Multiroutage par interface d'entrée

Pascal Mérindol, Jean-Jacques Pansiot, Stéphane Cateloin

Equipe Réseaux et Protocoles, LSIIT - Pôle API Boulevard Sébastien Brant 67400 ILLKIRCH FRANCE

Les protocoles de routage actuellement déployés ne profitent pas pleinement et équitablement des ressources que leur offre le réseau physique. D'une part, ces protocoles et leurs algorithmes de recherche opératoire sous jacents (Dijkstra, Bellman-Ford) ne maximisent pas la bande passante disponible entre deux points. D'autre part, le principe de sous-optimalité du meilleur chemin dans un graphe, qui permet à OSPF de router au saut par saut, entraîne un partage non équitable des ressources, aussi bien en termes de nombre de routeurs sollicités que de bande passante consommée. Le multiroutage permet de tirer profit du maillage du réseau physique et de partager plus équitablement les ressources dont il dispose. Dans cet article, nous proposons une solution extensible de multiroutage saut par saut capable de déployer plus de chemins qu'avec les propositions distribuées existantes.

Keywords: multiroutage, plus courts chemins multiples, ingénierie de trafic IP, algorithmique des graphes

1 Introduction

L'émergence de nouveaux besoins sur Internet nous amène à développer de nouvelles techniques de routage améliorant la qualité de service par défaut. Le routage multichemins, ou multiroutage, est une solution souvent désignée qui se décline en deux catégories. Les méthodes de multiroutage à la source, comme celle développée dans [KKDC05], combinent un algorithme de recherche dans les graphes (k-meilleurs-chemins [Epp94], CRA [NZ01]) pour générer les multichemins, à un protocole de signalisation pour les mettre en place (MPLS [RVC01], RSVP-TE [ABG$^+$01]). Il s'agit d'un routage à la source avec marquage préalable des chemins (sans que la route explicite ne soit incluse dans l'en-tête des paquets) dont l'extensibilité en terme de nombre de couples entrée-sortie est limitée. Certes, ces techniques peuvent être déployées en bordure de réseau pour étendre leur extensibilité en s'adressant à un agrégat de flux. Cependant, la répartition de charge ne pourra se faire que sur les routeurs d'entrée. Le routage saut par saut (ou distribué) dispose quant à lui d'atouts importants. Il s'agit, d'une part, d'une politique plus extensible dans la mesure où sa complexité est liée au nombre de destinations à atteindre et non pas à la quantité de couples source-destination *(s,d)*. D'autre part, cette politique présente une plus grande capacité de réaction aux variations de charge. En revanche, les protocoles proposées (MPATH [Vut01], ECMP [Moy98]) profitent seulement d'un ensemble de multichemins réduit car les conditions garantissant l'absence de boucles de routage sont suffisantes mais pas nécessaires. Dans cet article nous présentons un protocole capable de générer et de mettre en œuvre un nombre de multichemins important, accroissant ainsi le flot maximun entre deux points et les possibilités de redirection pour éviter les congestions.
La partie 2 présente les fondements de notre protocole. La partie 3 spécifie l'algorithme de construction à la source (3.1) et introduit succintement la phase de validation distribuée (3.2). Nous concluons sur les avantages apportés par notre méthode et discutons des perspectives de notre travail dans la dernière partie.

2 Présentation générale

2.1 Propositions existantes

Le multiroutage distribué s'intéresse seulement aux multichemins dont le premier saut est différent par destination. Il doit néanmoins s'assurer que la liberté de routage ajoutée n'induise pas la formation de boucles entre les routeurs capables de partager la charge. Pour cela, un routeur s vérifiera par exemple que son voisin v propose un meilleur coût (calculé au préalable sur v, comme sur s, avec Dijkstra) strictement

inférieur au sien pour une destination d. Plus formellement, on peut définir la condition $C_1(v,d) < C_1(s,d)$ qui permet de valider tous les chemins via v vers d pour s (notations données en TAB.1*). Cette relation, utilisée dans [Vut01], nécessite une phase de synchronisation entre routeurs adjacents. Que ce soit avec cette condition ou en vérifiant l'égalité des coûts pour les chemins calculés sur s, $C_j(s,d) = C_1(s,d)$ $_{(1<j\leq k^+(s))}$ pour valider les chemins d'indice inférieur ou égal à j calculés par s (ECMP), les routeurs se privent d'exploiter réellement leur degré sortant dans la mesure où les circuits que forment s et d ne peuvent être utilisés. En effet, il est possible de profiter des circuits présents sur un graphe G, modélisant un réseau donné, si on s'assure que la formation de boucles de routage (flux parcourant entièrement un circuit) est impossible.

