Algorithme distribué de routage compact en temps optimal †

Cyril Gavoille[1], Christian Glacet[2], Nicolas Hanusse[3] et David Ilcinkas[3]

[1] *LaBRI - Université de Bordeaux ;* [2] *LaBRI & INRIA Bordeaux Sud-Ouest ;* [3] *LaBRI - CNRS*

Nous présentons un algorithme distribué construisant des tables de routage de taille sous-linéaire en n, le nombre de nœuds du réseau. Le temps de convergence est proportionnel au diamètre, ce qui est optimal. Par rapport à BGP, la complexité du nombre de messages échangés est améliorée jusqu'à un facteur $\sqrt{n}$, alors que la longueur des routes induites par les tables est allongée d'un facteur garanti constant. Notre algorithme est conçu pour un environnement statique synchrone ou asynchrone et produit un schéma *name-independent*.

Keywords: Routage compact, algorithme distribué asynchrone

1 Contexte et motivations

Bon nombre de protocoles de routage sont basés sur le calcul distribué de plus courts chemins. Par exemple, le protocole BGP, chargé de véhiculer des messages entre les systèmes autonomes d'Internet (AS) consiste en une implémentation distribuée de l'algorithme de Bellman-Ford. Une implémentation distribuée possible calcule pour tout sommet source, le prochain voisin à atteindre (*next-hop*) et sa distance à sa destination (protocole de type *distance-vector*). Le protocole BGP quant à lui, construit des tables de routage contenant, pour chaque destination, un plus court chemin vers le sommet destination (protocole de type *path-vector*). Encoder un chemin dans les tables induit des entrées de plus grande taille mais permet d'assurer une meilleure stabilité (détection locale de boucles) et convergence (problème du *count-to-infinity*) si le réseau est dynamique. Une des caractéristiques importante de BGP est que le routage à proprement dit se fait en fonction du nom original de la destination (on parle alors de schéma ou de routage *name-independent*). Aussi il converge en temps optimal $O(D)$ où D est le diamètre (nombre de *hops*) du réseau. Ce point est crucial pour la qualité de service car les périodes d'instabilités entrainent de nombreux échecs de routage, il est impératif de réduire autant que possible la durée de celles-ci.

Viser des plus courts chemins est cependant coûteux à plusieurs égards. Gavoille et Pérennès [GP96] ont montré que cela nécessite un espace mémoire d'au moins $\Omega(n)$ bits simultanément pour $\Omega(n)$ nœuds ‡ du réseau. Le nombre de messages échangés pour la construction et la maintenance des tables est nécessairement important dans le pire des cas ($\Omega(n^2)$). On peut prouver que la complexité en nombre de messages de BGP est $\Theta(mn^2)$ où m est le nombre de liens. Notons toutefois que BGP se révèle plus économe lors d'un scenario synchrone, puisque seulement $\Theta(mn)$ messages sont alors nécessaires (voir la table 1). Proposer des solutions qui se révèlent économiques aussi bien dans les scénarios synchrones qu'asynchrones est un enjeux important puisque dans un réseau réel, on peut facilement alterner entre périodes faiblement et fortement asynchrones.

†Supporté par le projet européen EULER – STREP 7 et le projet ANR DISPLEXITY.

‡. Et ce, même pour un réseau de degré borné et de diamètre $O(\log n)$, même si le schéma n'est pas *name-independent*.

Une première piste en direction de l'économie est celle du *routage compact* où l'objectif est de réduire l'information nécessaire au routage (i.e., la taille des tables de routage), et de réduire d'autant le nombre de messages nécessaire à son calcul. Dans [AGM+08], les auteurs montrent qu'il est possible de concevoir des tables de routage *name-independent* de taille [§] $\sqrt{n}$ avec un étirement [¶] garanti ≤ 3 pour tout graphe. Autoriser des détours maîtrisés permet donc de réduire la taille des tables de routage. D'autres algorithmes de routage compact *name-independent* tels que [TZLL12], dédiés aux graphes de type *power-law* (le graphe des AS a cette propriété), permettent d'obtenir de meilleurs étirements en pratique. Cependant, les solutions de [AGM+08] et [TZLL12] ne proposent pas de mécanisme de construction distribuée des tables de routage. Dans [SGF+10], il est présenté un algorithme distribué permettant aussi de calculer des tables de taille sous-linéaire. L'étirement est ≤ 7, mais l'algorithme proposé n'est pas silencieux (des messages continuent de circuler après qu'il ait convergé). De plus, contrairement au notre, il ne garantit pas qu'à tout moment de la construction la mémoire nécessaire au calcul des tables reste sous-linéaire. Enfin, la présentation non détaillée de cet algorithme ne permet pas d'analyser les complexités en nombre de messages et temps de convergence.

