Stabilité et Auto-stabilisation de BGP[*]

Sylvie Delaët[(1)] Duy-So Nguyen[(2)] Sébastien Tixeuil[(1,3)]

[(1)] *LRI – CNRS UMR 8623, Université Paris-Sud XI, 91405 Orsay cedex, France.*
[(2)] *LICA, Université de Reims Champagne-Ardennes, 51687 Reims Cedex 2, France.*
[(3)] *INRIA Futurs – Equipe Grand Large.*

BGP (Border Gateway Protocol) est le protocole standard de routage inter-domaine dans Internet. Le routage inter-domaine permet le routage entre systèmes autonomes (tels que des Universités ou des entreprises). BGP est un protocole à vecteurs de chemins enrichi par des politiques qui permettent d'altérer les informations de routage transmises aux autres systèmes autonomes.
Le problème de la stabilité du routage inter-domaine est de savoir si, en partant d'une configuration cohérente bien connue, le protocole converge vers un chemin stable vers chaque destination. Il est bien connu que ce problème est NP-complet ou NP-difficile suivant les hypothèses retenues. Plusieurs approches ont été définies pour proposer des conditions suffisantes à la stabilité d'un système.
Le problème de l'auto-stabilisation du routage inter-domaine est de savoir si, en partant d'une configuration arbitraire, le protocole converge vers un chemin stable vers chaque destination. Cet article traite des relations entre les deux problèmes.

Mots-clefs : Routage inter-domaine, auto-stabilisation, stabilité, BGP, Internet.

1 Introduction

Internet est constitué d'un ensemble de réseaux interconnectés les uns avec les autres. Chaque réseau administré par une même entité est appelé un *système autonome* ou *domaine*. Une université ou une grande entreprise sont deux exemples de systèmes autonomes. Le routage *intra-domaine* permet à deux ordinateurs d'un même système autonome de communiquer entre eux. Le routage *inter-domaine* consiste à rendre possible les communications entre des ordinateurs situés dans des systèmes autonomes différents. Un protocole de routage inter-domaine s'appuie en général sur un protocole de routage intra-domaine pour gérer les aspects du routage qui sont locaux à un système autonome.

1.1 BGP

BGP [13,14] est le protocole standard de routage inter-domaine dans Internet. Dans le principe, il s'agit d'un protocole de *vecteurs de chemins*. Pour chaque destination possible (c'est-à-dire pour chaque système autonome), chaque routeur stocke le chemin (c'est-à-dire la liste des système autonomes traversés) pour atteindre cette destination. Ce mécanisme permet d'éviter les boucles qui peuvent se produire quand on utilise un routage de type *vecteur de distance*, où seule la distance à la destination et le prochain système autonome sur le réseau seraient stockés sur chaque routeur.
A ce principe, BGP permet d'ajouter des contraintes, également appelées *politiques de routage*. Par exemple, une entreprise peut ne pas souhaiter que les données qu'elle envoie à destination de l'un de ses clients transitent par l'intermédiaire du réseau d'une entreprise concurrente, et préférer emprunter un chemin plus coûteux mais assurant la confidentialité des informations. De la même manière, une Université peut souhaiter utiliser de préférence une liaison lente mais à faible coût plutôt qu'une liaison rapide et onéreuse quand cela est possible. Si les politiques de routage sont indispensables à un protocole de routage inter-domaine, elles interfèrent avec le processus normal de construction des vecteurs de chemin, et sont une source de l'instabilité du routage dans Internet [5,12].

[*] Ce travail a bénéficié du soutien de l'action CNRS ASTIC STAR et de l'action spécifique CNRS DYNAMO.

Les politiques de routage ne sont pas les seules responsables des problèmes de communication dans Internet. Dans [10], Jayaram et Varghese ont montré que si on fait l'hypothèse que les routeurs sont susceptibles de tomber en panne puis d'être redémarrés ultérieurement, alors le système peut être amené dans une configuration où l'état de la mémoire de chacun des routeurs est quelconque. Si de plus les liens de communication sont susceptibles de perdre ou dupliquer des messages, alors le système peut se retrouver dans une *configuration* (ou un état global du système) complètement arbitraire.
Une technique bien connue pour tolérer des défaillances de mémoire (telles qu'exécuter un système à partir d'une configuration initiale arbitraire) est l'*auto-stabilisation* [3]. L'auto-stabilisation a été introduite par Dijkstra en 1974 [2], comme une propriété d'un système qui garantit qu'à partir de n'importe quelle configuration initiale, le système converge vers une configuration correcte à partir de laquelle il vérifie sa spécification. Lamport en 1986 a souligné l'importance de ce concept dans le contexte de la tolérance aux fautes, puisqu'un système auto-stabilisant est en mesure de tolérer toute corruption des mémoires des processeurs et des liens de communication.