2.2 Le multiroutage distribué par interface entrante

L'originalité de notre protocole est la distinction faite par les routeurs sur l'interface d'entrée des flux pour les rediriger de manière plus fine. Cette distinction permet aux routeurs de profiter des circuits présents sur G sans que les flux de données ne bouclent. La figure 1 décrit les différentes étapes de notre procédé.

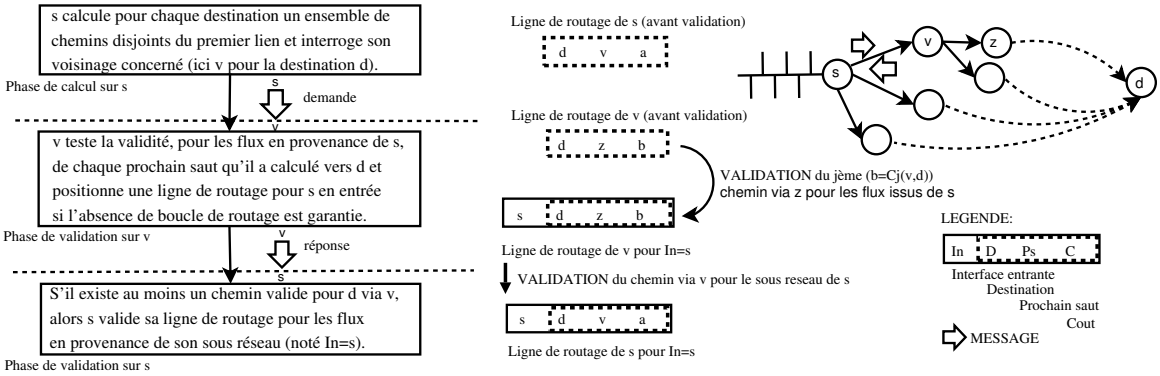

FIG. 1: Ajout de l'interface d'entrée dans une ligne de routage

$G(N,A,c)$	graphe G doté d'un ensemble de noeuds N et d'arcs A valués par c(a)>0.
$a = \{a.x, a.y\}$	arc reliant un noeud a.x à un noeud a.y.
$k^-(s)$, $k^+(s)$	degrés entrant et sortant d'un noeud s.
$P_j^m(s,d) = \{a_1,...,a_m\}$	chemin de m sauts reliant s à d dont le classement selon la métrique C est j.
$C_j^m(s,d) = \sum_{i=1}^{m} c(a_i)$	j^{eme} coût calculé par s pour la destination d $_{(1\leq j\leq k^+(s)),(0<m<\lvert N\rvert)}$.
$branche_p(s)$	tous les chemins $P_1(s,d)$ dont le premier saut est le noeud p.
arc transverse	arc reliant deux branches : $branche_{p1}(s)$ et $branche_{p2}(s)$ $_{(p1\neq p2)}$.
$P(f) = p$	fonction père renvoyant le prédécesseur p d'un noeud f dans un chemin P_1.
chemin transverse $Pt_j^m(s,d)$ $_{(1<j\leq k^+(s))}$	chemin dont les arcs a_i $_{(1\leq i\leq m-1)}$ forment un meilleur chemin : $P_1^{m-1}(s,a_{m-1}.y)$ et tel que l'arc a_m est transverse.
chemin transverse retour $Ptr_j^m(s,d)$ $_{(1<j\leq k^+(s))}$	chemin dont les arcs a_i $_{(\forall i\in[1,w])}$ forment un chemin transverse $_{(0<w<m)}$: $Pt_\alpha^w(s,a_w.y)$ $_{(1<\alpha\leq k^+(s))}$, puis un meilleur chemin $P_1^{m-w}(d,a_w.y)$, $_{(\forall i\in[w+1,m])}$.
chemin transverse avant $Pta_j^m(s,d)$ $_{(1<j\leq k^+(s))}$	chemin dont les arcs a_i $_{(\forall i\in[1,w])}$ forment un chemin transverse $_{(0<w<m)}$: $Pt_\alpha^w(s,a_w.y)$ $_{(1<\alpha\leq k^+(s))}$ ou transverse retour $Ptr_\beta^w(s,a_w.y)$ $_{(1<\beta\leq k^+(s))}$, suivi d'un meilleur chemin : $P_1^{m-w}(a_w.y,d)$, $_{(\forall i\in[w+1,m])}$.