Notre contribution. Nous proposons ici un algorithme distribuée asynchrone de construction de tables de routage compact. L'étirement est ≤ 7 et la taille des tables générées ainsi que l'espace mémoire des nœuds tout au long de l'algorithme est en $O(\sqrt{n})$. L'algorithme est silencieux et termine en temps optimal $O(D)$. Le schéma de routage est *name-independent*. Les complexités en nombre de messages sont détaillées dans la table 1. Le détail des complexités pour le nombre d'entrées échangées en synchrone (M_s) et en asynchrone (M_a) se trouvent dans chacune des sous-sections de la section 3. On remarquera que ces complexités améliorent celles de BGP asymptotiquement d'un facteur $\sqrt{n}$ dans les graphes *petit monde*, que cela soit dans le cas synchrone ou asynchrone. Elles améliorent également celles de l'algorithme présenté dans [GL12] (bien que présenté synchrone et de complexité $O(mn)$, cet algorithme peut être α-synchronisé [Pel87] en gardant une complexité de $O(mn)$ messages). Il est important de noter que notre algorithme fonctionne dans un graphe valué, mais les analyses de complexités sont proposées pour des graphes non valués.

Algorithme	**Mémoire** (♯**entrées**)	**Nombre d'entrées échangées**		**Taille des entrées (bits)**		**Etirement**
		Synchrone	**Async.**	**Synchrone**	**Async.**	
Path-vector	$\Theta(n)$	$\Theta(mn)$	$\Theta(mn^2)$	$O(D\log n)$	$O(n\log n)$	1
ce papier	$O(\sqrt{n}\log n)$	$O(m\sqrt{n}+n^{3/2}\min\{D,\sqrt{n}\log n\})$	$O(mn^{3/2})$	$O(D\log n)$ [‖]		7

TABLE 1: Comparaison de BGP et de notre algorithme distribué avec $k=\sqrt{n}$.

2 Description de l'algorithme de routage

Notations. Tout sommet u a une couleur $c(u) \in [1,k]$ choisie de manière aléatoire uniforme. Tout sommet de couleur 1 est appelé *un landmark*. L'ensemble des landmarks est noté L. De plus tout sommet possède une fonction de hachage équilibrée h associant à tout sommet u une couleur $h(u) \in [1,k]$. Pour chaque sommet u, on note B_u un ensemble de nœuds *proches* de u tel que B_u est *complète* (contenant un sommet de chaque couleur) et l_u le plus proche landmark de u.

Routage et tables de routage. Pour une requête de routage de u vers v, plusieurs cas sont possibles :

- $v \in B_u \Rightarrow u$ stocke dans sa table B_u la distance et le *next-hop* w vers v. Le sommet u route vers w.
- $v \in L \Rightarrow u$ stocke dans sa table L_u, pour tout landmark l, son père $pere_u[l]$ dans un arbre de plus court chemin enraciné en l. Le sommet u route vers $pere_u[l]$.

§. À un facteur polylogarithmique près.

¶. Ratio entre la longueur de la route utilisée et la distance.

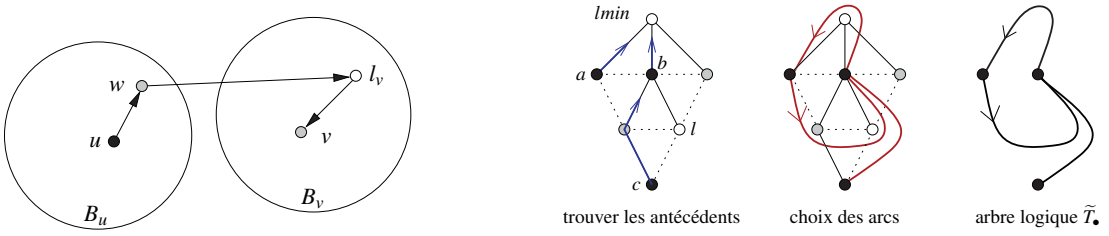
‖. En modifiant légèrement l'algorithme présenté en sous-section 3.2, il est possible de réduire la taille des messages échangés (réduire la taille des entrées). L'algorithme serait une version distribuée de [FG01], les entrées auraient une taille de $O(\log^2 n)$.

– $v \notin B_u \cup L$ et v est tel que $h(v) = c(u) \Rightarrow u$ stocke dans sa table *couleur* C_u un chemin allant de l_v à v. Pour router vers v, le sommet u enverra ce chemin à l_v qui l'utilisera pour router vers v.

