'exception de tout voisin ayant déjà vu le paquet. Il suffit d'utiliser la liste contenue dans le paquet. Cette précaution permet d'éviter les boucles, de renvoyer à l'émetteur ou à un émetteur précédent, c'est-à-dire un nœud ayant déjà vu le paquet ;

— regarde s'il est le destinataire. Si tel est le cas, le paquet contient une route recherchée ;

La figure 7.29. montre les différentes étapes de cette diffusion. La représentation sous forme de graphe ne peut pas du tout représenter les délais d'acheminement. Elle laisse croire que le meilleur chemin est le plus court en nombre de nœuds, ce qui est en général faux. Vous trouverez des exemples en lisant les chemins sur les matrices de coût.

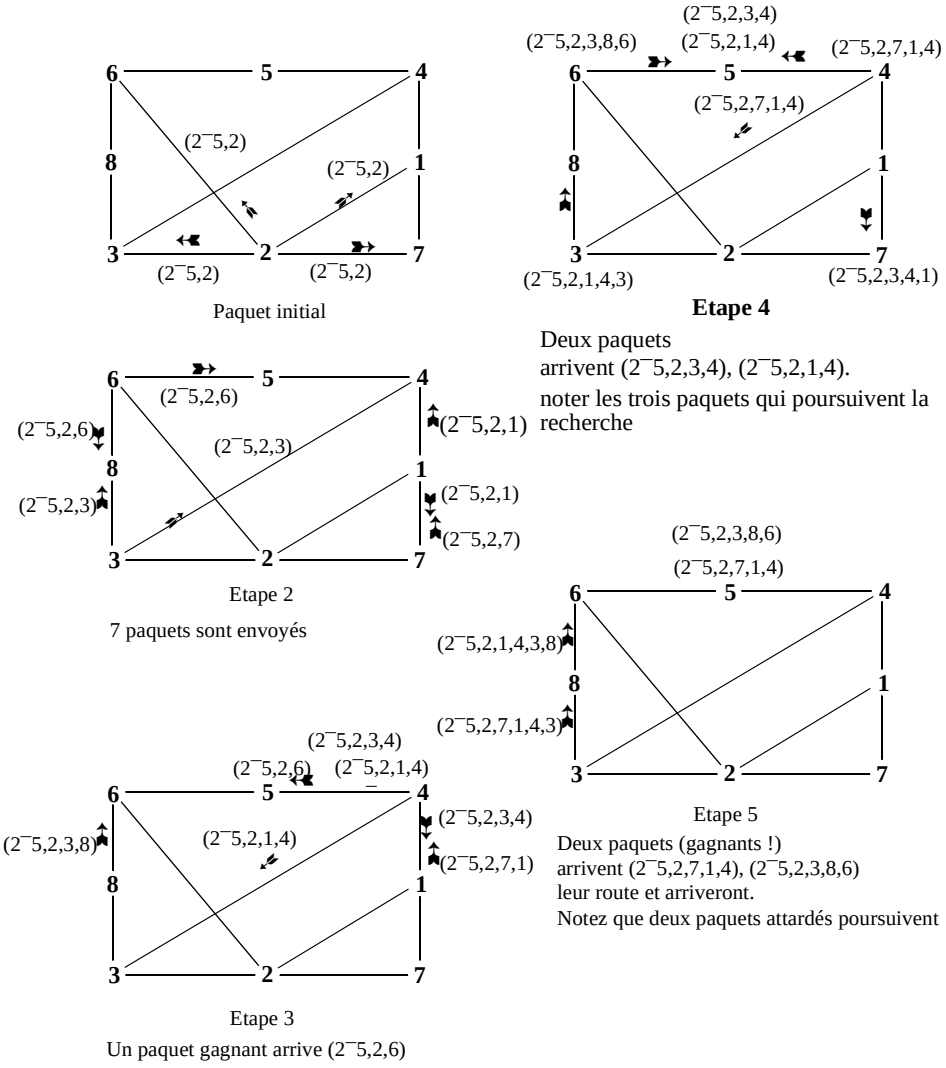
Dans cet exemple, le nœud 5 obtient 7 routes différentes à la fin de la recherche. Le nœud destinataire choisit parmi les chemins trouvés le plus intéressant, et renvoie un message par cette route au demandeur. Le demandeur mémorise dans une table de routage les chemins qu'il connaît et dont il a besoin. Il n'y a pas besoin de tables de routage dans les nœuds. La technique est appelée « source routing » parce que chaque message porte dans son en-tête réseau la liste des nœuds par lesquels passer pour atteindre la destination. Le nœud 5, sur cet exemple, a juste besoin d'inverser le chemin trouvé pour construire son message.

