# *Construction auto-stabilisante de forets de diffusion optimale*

Bertrand Ducourthial et Sebastien Tixeuil

*Laboratoire HEUDIASYC, UMR CNRS 6599, Universite de Technologie de Compiegne, 60205 Compiegne CEDEX*
*email:* Bertrand.Ducourthial@utc.fr
*Laboratoire de Recherche en Informatique, UMR CNRS 8623, Universite Paris-Sud, 91405 Orsay CEDEX*
*email:* tixeuil@lri.fr

Dans cet article, nous nous intéressons à la détermination des arborescences permettant les meilleures diffusions, dans le cas où le critère d'évaluation peut être exprimé localement au niveau de chaque lien, sous la forme d'un taux. Nous proposons un algorithme de construction d'arborescence auto-stabilisant, c'est-à-dire apte à resister aux défaillances temporaires des sites ou des liens de communication.

**Keywords:** forêt de diffusion, algorithme adaptatif, auto-stabilisation

## 1 Introduction

**Diffusion multi-source** Nous nous plaçons dans le cadre d'un réseau de communication dans lequel une ou plusieurs sources détiennent la même information, qui est désirée par les autres sites. Cette hypothèse de travail correspond par exemple à la mise à jour de sites miroirs de serveurs *ftp*, *news* ou *html* dans le réseau. Elle correspond aussi à l'écoute d'une émission radio via un réseau cablé, dans lequel les sources sont les machines équipées d'une carte de réception radiophonique.

Nous souhaitons déterminer, pour chaque site, la meilleure source et le meilleur chemin depuis cette source jusqu'au site. Il s'agit donc de construire une forêt couvrante du réseau (*ie.* un ensemble d'arborescences tel que chaque site est dans une et une seule arborescence), telle que chaque site soit rattaché par le meilleur chemin à une meilleure source. Par cette hypothèse, notre travail s'apparente au *broadcast* de grande quantité d'information plutôt qu'au *multicast* [3]. Néanmoins, son principe pourrait être adapté au cas plus général du *multicast*, dans lequel certains sites ne souhaitent pas l'information (l'arborescence n'est donc plus forcément couvrante, et peut contenir des nœuds intermédiaires n'utilisant pas l'information qu'ils retransmettent).

**Taux par lien** La distance n'est pas toujours le critère adapté dans le choix de la meilleure source. L'algorithme de [7] autorise différents critères de choix (y compris la distance ou le débit). Dans cet article, nous l'adaptons afin de développer une approche générale pour les critères exprimables sous la forme d'un taux par lien. Cela peut correspondre, par exemple, au taux de fiablité du lien, au taux de perte de paquets sur le lien, *etc*.

Pour ne pas obtenir des taux trop fluctuants, il est possible d'utiliser des techniques telles que les moyennes mobiles, calculées pendant une période de temps à déterminer. Par la suite, nous supposons qu'à chaque lien $(u,v)$ du réseau est associé un taux $\tau_{u,v}$, compris entre 0 et 1, tel que plus le taux approche 1, plus le lien est performant.

**Algorithme réparti** Connaissant les différents taux des liens, le taux global d'un chemin peut alors être calculé. Pour un site donné, ce taux global permet de déterminer la meilleure source et le meilleur chemin depuis cette source. Chaque site étant concerné, une approche centralisée entraînerait un fort encombrement du réseau, d'autant que les taux associés aux liens — contrairement au débit maximal des liens —

évoluent continuellement. Nous proposons donc un algorithme réparti pour construire les arborescences des meilleures diffusions.

**Tolérance aux défaillances** La fiabilité des composants de base (processeurs et liens) n'étant pas parfaite, plusieurs types de défaillances peuvent être répertoriés, suivant leur localisation (processeur ou lien) et leur nature. En particulier, des corruptions de mémoire sur les processeurs peuvent survenir suite à une panne immédiatement suivie d'une réparation (voir [10]), ou encore lors de défaillances temporaires de l'alimentation électrique. Pour un lien de communication, des débranchements et branchements successifs, ainsi que des parasites électromagnétiques, peuvent conduire à des problèmes similaires.

Dans un cadre réparti, où les processeurs n'ont qu'une vision locale du système, il est souvent impossible de déterminer qu'une telle défaillance a eu lieu. L'auto-stabilisation [2] est une technique qui permet à un système réparti d'exhiber un comportement correct au bout d'un temps fini, même en partant d'un état initial corrompu et incorrect. Notre solution au problème des forêts couvrantes de diffusion est auto-stabilisante, ce qui lui permet de tolérer les corruptions transitoires des mémoires, ainsi que de s'adapter sans intervention extérieure à des variations des taux des liens.