1.2 Etat de l'art

Récemment, le problème de la *stabilité* du routage inter-domaine dans Internet a fait l'objet de plusieurs publications [4,5,6,7,8]. Un protocole de routage inter-domaine est dit stable si en partant d'une configuration (ou état global du système) initiale *cohérente*, on aboutit forcément à une configuration *stable*. Une configuration est cohérente si les informations que chaque routeur possède à propos des caractéristiques du réseau (topologie, coût des communications entre systèmes autonomes) sont correctes. Une configuration est stable si les vecteurs de chemins vers les destinations sont stables (ils ne sont plus modifiés) et corrects (si on suit le chemin on atteint effectivement la destination). Dans [5], Griffin et Wilfong ont montré que dans le cas général (avec l'ensemble des politiques de routage autorisées par BGP), le problème de savoir si un système particulier était stable était un problème NP-complet ou NP-difficile suivant les hypothèses faîtes sur le réseau. A partir de là, les travaux ultérieurs [4,6,7] ont consisté à trouver des conditions *suffisantes* pour garantir la stabilité du système.
Une première approche [7] est de déterminer une *condition globale* (c'est-à-dire qu'elle s'applique sur l'ensemble du réseau) sur les politiques de routage de chaque système autonome qui, si elle est respectée, garantit que le système converge vers un état stable. Un problème de cette approche est qu'il est nécessaire de connaître toutes les politiques de routages de tous les systèmes autonomes, ce qui requiert des ressources importantes et qui implique une confiance du système autonome envers le vérificateur. Par ailleurs, à chaque ajout ou suppression de système autonome (voire de routeur), le mécanisme de vérification doit être exécuté à nouveau.
Une deuxième approche [4] est de déterminer une *condition locale* (c'est-à-dire qu'elle s'applique à un routeur particulier) sur les politiques de routages de chaque système autonome qui, si elle est respectée, garantit que le système converge vers un état stable. Cette approche ne résout que partiellement le problème posé précédemment car elle requiert que les relations entre systèmes autonomes voisins (client-fournisseur, ou pair à pair) soient clairement définies. Ceci implique que lors de l'ajout ou de la suppression de systèmes autonomes (voire de routeurs), il faut mettre à jour les informations concernant les relations entre systèmes autonomes voisins et vérifier à nouveau que les conditions sont localement vérifiées.
Une troisième approche [6] est *dynamique* (c'est-à-dire qu'elle intervient pendant l'exécution du protocole de routage, par opposition aux autres approches, *statiques*, qui nécessitent d'être exécutées avant ou après le protocole de routage). Elle consiste à stocker l'*histoire* des chemins vers chaque destination. Une histoire est constituée de la liste des chemins vers une destination, et comprend tous les chemins vers cette destination qui ont été calculés au cours de l'exécution du protocole de routage. Si une boucle apparaît dans cette histoire, c'est qu'un état stable ne pourra être atteint et le chemin est supprimé de la table de routage. Ce protocole ne souffre d'aucun des défauts des algorithmes précédents, mais requiert sur chaque routeur une quantité importante de mémoire (pour stocker l'histoire des chemins vers chaque destination).