Pour router vers un autre sommet v, le sommet u envoie la requête à $w \in B_u$ tel que $c(w) = h(v)$ (voir figure 1(a)). Pour le routage vers les nœuds de B_u il est nécessaire que la boule B_u soit *monotone* : si u connait le *next-hop* w pour une destination $v \in B_u$, alors v doit être dans B_w.

3 Résumé de l'algorithme distribué

L'objectif de cet algorithme est de construire $\forall u \in V$ les tables B_u, L_u et C_u nécessaires au routage. Il se découpe en cinq parties : (1) Construction de la boule de voisinage B_u. (2) Construction $\forall l \in L$ de l'arbre de landmark T_l et élection d'un landmark $lmin \in L$. (3) Détection de la terminaison des arbres de landmark en utilisant T_{lmin}. (4) Construction pour toute couleur $c \in [1,k]$ d'un arbre logique $\widetilde{T}_c$ entre les nœuds de couleur c. (5) Diffusion d'informations de routage dans $\widetilde{T}_c$. Les algorithmes (2),(3),(4) et (5) sont effectuées en série, l'algorithme (1) tourne en parallèle. Dès que ces cinq algorithmes ont convergé, les tables de routage calculées garantissent un étirement d'au plus 7 avec une mémoire $O(\sqrt{n}\log n)$ pour $k = \sqrt{n}$ [GL12].



(a) Routage de u à v via w tel que $c(w) = h(v)$

(b) Construction de $\widetilde{T}_\bullet$ (avec $l_a = lmin$ et $l_b = l_c = l$)

FIGURE 1:

3.1 Création des boules de voisinage. L'algorithme utilisé est une adaptation du protocole *distance-vector* à la différence qu'un sommet u ne garde en mémoire sa distance à v que s'il ne peut pas enlever v de sa boule B_u sans altérer les propriétés nécessaires au routage (la boule doit rester monotone et complète). Les sommets sont supprimés par ordre inverse d'ajouts en cas d'égalités.

Complexités. En synchrone, tout sommet u change au pire $|B_u|$ fois d'état.Un sommet est ajouté définitivement à B_u (la première distance apprise est la distance dans G). En asynchrone, u connait à tout instant $|B_u|$ nœuds. Dès que u change d'état il réduit sa distance à un sommet de B_u. Le sommet u changera $O(|B_u|^2)$ fois d'état, entrainant $degre(u)$ envois de messages. La taille d'une boule de voisinage est, avec grande probabilité $O(k\log k)$. Donc $M_s(\text{BOULES}) = O(mk\log k)$ et $M_a(\text{BOULES}) = O(m(k\log k)^2)$.

3.2 Construction des arbres de landmark et élection de *lmin*. Tout landmark l initie la construction d'un arbre de plus court chemin (BFS distribué), tout sommet stockera son père dans T_l ainsi que sa distance actuelle à l. On ajoute à cet algorithme une détection de terminaison utilisant un mécanisme de synchronisation locale (avec ses voisins) qui ne génère aucun surcout significatif. De cette façon lorsque l est synchronisé avec tous ses voisins il est sûr que l'arbre T_l est un arbre de plus court chemin couvrant G.

En ajoutant aux messages de synchronisation l'identifiant du plus petit landmark du sous-arbre courant, tous les landmark une fois leurs arbres terminés connaitrons le landmark d'identifiant minimum $lmin$. Dès que $lmin$ apprend qu'il est le plus petit landmark il diffuse son identifiant dans G.

Complexités. La complexité est celle de n/k BFS soit $M_s(\text{LMIN}) = O(mn/k)$ et $M_a(\text{LMIN}) = O(mn^2/k)$

3.3 Détection de la terminaison de tous les arbres de landmark par *lmin*. À la manière de l'étape précédente, on va utiliser la diffusion de terminaison (diffusion de $lmin$ dans T_{lmin}) combinée avec des

synchronisations pour que *lmin* apprenne la liste de tous les landmarks L. Lorsqu'un sommet se synchronise avec un voisin il lui envoie la liste des landmarks présents dans son sous-arbre. Quand *lmin* s'est synchronisé avec ses voisins pour la terminaison de l'arbre T_{lmin} il connait la liste L de tous les landmarks. Parallèlement, dès qu'un landmark apprend la terminaison de son arbre (sous-section 3.2), il prévient *lmin*. Ainsi *lmin* est capable de détecter la terminaison de tous les arbres de landmarks et diffuser cette information.