TAB. 1: Définitions

* Pour $\alpha < \beta$ on a $C_\alpha(s,d) \leq C_\beta(s,d)$. Si deux chemins ont leur premier saut en commun, seul le chemin de meilleur coût est considéré.

3 Description du protocole

Cette partie présente les deux phases de construction des tables de routage en vérifiant, avec une granularité basée sur l'interface entrée, l'absence de boucles de routage.

3.1 Phase de calcul à la source

Il s'agit ici de définir un algorithme capable de calculer un ensemble de multichemins, dont le premier saut diffère d'une source s vers chaque sommet d'un graphe G, en un temps d'exécution du même ordre que celui de Dijkstra [Dij59]. D'une part, calculer l'ensemble complet de ces chemins paraît très coûteux : $O(|N|^3)$ sur un réseau fortement maillé. D'autre part, le volume des tables de routage ne doit pas être excessif : or avec notre procédé un routeur s aura au pire $k^-(s) \times k^+(s) \times |N|$ lignes de routage, réparties sur ses $k^-(s)$ interfaces d'entrée, ce qui semble raisonnable dans la mesure où un routeur se doit d'être dimensionné (au sens ressources matérielles) en fonction de son arité. Le nombre de chemins générés sur un nœud s pour une destination d est inférieur ou égal à $k^+(s)$. Cependant, grâce au comportement distribué de notre méthode, le nombre de routes utilisées pour une paire (s,d) est fonction du nombre de prochains sauts validés de proche en proche à partir de s jusqu'à d (voir paragraphe 3.2). La figure 3 illustre le fonctionnement de l'algorithme de recherche (ALG.1). Les arcs pleins (meilleurs chemins P_1) sur la figure 3 forment un arbre des plus courts chemins enraciné en *4* (arbre lexicographique sur l'exemple) constitué de trois branches. Les arcs en pointillé (« arc transverse ») sont ignorés lors de l'exécution de Dijkstra. Les arcs à tiret (arcs $\{fils, pere\}$, l'existence des arcs est considérée symétrique sur le graphe orienté G) sont utilisés pour générer des chemins « transverse retour ». L'arc $\{1,2\}$ n'est pas marqué par des tirets car aucun arc « transverse » n'atteint le nœud *1*. Cet arc n'est donc pas exploitable pour former un chemin « transverse retour ». Enfin, on utilise les arcs pleins pour propager de père en fils les chemins créés précédemment, « transverse ou transverse retour », afin de générer des chemins « transverse avant ». Notre algorithme calcule un sous-ensemble de chemins, où les arcs a_1 diffèrent pour une même destination, d'une source s vers tout les sommets d d'un graphe G. Plus précisement, il calcule toutes les possibilités de routage « une branche distante », autrement dit tout les premiers sauts possibles pour chaque destination à une branche d'écart de la branche d'un arbre (selon le balayage des successeurs) des plus courts chemins. L'ensemble Mc, généré par l'algorithme 1, est composé des premiers sauts représentants pour une racine s les meilleurs chemins distants d'une branche. Ces chemins comportent au maximum un arc « transverse » et n'appartiennent pas à la même branche, ils constituent les lignes (avant validation) de la table de routage illustrées en figure 1. Sur la figure 2, nous considérons le coût des arcs comme homogène. Ainsi, on obtient (FIG.3) une représentation graphique de l'exécution de l'algorithme 1, pour *s=4*, sur laquelle on observe par exemple une branche $branche_2(4) = \{P_1^1(4,2), P_1^2(4,1), P_1^2(4,3)\}$, un chemin « transverse » $Pt_2^2(4,2) = \{\{4,5\},\{5,2\}\}$, un chemin « transverse retour » $Ptr_3^3(4,2) = \{\{4,6\},\{6,3\}\{3,2\}\}$ et deux chemins « transverse avant » $Pta_2^3(4,1) = \{\{4,5\},\{5,2\},\{2,1\}\}$ et $Pta_3^4(4,1) = \{\{4,6\},\{6,3\}\{3,2\}\{2,1\}\}$. L'arc $\{1,3\}$ est ignoré par *4* dans la mesure où c'est au nœud *2* de le prendre en compte dans ses calculs (arc « transverse » pour *s=2*). Le paragraphe suivant explique notamment comment les nœuds *2* et *4* peuvent s'entendre pour synchroniser leurs tables de routage, en activant une ligne de routage pour leur interface d'entrée respective, si la validité du premier saut est prouvée. Avec un tas de Fibbonnaci pour modéliser le tableau Tm, et une matrice Mc indexée en colonne par interface sortante, on obtient une complexité en $O(|N|log|N| + |A| + |N| \times k^+(s))$ au pire.