1.3 Notre Contribution

Dans cet article, nous étudions la propriété d'auto-stabilisation du protocole de routage inter-domaine dans Internet en utilisant le même modèle que [5,8]. La propriété d'auto-stabilisation permet au protocole de démarrer depuis une configuration initiale arbitraire (par exemple suite à des défaillances du réseau, ou bien suite à l'ajout ou à la suppression de plusieurs routeurs dans des systèmes autonomes différents) et d'atteindre une

configuration cohérente à partir de laquelle une configuration stable pourra être atteinte en utilisant une des approches évoquées dans [4,6,7].
Plus précisément, nous montrons que :

1. Si le protocole BGP est configuré sans politique de routage (c'est-à-dire que la préférence est systématiquement donnée au chemin comportant le plus petit nombre de systèmes autonomes traversés, et que toutes les routes sont transmises aux systèmes autonomes voisins), alors il est auto-stabilisant ;

2. Si le protocole BGP est configuré de telle sorte que les contraintes sur les politiques vérifient la propriété globale de [7], alors il est auto-stabilisant ;

3. Si le protocole BGP est configuré de telle sorte que les contraintes sur les politiques vérifient la propriété locale de [4], alors il est auto-stabilisant ;

4. Si le protocole BGP est associé au protocole de vérification dynamique des histoires de [6], alors il est auto-stabilisant.

Pour des raisons de place, les preuves sont omises de ce résumé.

2 Modèle

2.1 Système réparti et auto-stabilisation

Un système réparti est un ensemble de machines qui peuvent communiquer entre elles *via* des liens de communication. Les machines dans le système réparti sont aussi appelées les processeurs, les sites ou les nœuds. Un système réparti est modélisé par un graphe de communication $G = (V, E)$ où V est un ensemble représentant les processeurs et E un ensemble représentant les liens de communication entre deux processeurs capables de communiquer directement.
L'algorithme localement exécuté sur chaque nœud est présenté au moyen de *variables partagées* et d'*actions*. Une *configuration* est le produit cartésien à un instant donné de toutes les variables de tous les nœuds du réseau. Chaque processeur peut lire ses propres variables partagées ainsi que celles de ses voisins, mais ne peut écrire que dans ses propres variables partagées. Chaque action est présentée sous la forme : *<label> ::<règle gardée>→<instruction>*. Une *règle gardée* est un prédicat sur les variables que peut lire le processeur, *l'instruction* n'est exécutée que si l'évaluation du prédicat correspondant renvoie *vrai*.

Un *système de transitions* est un couple $S=(C,\rightarrow)$, où C est un ensemble de configurations et $\rightarrow$ une relation binaire sur C. Une *exécution* de S est une suite maximale de configurations (finie ou infinie) $E = (\gamma_0, \gamma_1, \ldots, \gamma_i, \ldots)$ telle que pour tout $i\geq 0$, $\gamma_i \rightarrow \gamma_{i+1}$. Une *spécification* est un prédicat sur l'ensemble des exécutions.
Un système de transition $S=(C,\rightarrow)$ est *auto-stabilisant* pour la spécification P s'il existe un ensemble $L\subseteq C$ de configurations légitimes vérifiant les propriétés suivantes : (1) *Correction*: toute exécution débutant par une configuration dans L satisfait P ; (2) *Convergence* : toute exécution atteint une configuration dans L.

La complexité en espace d'un algorithme est la taille (en bits) des variables utilisées localement par chaque processeur. La complexité en temps d'un algorithme auto-stabilisant est le nombre de tours nécessaires pour parvenir à une configuration légitime (au cours d'un tour, chaque processeur exécute ou tente d'exécuter son code local).

2.2 Modèle de BGP

Dans cette partie, nous reprenons pour partie le modèle de [8], en représentant Internet par un graphe connexe bidirectionnel $G = (V, E)$, appelé graphe d'AS (d'après l'anglais Autonomous System). Chaque nœud v dans V est un AS et est numéroté de 0 à n. Pour plus de simplicité, chaque AS n'a qu'un seul routeur, ce qui permet de considérer v comme un routeur BGP. Chaque arête dans E présente une liaison entre deux routeurs BGP dans deux AS différents. L'ensemble des voisins de $v\in V$ est déterminé par $Voisins(v) = \{u|(u, v)\in E\}$. Sans perte de

généralité, nous supposons en outre qu'il n'y a qu'une seule destination (le noeud *0*), tous autres noeuds essayant d'établir un chemin vers *0*. Pour que ce modèle puisse s'adapter à des algorithmes de type vecteur de chemins, dans la suite le mot *chemin* dénote l'attribut *AS_PATH* du protocole BGP et le mot *route* est un ensemble d'attributs du chemin qui dépend de l'algorithme utilisé.