Complexités. Le sommet *lmin* diffuse son identifiant dans T_{lmin} et chaque landmark fait remonter son identifiant dans T_{lmin} on a donc $M_s(\text{TERMINAISON}) = M_a(\text{TERMINAISON}) = O(m+n^2/k)$

3.4 Construction des arbres logiques. Pour toute couleur $c \in [1,k]$ on construit un arbre logique $\widetilde{T}_c$ dont les arêtes sont bi-dirigées (voir figure 1(b)). Un arc (u,v) dans $\widetilde{T}_c$ est un chemin de routage vers v via le landmark l_v. Pour permettre la construction de $\widetilde{T}_c$, le sommet *lmin* doit stocker l'identifiant d'un sommet $racine_c$ ainsi que l'arc $(lmin, racine_c)$. Cette construction est initiée par tout sommet u tel que $c(u) = c$ dès qu'il apprend la terminaison de construction des arbres de landmark. Le sommet u envoie à son père dans T_{lmin} un message $m_u = (arc(l_u,u))$. Pour tout message $m_u = (arc(l_u,u))$ reçu par un nœud v :

(1) **si** $v = lmin$
- si $racine_c = \emptyset$, alors $racine_c = u$ et $arc_c = arc(l_u,u)$
- sinon *lmin* envoie les message m_{racine_c} à u par $arc(l_u,u)$ et m_u à $racine_c$ par arc_c.

(2) **sinon**
- si $c(v) \neq c(u)$ le nœud v envoie m_u à son père dans T_{lmin}.
- si $c(v) = c(u)$ le nœud v ajoute l'arc (v,u).

Complexités. Avec grande probabilité un nœud u est à distance $\leq k \log k$ de *lmin* ou de son premier antécédent de couleur $c(u)$ dant T_{lmin}. De plus, tous les arbres sont des arbres de plus court chemin dont le diamètre est $\leq 2D$. La longueur maximale d'un arc (u,v) est au plus 7 fois la distance entre u et v dans T_{lmin}. La longueur maximale d'un arc (u,v) est alors, avec grande probabilité, $O(\min(2D, k\log k))$. Pour tout arc (u,v) ajouté à un arbre logique $\widetilde{T}_{c(u)}$, les nœuds u et v s'échangent un message via celui-ci. Le nombre total d'arcs crées est de n. On a donc $M_s(\text{GRAPHELOGIQUE}) = M_a(\text{GRAPHELOGIQUE}) = O(n\min(D, k\log k))$.

3.5 Diffusion des informations de routage via Landmark. Tout nœud v doit diffuser son landmark l_v ainsi que le chemin de l_v vers v à tous les nœuds $\{u \in V | c(u) = h(v)\}$. Dès que v apprend la terminaison des arbres de landmarks il envoie $(l_v, arc(l_v,v))$ au plus proche nœud u tel que $c(u) = h(v)$. Le nœud u se chargera de diffuser cette information de routage dans $\widetilde{T}_{c(u)}$. Enfin, lorsqu'un nœud u ajoute un arc (u,w) dans $\widetilde{T}_{c(u)}$ il envoie sa table C_u à w par l'arc (u,w).

Complexités. Chacun des graphes logiques contient $O(n/k)$ arcs de longueur $\min(2D, k\log k)$. Une entrée de routage pour un nœud v peut être propagée au plus une fois sur chaque arc de $\widetilde{T}_{h(v)}$. On a donc $M_s(\text{DIFFUSIONS}) = M_a(\text{DIFFUSIONS}) = O(n^2/k \min(D, k\log k))$.

RÉFÉRENCES

[AGM+08] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. *ACM Transactions on Algorithms (TALG)*, 4(3) :37, 2008.

[FG01] P. Fraigniaud and C. Gavoille. Routing in trees. *Automata, Languages and Programming*, pages 757–772, 2001.

[GL12] C. Glacet and V. Lucas. Vers un routage compact distribué. *Algotel*, 2012.

[GP96] C. Gavoille and S. Pérennès. Memory requirement for routing in distributed networks. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 125–133. ACM, 1996.

[Pel87] D. Peleg. *Distributed computing : a locality-sensitive approach*, volume 5. Society for Industrial Mathematics, 1987.

[SGF+10] A. Singla, P. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names. In *Proceedings of the 6th International Conference*, page 20. ACM, 2010.

[TZLL12] M. Tang, G. Zhang, T. Lin, and J. Liu. Hdlbr : a name-independent compact routing scheme for power-law networks. *Computer Communications*, 2012.