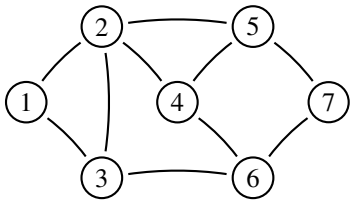

FIG. 2: Un exemple de graphe G

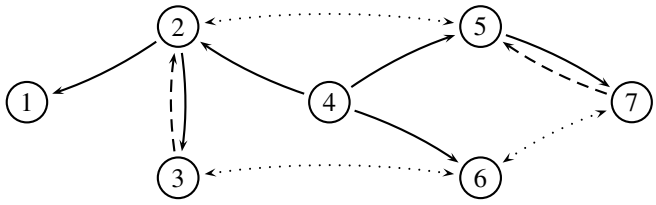

FIG. 3: Illustration de l'algorithme 1 (s=*4*) sur G

Fonction Dijkstra-transverse(G(N,A,c) : **graphe** s : **sommet**) : **Matrice de coût Mc**
  $Mc(k^+(s),|N|)$: **Matrice de coût pour chaque prochain saut sur s**
  $Tm(|N|,2)$: **Table des meilleurs sauts (coût C_1 et prochain saut p_s)**
  $Ta(|N|)$: **Table de balayage des noeuds**, $P(|N|)$: **Table père**
  $Tm(d,0) = \infty$, $Mc(v,d) = \infty \; \forall d \in N$, $\forall$ v successeur de s
  $Mc(v,s) = 0 \; \forall$ v successeur de s, $Tm(s,0) = 0$, $Tm(s,1) = s$
  Tant que ($\exists$ un noeud n $\in$ N non marqué) **faire**
    sélectionner x $\in$ N de plus petit coût Tm(x,0), placer x dans Ta
    Pour y, successeur direct de x **allant de** 1 **à** $k^+(x)$ **faire**
      Si ($Tm(x,0) + c(x,y) < Tm(y,0)$) **Alors**
        mettre à jour Tm (coût et p_s) pour la destination y, $P(y) = x$
        mettre à jour Mc pour la destination y à l'index $p_s = Tm(x,1)$
        [meilleur chemin]
      Sinon
        Si ($Mc(Tm(x,1),x) + c(x,y) < Mc(Tm(x,1),y)$) **Alors**
          mettre à jour Mc pour la destination y à l'index $p_s = Tm(x,1)$
          [chemin transverse Pt]
        Fin Si
      Fin Si
      marquer x
    Fin Pour
  Fait
  Pour i **allant de** |N| **à** 1 **faire**
    Pour v, successeur direct de s **allant de** 1 **à** $k^+(s)$ **faire**
      Si (Mc(v,Ta(i))+$c(Ta(i),P(Ta(i)))$ < Mc(v,P(Ta(i)))) **Alors**
        mettre à jour Mc pour la destination P(Ta(i)) à l'index $p_s = v$
        [chemin transverse retour Ptr]
      Fin Si
    Fin Pour
  Fin Pour
  Pour i **allant de** 1 **à** |N| **faire**
    Pour v, successeur direct de s **allant de** 1 **à** $k^+(s)$ **faire**
      Si (Mc(v,P(Ta(i)))+$c(P(Ta(i)),$Ta(i)) < Mc(v,Ta(i))) **Alors**
        mettre à jour Mc pour la destination Ta(i) à l'index $p_s = v$
        [chemin transverse avant Pta]
      Fin Si
    Fin Pour
  Fin Pour
  Retourner Mc
Fin

ALG. 1: Dijkstra-transverse

3.2 Phase de validation distribuée

Il faut à présent s'assurer que les circuits formés par la composition, entre routeurs adjacents deux à deux, des chemins générés par l'algorithme défini dans la section précédente, permettent néanmoins l'aiguillage (sortie de la boucle de routage) des données. En effet, il s'agit ici de transformer les chemins, calculés par un nœud, en routes empruntées par un paquet. Une route est une suite de premiers sauts validés de proche en proche selon l'interface entrante. La relation de validation pour le j^{eme}, au sens $C_j(v,d)$, chemin de v vers d, pour s en entrée, sera : $C_j(v,d) \leq C_1(s,d)_{\ (1 \leq j \leq k^+(v))}$. Cela implique que le j^{eme} chemin (ici l'expression d'un chemin est réduite à un premier saut) calculé par v pour la destination d soit valide pour