Un chemin P dans G est une séquence ordonnée de noeuds $P = (v_k, v_{k-1}, \ldots, v_0)$ où $k \geq 0$. Si $k = 0$, le chemin P est *vide* et noté ε. Si $k>0$, alors v_k est noté *Suivant(P)*. Un chemin est *incorrect* s'il existe $(v_i, v_{i-1}) \notin E$. Un chemin non vide $P = (v_p, v_{p-1}, \ldots, v_0)$ et un chemin $Q = (w_k, w_{k-1}, \ldots, w_0)$ peuvent être enchaînés pour devenir un chemin PQ si $v_0=w_k$. Le résultat s'écrit : $PQ = (v_p, v_{p-1}, \ldots, v_0, w_k, w_{k-1}, \ldots, w_0)$. Q est appelé *sous-chemin* de PQ de longueur k. Pour chaque chemin P, l'enchaînement de P avec un chemin vide est lui-même. Quand P est un chemin commençant par v et que $(u,v) \in E$, on dit que $(uv)P$ est un chemin *étendu* de P à v. Un chemin $P = (v_k, v_{k-1}, \ldots, v_0)$ est un chemin *simple* si tout $v_i \in P$ est unique. Dans cet article, nous considérons uniquement les chemins simples, car BGP abandonne tous les chemins qui contiennent des cycles.

Chaque nœud v dans le système possède une *fonction de priorité* notée λ^v. S'il y a un choix à effectuer entre deux chemins, le chemin avec la valeur de la priorité la plus élevé est choisi. Un *vecteur de chemins* est déterminé par la fonction π qui associe à chaque nœud $v \in V$ un chemin $P_v = \pi(v)$. π s'écrit $(P_n, P_{n-1}, \ldots, P_1)$ avec $P_i = \pi(i)$, pour $1 \leq i \leq n$. π est un vecteur de chemin *correct* si tous les chemins de π sont corrects (c'est-à-dire qu'ils existent). Soit π un vecteur de chemins, l'ensemble des chemins offerts à u et noté *Choix(u)* est défini comme suit :

$$Choix(u,\pi)=\{(u\ v)Import(\pi(v))|(u\ v)\in E\}$$

Où $Import(\pi(v))$ dénote un sous-ensemble des vhemins de $\pi(v)$.
Si maintenant on pose $W = Choix(u,\pi)$, le meilleur chemin partant de u sur le vecteur π (noté $Meilleur(u, \pi)$) est déterminé par $Max(u, W)$:

$$Max(u,W)=\begin{cases} P\in W \,|\, \lambda^u(P) \ est\ le\ plus\ élevé\,,\ si\ W\ est\ non\ vide \\ \varepsilon,\ si\ W\ est\ vide \end{cases}$$

Un vecteur de chemin π est *stable* si et seulement si tout chemin $\pi(v)$ de v est un meilleur chemin de v sur π. Une *transformation* d'une route T est une fonction qui transforme une route en une autre en supprimant, ajoutant, ou modifiant l'un des attributs de la route ; on note $T(r)=r'$ la transformée de la route r par T.
Dans la suite, $u \Leftarrow w$ dénote un ensemble de routes de w à u. Un processus de mouvement d'une route de w à u consiste en trois transformations. Tout d'abord, $r_1=Export(u \Leftarrow w,r)$ est le résultat de l'application des politiques d'export configurées sur w pour r, et s'exprime au moyen d'une fonction qui renvoie une nouvelle route. Ensuite w ajoute son numéro dans le chemin r_1 et détecte le cycle de chemin, $r_2=PVP(u \Leftarrow w, r_1)$, où PVP dénote un protocole de vecteurs de chemins. Finalement, u applique les politiques d'import configurées pour décider d'utiliser ou non la route r_2, et le résultat est $r_3= Import(u \Leftarrow w, r_2)$. Ces trois transformations peuvent être considérées comme une opération atomique $pt(u \Leftarrow w, r)$. Un chemin $P=(u\ v_k\ v_{k-1}\ \ldots\ v_0)$ est autorisé sur u si (1) P est vide ou si (2) P satisfait la condition suivante :

$$P=Chemin(pt(u \Leftarrow v_k, pt(u \Leftarrow w, r_{k-1}) \ldots\ pt(u \Leftarrow w, r_0)))$$
où $Chemin(r_0)=0$