# 2 Modèle

Dans cette section, nous décrivons brièvement le modèle de [7], utilisé dans la suite de l'article.

**Réseau** Nous modélisons le réseau par un graphe orienté $G(V,E)$. Nous notons $\delta_v^-$ le nombre d'arcs entrants du nœud $v$. Un tel nœud est pourvu d'une mémoire $M_v$ de $\delta_v^- + 2$ cases (figure 1). $M_v[0]$ est une valeur qui ne peut être altérée (*eg.* stockée en ROM), et qui indique si le nœud est une source ou non (un bit pourrait suffire). Les cases $M_v[1]$ à $M_v[\delta_v^-]$ sont utilisées pour stocker les valeurs lues sur les $\delta_v^-$ antécédants dans le réseau. Le nœud $v$ utilise ces $\delta_v^-$ valeurs pour calculer un résultat, qu'il stocke dans $M_v[\delta_v^- + 1]$. Ce résultat pourra être lu par ses successeurs, par la primitive `read` $(M_v[\delta_v^- + 1])$.

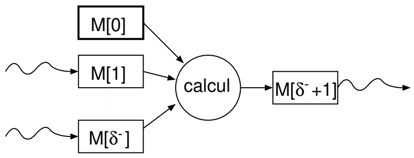


**Fig. 1:** Modélisation d'un nœud du réseau.

**Algorithme paramétré** Nous écrivons l'algorithme réparti sous la forme de règles. Les règles de copies permettent de lire un résultat sur un antécédant dans le réseau. La règle de calcul permet d'exploiter ces résultats ainsi que la valeur locale du nœud ($M[0]$) pour construire un nouveau résultat. Cette règle est paramétrée par un opérateur $n$-aire, noté ici $f$.

**Algorithme 1** Algorithme paramétré par $f$, nœud $v$

<u>Copie :</u>

$\triangleright$ *Pour chaque antécédant u de v, tel que l'arc $(u,v)$ soit le $i^{ème}$ arc entrant de v.*

$M_v[i] \leftarrow$ `read` $(M_u[\delta_u^- + 1])$

<u>Calcul :</u>

$M_v[\delta_v^- + 1] \leftarrow f\left(M_v[0], M_v[1], \ldots, M_v[\delta_v^-]\right)$

Notons que ces règles satisfont une contrainte d'atomicité faible : à chaque exécution de règle, au plus un registre de communication est utilisé. Ceci garantit une implantation facile dans un systéme où les processeurs communiquent au moyen de messages.

A des fins de simplification, les règles données ici sont toujours activables. Toutefois, une technique présentée dans [1] permet le bon fonctionnement de l'algorithme malgré un ordonnancement non équitable.

**Auto-stabilisation** La notion d'auto-stabilisation a été définie par Dijkstra en 1974 [2, 11], comme une propriété d'un système réparti à retrouver sans intervention extérieure un comportement correct au bout d'un temps fini et indépendament de l'état global du système à l'origine.

L'auto-stabilisation est particulièrement utile lors de calculs globaux distribués (voir [4]), tels la construction de forêts couvrantes, puisqu'un algorithme vérifiant cette propriété est naturellement adaptatif aux entrées réparties sur les processeurs du système.

Dans le cadre particulier de la construction d'arbres et de forêts vérifiant des critères particuliers, on peut citer les travaux de Dolev et Herman [5] (pour les arbres de plus courts chemins) et de Schneider et Gouda [9] (pour les arbres de flot maximal).

**Opérateurs** Les résultats de G. Tel sur les infimums [12] assurent que l'algorithme 1 se stabilisent lorsque l'opérateur $f$ est binaire, associatif, commutatif et idempotent ($s$-opérateur). Cela signifie, par exemple, que si chaque nœud calcule le minimum des valeurs de ses antécédants, alors au bout d'un temps fini plus aucune nouvelle valeur ne sera produite. Dans [6], nous avons introduit des opérateurs plus généraux ($r$-opérateurs), et nous avons montré que ces $r$-opérateurs conduisaient à la stabilité du réseau.

Dans [7], nous avons montré (i) que les $s$-opérateurs ne pouvaient conduire à un algorithme auto-stabilisant sur tout type de topologie, et (ii) que les $r$-opérateurs strictement idempotents fournissent des algorithmes autostabilisants quels que soit la topologie du réseau. Nous résumons ci-dessous les conditions suffisantes portant sur le paramètre $f$ de l'algorithme, de telle sorte que ce dernier soit auto-stabilisant sur tout type de topologie, malgré une atomicité faible.