Le nœud demandeur, en l'occurrence, mémorise le chemin trouvé et l'heure à laquelle il a été trouvé. Il utilisera ce chemin pour tous les nouveaux messages pendant une durée Δ, appelée durée de validité. Dans cette technique, chaque nœud conserve une liste des chemins connus vers toutes les destinations demandées par ses utilisateurs (cf. figure 7.31.). Les nœuds du réseau n'ont pas besoin de tables de routage.

La durée de validité Δ du chemin trouvé a pour but de forcer le nœud à vérifier périodiquement que la route obtenue reste bien la meilleure. Cela réalise, comme dans l'algorithme centralisé, l'adaptation aux changements d'état du réseau.

Cette technique de source routing permet de réduire la table de routage chez chaque abonné aux seules destinations effectivement utilisées, comme le montre la figure 7.30. L'âge permet de mesurer le temps qui s'est écoulé depuis la détermination de ce chemin : il ne peut excéder la durée de validité. Chaque nœud cherche un chemin quand un utilisateur en a effectivement besoin ou quand un chemin connu atteint sa durée de validité.

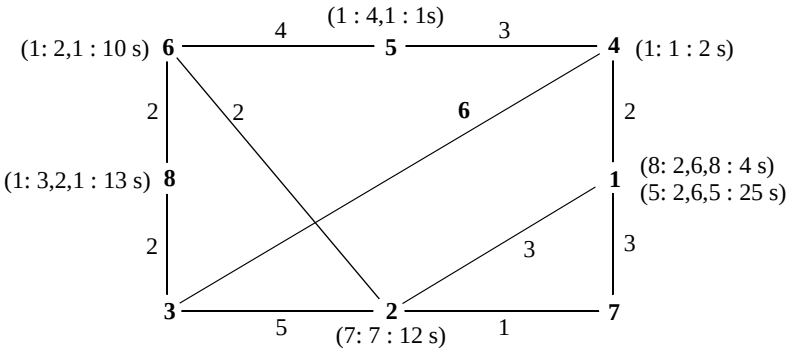
Cette technique permet de trouver des abonnés mobiles, c'est-à-dire qui ne sont pas toujours rattachés au même nœud dans le réseau. L'adresse dans ce cas ne porte pas d'information sur la localisation géographique et une traduction « nom-position



Il n'y a aucune raison pour que ces événements soient synchrones. Ceci est uniquement un artifice de dessin

Figure 7.29. *Itérations successives de l'algorithme d'inondation d'un message issu de 2 vers 5.*

géographique de rattachement » n'est pas possible. Il faut établir dynamiquement la correspondance.



Il n'est pas garanti que les routes aller et retour soient identiques. Chaque vecteur a un âge qui correspond à la durée qui s'est écoulée depuis son évaluation. Cela permet d'oublier les vecteurs trop anciens afin de rechercher une nouvelle route et ainsi de prendre en compte les changements dans le réseau. Sur la figure on observe que les deux vecteurs les plus anciens ne correspondent plus à la route optimale.

Figure 7.31. *Etat des vecteurs de routage (destination : route : âge) dans différents nœuds en fonction des besoins de chaque nœud en communication*

Destination	Chemin	Age
5	2, 6	10 s
4	2 ,4	24 s

Figure 7.30. *Exemple de table qui sera maintenue par un abonné du nœud 2 utilisant l'algorithme du source routing*

Par contre, la diffusion est très saturante pour le réseau, puisqu'elle génère un grand nombre de messages inutiles. Il est possible de réduire ce nombre en définissant le diamètre, D, du réseau comme étant le nombre minimal de nœuds qu'un message doit traverser pour rejoindre deux nœuds les plus éloignés du réseau. On arrêtera la diffusion lorsque la longueur du chemin parcouru a dépassé $D + \Delta t$, Δt étant une marge que l'on se donne pour supporter les pannes de voie.