Un chemin autorisé P de longueur k est autorisé sur v_{k+1} si sa route r satisfait la politique d'import, $Import(v_{k+1} \Leftarrow v_k,r)$ et si la route de son sous-chemin de longueur $k-1$ est autorisée sur v_k. Chaque noeud $u \in V \setminus \{0\}$ possède un ensemble de chemins P^u correspondant à tous les chemins autorisés sur u. Cet ensemble est inclus dans l'ensemble de tous les chemins corrects de u à 0 dans G. Soit P^S l'ensemble de tous les chemins autorisés dans le système : $P^S=\{P^v \mid \forall v \in V \setminus \{0\}\}$.
Soit $\nabla=\{\lambda^v|v\in V\setminus\{0\}\}$ l'ensemble de toutes les fonctions de priorité de $v \in V\setminus\{0\}$, C_S l'ensemble de toutes les configurations du système contenant uniquement des vecteurs de chemins autorisés. Nous appelons $SPP = (G,C_S,\nabla)$ une instance du problème de *stabilité des chemins*, avec $G=(V, E)$ et $v \in V$ si les conditions suivantes sont satisfaites :

1. Le chemin vide est autorisé : $\varepsilon \in P^v$;

2. Le chemin vide a la priorité la plus faible : $\lambda^v(\varepsilon) = 0$;

3. Si $\lambda^v(P_1)=\lambda^v(P_2)$, alors soit $P_1=P_2$ soit $P_1=(v\ u)P_1'$ et $P_1=(v\ u)P_1'$;

4. Si $P \in P^v$, alors P est un chemin simple.

2.3 Un algorithme simple de vecteur de chemin (PVP1)

Soit $u \in V$, $w \in Voisin(u)$. Chaque nœud u dans le système possède deux variables principales :

1. $rib(u)$ contient le chemin le plus récent de u à la destination 0. Dans notre modèle, $rib(u)$ correspond à $\pi(u)$.

2. $rib_out(u \Leftarrow w)$ contient le chemin que w va envoyer à u.

Pour plus de simplicité dans la présentation de l'algorithme, nous définissons la fonction *Meilleur(u)* correspondant au processus de décision d'un chemin :

$$Choix(u)=\{P \mid P= Import\ (u, rib_out(u \Leftarrow w)), w \in Voisin(u)\}$$
$$Meilleur(u)=Max(u,Choix(u))$$

Algorithme pour le nœud u (PVP1)

Variable partagée :

$rib_out(w \Leftarrow u)$

Actions :

$Décision :: rib(u) \neq Meilleur(u) \rightarrow$
$rib(u) := Meilleur(u)$;
$rib_out(w \Leftarrow u) := Export(Meilleur(u))$;

Si π est le vecteur de chemin, le graphe de routage $G_\pi=(V_\pi, E_\pi)$ (avec $\pi=(P_1,P_2,\ldots,P_n)$) est construit en ajoutant toutes les paires (w, v) où w et v sont les deux premiers nœuds de chaque chemin P_i. Le graphe G_π est un *arbre de routage* si G_π est un arbre. Formellement, G_π est construit comme suit :

$$V_\pi = \cup_{1 \le i \le n, P_i \in \pi} \{(w, v) \in V \wedge P_i = (wv \ldots 0)\}$$
$$E_\pi = \cup_{w,v \in V_\pi} \{(w, v) \in E\}$$

Soit π le vecteur de chemin de c_L. La configuration c_L est *légitime* si et seulement si elle satisfait les deux conditions suivantes :

1. Le vecteur de chemin π de la configuration c_L est un arbre de routage ;

2. Le chemin de chaque nœud u est le meilleur chemin de u sur π, $\pi(u)=Meilleur(u,\pi)$.

3 Résultats

3.1 BGP Sans Politique

Dans cette section, nous considérons un protocole BGP simple où aucune politique n'a été configurée. Un tel système est similaire à un système qui utilise l'algorithme PVP1 et où tous les nœuds $v \in V$ utilisent les mêmes politiques import et export : (1) v envoie tous les chemins appris à tous ses voisins, (2) v accepte tous les chemins reçus dans le processus de décision. Dans un tel système, P^v contient tous les chemins corrects possibles. De plus, la valeur de priorité déterminée par la fonction λ^v de chaque nœud $v \in V$ n'est basée que sur deux propriétés du chemin : la longueur du chemin et le numéro du *Suivant*. Si le chemin de longueur la plus courte est unique, alors il est choisi. Dans le cas contraire, un chemin dont le numéro de *Suivant* est le plus petit est choisi.