1. Pour un entier $n > 1$ donné, $f\ :\mathbb{S}^n \mapsto \mathbb{S}$ est défini par : $f(x_0,\dots,x_n) = x_0 \oplus r_1(x_1) \oplus \cdots \oplus r_n(x_n)$ ;
2. l'ensemble de définition $\mathbb{S}$ est fini, ou toute suite strictement croissante est non bornée ;
3. $\oplus$ est un $s$-opérateur qui induit une relation d'ordre total $\preceq_\oplus$ sur $\mathbb{S}$ (*i.e.* $x \preceq_\oplus y \equiv x \oplus y = x$) ; on note $e_\oplus$ le plus grand élément de $\mathbb{S}$ ;
4. les fonctions $r_i$ sont des homomorphismes de $(\mathbb{S},\oplus)$ ($r_i(x \oplus y) = r_i(x) \oplus r_i(y)$) ;
5. pour toute fonction $r_i$ et pour tout $x \in \mathbb{S} \setminus \{e_\oplus\}$, $x \prec_\oplus r_i(x)$ (*i.e.* $x \preceq_\oplus r_i(x)$ et $x \neq r_i(x)$).

Les conditions 1, 3, 4 et 5 définissent un $r$-opérateur strictement idempotent. La condition 2 et l'hypothèse de l'ordre *total*, ne sont imposées que par le choix d'une atomicité faible. Une atomicité composité permet de s'en affranchir [8]. On vérifie ainsi que l'opérateur $(x,y) \mapsto \min(x, y+1)$ fournit un algorithme auto-stabilisant ; il permet de calculer la distance à la source la plus proche [7].

**Configuration terminale** Lorsque les conditions ci-dessus sont remplies, l'algorithme 1 termine en temps fini quelque soit les valeurs initiales des mémoires et des pannes temporaires rencontrées. Le résultat sur le nœud $v$ est donné par :

$$\bigoplus \{r_{P_{u\to v}}(M_v[0]), u \text{ antécédant de } v \text{ dans le réseau }\} \tag{1}$$

où $r_{P_{u\to v}}$ est la composition des fonctions $r_i$ rencontrées sur le chemin $P_{u\to v}$ reliant le nœud $u$ au nœud $v$. Cela correspond au minimum (au sens de $\preceq_\oplus$) des valeurs en ROM de chaque antécédant $u$, transformées par les fonctions $r_i$ le long des chemins reliant l'antécédant $u$ au nœud $v$.

L'évaluation du temps requis pour la convergence nécessite d'ajouter une hypothèse de synchronisation du réseau. Dans le cas de l'application que nous développons ci-après, $\mathbb{S}$ est fini, et contient un nombre d'éléments lié à la précision de l'échantillonage des taux. Sous cette hypothèse, et en supposant le réseau synchrone, le temps de convergence est au pire de l'ordre de $O(D + |\mathbb{S}|)$, où $D$ est la plus grande distance séparant une source d'un autre site dans le réseau, et où $|\mathbb{S}|$ désigne le nombre d'éléments de $\mathbb{S}$.

# 3 Forêt couvrante de diffusion

Afin d'obtenir un algorithme auto-stabilisant construisant la forêt couvrante de diffusion optimale relativement aux critères des taux, il nous faut trouver un opérateur $f$ qui traduise le choix du meilleur chemin, et qui vérifie les conditions du § 2.

**Taux** Soit $k$ le nombre de bits d'une case mémoire d'un nœud. On échantillonne les taux des liens sur $2^k - 1$ valeurs, depuis 0 (ligne hors-service) jusqu'à $2^k - 1$ (ligne parfaite). Nous supposons en outre qu'aucune ligne n'est parfaite dans le réseau (cette hypothèse peut être intégrée à la définition des taux) ; aucun taux n'est donc égal à $2^k-1$. Le taux d'un chemin $P$ sera obtenu en effectuant le produit des taux des arcs le composant.

**Opérateur** D'après ce qui précède, pour connaître le chemin le plus fiable depuis une source, chaque nœud doit effectuer des multiplications et des calculs de maximum pour distinguer les arcs incidents dont le taux est le plus élevé. Construisons maintenant un opérateur qui effectue ce calcul et qui, lorsqu'il est utilisé en paramètre de l'algorithme 1, doit permettre de résoudre le problème de manière auto-stabilisante.