le nœud s en entrée. Ainsi, tous les chemins d'indice inférieur selon la métrique C sont a fortiori validés. Ces chemins sont activés dès lors que v valide ces prochains sauts pour son sous-réseau en entrée (les meilleurs premiers sauts de coût égaux sont validés d'office pour chaque destination). Par exemple, sur la figure 1, le chemin d'indice j, selon la métrique C, via z sur v sera validé pour s en entrée si son coût b est inférieur ou égal à $C_1(s,d)$ (meilleur coût calculé sur s pour d). L'échange de messages entre s et v est illustré sur la figure 1. Une *demande* envoyée de s vers v contient le meilleur coût calculé par s vers chaque destination d dont v est un prochain saut possible. Le routeur v envoie alors une *réponse* contenant autant de booléens que de destinations d qu'il a reçues dans le message *demande*. Chacun de ces booléens représente la validité d'au moins un chemin calculé par v pour chacune des destinations d. Le nombre de messages échangés sur l'ensemble d'un réseau modélisé par G est au pire 2|A|. Sur la figure 3, le nœud *2* peut valider trois chemins pour son sous réseau (via *v=4, 5 et 3* pour *In=2*) vers la destination *d=7*, alors que, pour des flux en provenance de *4*, seule la ligne via *v=5* (pour *In=4*) sera validée vers *d=7*. Ainsi, le circuit $\{4,2,5,4\}$ sera exploité par *4* grâce au nœud *5* (le plus proche de *7*) qui aiguillera les flux en provenance de *2* directement vers *7*, et le chemin $Pta_3^3(4,7)$ calculé par *4* passant par *2* et *5* sera validé de proche en proche pour former une route. La relation définie au paragraphe 2.1 en empêcherait l'utilisation avec MPATH, car $C_1(2,7) = C_1(4,7)$, et par conséquent le nœud *4* ne pourrait pas router ses flux via *2* pour la destination *7*.

4 Conclusion et perspectives

Dans cet article nous avons traité la première étape du multiroutage qui consiste à générer les multichemins le plus efficacement possible (faible coût processeur et temps supplémentaire de convergence négligeable). Notre algorithme est capable de déployer un sous-ensemble de multichemins, dont le premier arc est distinct par destination, plus important qu'avec les propositions existantes. Les résultats obtenus par simulation sur un réseau de 25 nœuds (Renater3) ont indiqué un gain significatif sur le flot maximum moyen (*FM*) et sur le nombre de chemins générés (*NC*). En effet, on observe sur cette topologie des gains de 13% (*FM*) et 35% (*NC*) par rapport à ECMP et de 12% (*FM*) et 25% (*NC*) par rapport à MPATH. Nous évaluons actuellement la possibilité d'approfondir la phase de validation distribuée de notre algorithme afin d'augmenter encore le nombre de multichemins validés sans accroître le temps de convergence et le nombre de messages envoyés. La suite de notre travail portera sur la mise en place d'un système de multiroutage réactif aux variations de charge, où chaque routeur capable de répartir la charge pour une destination assurera la distribution localement. L'un des problèmes majeurs sera de définir les proportions de trafic à distribuer sur les prochains sauts validés en tenant compte du contrôle de congestion TCP.

Références

[ABG+01] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. Rsvp-te : Extensions to rsvp for lsp tunnels. Request For Comments 3209, Internet Engineering Task Force, 2001.

[Dij59] E.W. Dijkstra. A note on two problems in connection with graphs. *In Numerische Mathematik, vol. 1*, pages 269–271, 1959.

[Epp94] D. Eppstein. Finding the k shortest paths. *In Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 154–165, 1994.

[KKDC05] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope : Responsive yet stable traffic engineering. In *SIGCOMM '05 : Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2005.

[Moy98] J. Moy. Ospf version 2. Request For Comments 2178, Internet Engineering Task Force, 1998.

[NZ01] S. Nelakuditi and Z.-L. Zhang. On selection of paths for multipath routing. In *Proceedings of IWQoS*, 2001.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching (mpls). Request For Comments 3031, Internet Engineering Task Force, January 2001.

[Vut01] S. Vutukury. *Multipath Routing Mechanisms for Traffic Engineering and Quality of Service in the Internet*. PhD thesis, University of California, Santa Cruz, March 2001.