Théorème : *BGP sans politique est auto-stabilisant. La mémoire utilisée sur chaque nœud est de* $(2n^2 + n) \times T_p$ *(où n est le nombre de nœuds du système, et où* T_p *est la longueur maximum d'un chemin dans G). La solution stable est atteinte en D rounds, où D est le diamètre de G.*

3.2 BGP avec Conditions Locales

L'algorithme pour cette solution est le même que PVP1. La solution que nous considérons dans cette section est suggérée par Gao, Griffin et Rexford [4]. C'est une solution statique. Au lieu de considérer tous les AS comme égaux, on distingue plusieurs types de relations hiérarchiques entre AS voisins : une relation de type *Fournisseur-Client* existe entre un fournisseur et un client (le fournisseur fournit la connexion du reste de l'Internet au client), et une relation de type *Ami-Ami* existe entre deux AS qui sont d'accord pour échanger le trafic de leurs clients.
Dans ce modèle, chaque nœud v dans le système doit configurer manuellement ses fournisseurs, ses clients et ses amis (notés *Fournisseur(v), Client(v), Ami(v)*). A partir des relations entre les nœuds du système, nous pouvons distinguer trois types de chemins :

1. *chemin de fournisseur* : un chemin P de $v \in V$ est chemin de fournisseur si et seulement si $Suivant(P) \in Fournisseur(v)$;

2. *chemin de client* : un chemin P de $v \in V$ est chemin de client si et seulement si $Suivant(P) \in Client(v)$;

3. *chemin d'ami* : un chemin P de $v \in V$ est chemin d'ami si et seulement si $Suivant(P) \in Ami(v)$.

L'auto-stabilisation du protocole est assurée si les règles suivantes sont localement vérifiées par chaque processeur :

1. *règles d'export :*
   a. Les chemins de fournisseur ne doivent pas être diffusés aux autres fournisseurs ;
   b. Les chemins d'ami ne doivent pas être diffusés aux fournisseurs et vice-versa ;
   c. Les chemins d'ami ne doivent pas être diffusés aux autres amis.

   Ces règles sont conformes aux politiques appliquées dans le modèle commercial de l'Internet 0, et elles restreignent peu la flexibilité de BGP.

2. *règle d'import* : Pour chaque $v \in V$, soient P_1 et P_2 deux chemins pour la même destination. La politique d'import appliquée à v doit satisfaire la condition suivante :

   Si

   $$Suivant(P_1) \in Client(v) \wedge Suivant(P_2) \in (Ami(v) \cup Fournisseur(v))$$

   Alors

   $$P_1.Loc_pref > P_2.Loc_pref$$

   Cette règle signifie que s'il existe plusieurs chemins possibles, alors le chemin de client est le chemin de priorité la plus élevée. Dans la pratique, cette règle ne limite pas la capacité de préférence du nœud, qui peut choisir un chemin de client, de fournisseur ou d'ami.

Théorème : *BGP avec les règles d'import et d'export satisfaites sur chaque processeur est auto-stabilisant. La mémoire utilisée sur chaque nœud est de* $(2n^2 + n) \times T_p$ *(où n est le nombre de nœuds du système, et où* T_p *est la*

longueur maximum d'un chemin dans G). La solution stable est atteinte en 2n tours, où n est le nombre de nœuds du système.