Soit $\tau_v^i$ le taux du $i^{\text{ème}}$ arc incident du nœud $v$. Nous définissons l'opérateur $\mathrm{maxmul}_\tau$ par :

$$\mathrm{maxmul}_\tau(x_0,\ldots,x_{n-1}) = \max\left(x_0, \lfloor x_1\times\tau_v^1/(2^k-1)\rfloor,\ldots,\lfloor x_{n-1}\times\tau_v^{n-1}/(2^k-1)\rfloor\right)$$

**Auto-stabilisation** Pour que l'algorithme 1 soit auto-stabilisant lorsqu'il est instancié par l'opérateur $\mathrm{maxmul}_\tau$, il nous faut montrer que cet opérateur est un $r$-opérateur strictement idempotent sur l'ensemble $\mathbb{S}=\{1,\ldots,2^k-1\}$.

L'ensemble $\mathbb{S}$ est fini ; la condition n°2 est donc remplie. L'opérateur max est un $s$-opérateur sur $\mathbb{S}$ qui définit une relation d'ordre *total* $\preceq_{\max}$ qui est, en fait, la relation d'ordre usuelle $\geq$ sur les entiers (condition n°3). Pour chaque taux $\tau_v^i$, la fonction $\mathrm{r}_v^i$ définie par $\mathrm{r}_v^i(x)\mapsto\lfloor x\times\tau_v^i/(2^k-1)\rfloor$ est un homomorphisme de $(\mathbb{S},\max)$ (condition n°4). En effet,

$$\begin{array}{rcl}
\mathrm{r}_v^i(\max(x,y)) & = & \lfloor\max(x,y)\times\tau_v^i/(2^k-1)\rfloor \\
& = & \lfloor\max(x\times\tau_v^i/(2^k-1), y\times\tau_v^i/(2^k-1))\rfloor \\
& = & \max(\lfloor x\times\tau_v^i/(2^k-1)\rfloor, \lfloor y\times\tau_v^i/(2^k-1)\rfloor) \\
& = & \max(\mathrm{r}_v^i(x),\mathrm{r}_v^i(y))
\end{array}$$

Puisqu'aucun lien n'est parfait, nous avons $\tau_v^i\in\mathbb{S}\setminus\{2^k-1\}$, d'où $x\prec_{\max}\mathrm{r}_v^i(x)$ (*i.e.* $x>\mathrm{r}_v^i(x)$) pour chacune des $r$-fonctions $\mathrm{r}_v^i$ (condition n°5).

Les 5 conditions du § 2 sont donc vérifiées. L'opérateur $\mathrm{maxmul}_\tau$ est donc un $r$-opérateur $n$-aire strictement idempotent et l'algorithme 1 est auto-stabilisant lorsqu'il est instancié par cet opérateur.

**Arborescence** Il nous reste à montrer que l'algorithme permet effectivement de construire la forêt couvrante des meilleurs chemins relativement au critère des taux des liens.

Supposons que pour chaque source $u$, la donnée stockée en ROM ait pour valeur $M_u[0]=2^k-1$, tandis que pour tous les autres nœuds $v$, $M_v[0]=0$. Une fois le système stabilisé, chaque nœud $v$ obtient le résultat suivant (*cf.* équation (1)) :

$$M[\delta_v^-+1] = \max\{\mathrm{r}_{P_{u\to v}}(M_u[0]), u \text{ antécédant de } v \text{ dans le réseau }, P_{u\to v} \text{ un chemin de } u \text{ à } v\}$$

ce qui, eu égard aux valeurs en ROM, est équivalent à $M_v[\delta_v^-+1]=\mathrm{r}_{P_{u\to v}}(M_u[0])$, où $\mathrm{r}_{P_{u\to v}}(M_u[0])$ est le chemin de plus grand taux depuis une source antécédante $u$ jusqu'au nœud $v$. Par conséquent, le $r$-opérateur $\mathrm{maxmul}_\tau$ calcule le taux maximal des chemins depuis les sources antécédantes de chaque nœud.

Pour obtenir une forêt, il suffit ensuite à chaque nœud de choisir un meilleur antécédant, c'est-à-dire de sélectionner parmis ses arcs entrants, celui tel que la case mémoire $M_v[i]$ associée contienne la plus grande valeur.

La figure 2-gauche montre une configuration incorrecte due à des défaillances transitoires, tandis que la figure 2-droite montre une configuration correcte obtenue après stabilisation de l'algorithme.