7.4. Conclusion

C'est le rôle d'un algorithme spécifique de calculer la meilleure route possible entre tous les couples de correspondants possibles. Cet algorithme est exécuté soit périodiquement, soit sur événement réseau (exemple : panne de routeur). Cet algorithme peut être :

— centralisé : c'est-à-dire exécuté par un seul site de gestion qui envoie les résultats aux routeurs (cas des tables de routage réparties). Il est évident que, si la table de routage est centralisée, le centre de gestion effectue l'algorithme de recherche des chemins optimaux. Cette approche a l'avantage de fournir un résultat cohérent à tout instant pour les choix de chemin. Si les tables sont réparties, le site ayant exécuté l'algorithme distribue aux routeurs les résultats qui le concerne. Pendant ce temps, des décisions cohérentes peuvent être prises puisque l'on ne peut pas garantir que tous les routeurs recevront leur fraction de table simultanément. Les informations de changement d'état local à chaque routeur doivent être envoyées au site de gestion ;

— distribué : chaque routeur effectue l'algorithme en fonction des informations locales et de celles qu'il reçoit des autres routeurs. Les exécutions ne sont bien évidemment pas synchrones. Un changement peut intervenir à tout instant en un point quelconque du réseau.

Les tables locales évoluent donc de manière différentes dans chaque routeur. Pour obtenir un résultat cohérent, il faut que l'algorithme converge en un nombre d'échanges entre routeurs raisonnable. La convergence de l'algorithme est toujours une fonction proportionnelle au nombre de routeurs. La convergence prend un certain temps qui doit être inférieur à la durée entre deux invocations de l'algorithme, sinon la solution va diverger.

De ce qui précède, il ressort deux éléments :

— il faut que les routeurs envoient des informations sur l'état de leurs lignes pour pouvoir calculer le meilleur chemin. Ces informations mettent un délai pour arriver. Le calcul du meilleur chemin est donc toujours effectué sur des valeurs périmées. Si l'état du réseau change rapidement, les décisions prises risquent d'être néfastes et donc de conduire à des situations de congestion ;

— les tables ne sont pas mises à jour de manière synchrone. Il peut donc arriver que des boucles soient crées momentanément ; ce qui impliquera par exemple qu'un datagramme passera plusieurs fois par le même routeur avant d'atteindre son destinataire. Cet élément est capital pour comprendre les causes de déséquencement et les accroissements possibles du délai de transfert d'un message qui peuvent se produire au niveau réseau. Cela implique au niveau transport des mécanismes complexes pour se protéger contre des durées de transmission exceptionnellement longues. On appelle « fantôme » des messages attardés dans le réseau.

Il est nécessaire que chaque routeur effectue en permanence une mesure des grandeurs qui vont être utiles à l'exécution de l'algorithme de calcul des chemins. Ces grandeurs sont : le délai de traversée du routeur, les activités des lignes dont il dispose, le délai d'acheminement des messages, le taux d'utilisation des ressources locales, la tailles des files d'attente, etc. Elles vont permettre de calculer un coût pour une liaison. C'est une fonction de coût global qui va être minimisée dans le calcul des tables de routage. Une liaison coupée (en panne, absente…) aura un coût infini afin d'interdire ce chemin.

Il existe de nombreux algorithmes de routage [11], [27], [13]. Nous n'avons donné ici que la philosophie des principes utilisés. Les algorithmes de routage sont le plus brillant exemple d'algorithme réparti complexe. Ils continuent à faire l'objet de

nombreuses études pour les réseaux, les machines parallèles… Ils sont indépendants des requêtes des utilisateurs et leur sont totalement transparents.

Il est important de se rappeler que la fonction de routage introduit des délais aléatoires pour la transmission des messages dont les couches utilisatrices du service devront s'accommoder.