# 4 Conclusion

Dans cet article, nous avons proposé un algorithme réparti construisant une forêt de diffusion optimale eu égard à un critère exprimé sous la forme d'un taux par lien. Cet algorithme est auto-stabilisant. Il peut donc supporter tout type de corruption des mémoires volatiles, causées par exemple après le redémarrage d'un site tombé en panne. En outre, il ne nécessite pas d'initialisation, puisqu'il donne une solution correcte en un temps fini quelque soit la configuration de départ. Enfin, il est *adaptatif aux entrées* : lorsque les taux

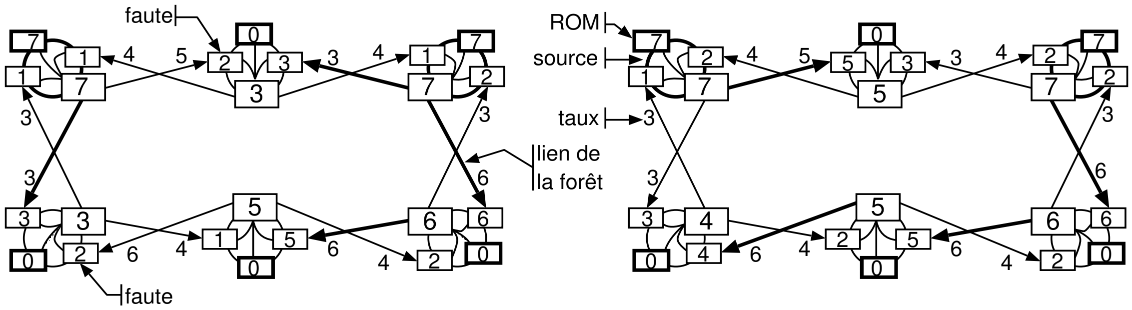


**Fig. 2:** Exemple d'exécution avec des registres de 3 bits et deux sources (cercles gras). A gauche, mauvaise forêt de diffusion suite à des défaillances. A droite, forêt correcte, obtenue automatiquement après un temps fini.

des liens sont modifiés pendant l'exécution répartie de l'algorithme, la forêt est automatiquement mise à jour.

Pour obtenir ce résultat, nous avons défini un nouvel opérateur, maxmul, afin d'instancier l'algorithme paramétré de [7]. Nous avons montré que cet opérateur vérifie les conditions assurant l'auto-stabilisation de l'algorithme. Les hypothèses sur lesquelles reposent notre travail correspondent au cas des diffusions multi-sources dans un réseau de communication. Une étude complémentaire devrait permettre d'adapter notre algorithme au cas des multicasts multi-sources.

# References

[1] S. K. Das, A. K. Datta, and S. Tixeuil. *Self-Stabilizing Algorithms in DAG Structured Networks*. 1999 International Symposium on Parallel Architectures, Algorithms

[2] E. W. Dijkstra. *Self-Stabilizing Systems in Spite of Distributed Control*. Communications of the ACM 17,11 pp. 643-644, 1974.

[3] C. Diot, W. Dabbous et J. Crowcroft. *Multipoint communication: a survey of protocols, functions, and mecanisms*. IEEE Journal on Selected Areas in Communications, 15, pp. 277-290, 1997.

[4] S. Dolev, M.G. Gouda and M. Schneider, Memory requirements for silent stabilization, PODC'96, pp. 27-34, 1996.

[5] S. Dolev and T. Herman, Superstabilizing protocols for dynamic distributed systems, Chicago Journal of Theoretical Computer Science, décembre 1997.

[6] B. Ducourthial. *New operators for computing with associative nets*. Proceedings of the Fifth International Colloquium on Structural Information and Communication Complexity (SIROCCO'98), Amalfi, Italie, pp. 55-65, Carleton Scientific, Juin 1998

[7] B. Ducourthial et Sébastien Tixeuil. *Self-stabilizing Global Computations with r-operators*. Proc. of the 2$^{nd}$ International Conference On Principles Of DIStributed Computing (OPODIS'98), pp. 99–113, Hermes, 1998.

[8] B. Ducourthial et Sébastien Tixeuil. *Self-stabilization with Path Algebra*. Theoretical Computer Science, Special issue on Max-Plus Algebra, à paraître.

[9] M.-G. Gouda et M. Schneider, Maximum flow routing, Second Workshop on Self-stabilizing Systems, pp. 2.1-2.13, 1995.

[10] M. Jayaram and G. Varghese, Crash failures and drive protocols to arbitrary states, PODC'96, pp. 247-256, 1996.

[11] M. Schneider. *Self-stabilization*. ACM Computing Surveys, vol. 25, pp. 45-67, 1993.

[12] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK, 1994.