3.3 BGP avec Conditions Globales

Les politiques configurées sur chaque nœud peuvent être contradictoires (voir [7]). Pour identifier les politiques contradictoires entre elles, considérons deux nœuds u et v où $v \in Voisin(u)$. Supposons que u veut envoyer un message à la destination *via* son voisin v. Le nœud v peut choisir entre deux chemins P et Q pour la destination. De plus, u peut également choisir entre $(u\ v)P$ et $(u\ v)Q$. Nous pouvons illustrer ce cas au moyen de la Figure 1 :

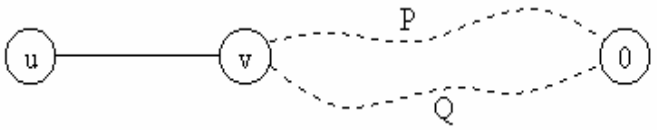

Figure 1

Nous considérons deux cas particuliers dans la relation entre un chemin autorisé de v et un autre de u. Un couple de la forme $(P, (u\ v)P)$ est une relation de transition et un couple de la forme $(Q, (u\ v)P)$ est une relation de conflit si les conditions suivantes sont satisfaites :

1. P et Q sont des chemins autorisés sur v ;
2. $(uv)P$ est un chemin autorisé sur u ;
3. sur le nœud v, on a $\lambda^v(P) \leq \lambda^v(Q)$;
4. sur le nœud u, on a $(uv)Q \notin P_u$ ou $\lambda^u((uv)Q) < \lambda^u((uv)P)$.

Le *graphe de conflit* de $SPP=(G, C_S, \lambda)$ est un graphe orienté $G_c(SPP)$ où les nœuds du graphe sont étiquetés par les chemins autorisés de SPP. Il y a 2 types d'arcs : les arcs de transition et les arcs de conflit. Un arc de transition correspond exactement à la relation de transition $(P, (u\ v)P)$, et il est dirigé de P vers $(u\ v)P$ et noté $P \rightarrow (u\ v)P$. Un arc de conflit correspond exactement à la relation de conflit $(Q, (u\ v)P)$, et il est orienté de Q vers $(u\ v)P$ et noté $Q \rightarrow (u\ v)P$.

Théorème : *Si le graphe de conflit de SPP du système BGP est acyclique, alors le système est auto-stabilisant. La mémoire utilisée sur chaque nœud est de* $(2n^2 + n) \times T_p$ *(où n est le nombre de nœuds du système, et où* T_p *est la longueur maximum d'un chemin dans G).*

3.4 BGP avec histoire de construction des routes

Dans [6], Griffin et Wilfong ont proposé une solution dynamique qui garantit que le protocole ne divergera pas. Cette solution modifie l'algorithme de base pour empêcher les routes produites par les conflits de politiques. Ces mécanismes sont basés sur la définition de l'histoire du chemin.
L'*histoire d'un chemin* est une sequence $h = e_k\ e_{k-1}\ e_1$ où e_i est l'événement de changement du chemin, c'est à dire un couple $e = (s, P)$ où $s = \{+, -\}$ est le signe de e et P est un chemin. Une histoire de chemin $h = e_k\ e_{k-1}\ e_1$ contient un cycle s'il existe e_i, e_j où $P_i = P_j$. Le calcul de l'histoire à chaque nœud u (c'est à dire le remplacement d'un chemin P_{ancien} par un chemin $P_{nouveau}$) est réalisé comme suit :

1. si $\lambda^u(P_{ancien}) < \lambda^u(P_{nouveau})$, $P_{nouveau} = (uw)P_2$ alors l'histoire de ce chemin est $(+, P_{nouveau})h$, où h est l'histoire de P_2 reçue de son voisin w ;
2. si $\lambda^u(P_{ancien}) > \lambda^u(P_{nouveau})$, $P_{ancien} = (uw)P_1$, alors l'histoire de nouveau chemin est $(-, P_{ancien})h$, où h est l'histoire la plus récente que u a reçu de son voisin w ;

3. si $\lambda^u(P_{ancien})=\lambda^u(P_{nouveau})$, $P_{nouveau}=(uw)P_2$, $P_{ancien}=(uw)P_1$, alors l'histoire de nouveau chemin est $(s, Q)h$, où le signe s est le signe du plus récent événement de changement dans h, et $Q = P_{ancien}$ si s = - et $P_{nouveau}$ sinon.

Pour prévenir la construction de chemins dont l'histoire contient un cycle, chaque processeur possède une variable supplémentaire $B(u)$. Par ailleurs, les variables initiales de l'algorithme contiennent également l'histoire des chemins qu'elles véhiculent. Dans le processus de sélection d'un meilleur chemin, seuls les chemins ne figurant pas dans $B(u)$ peuvent être choisis.

Théorème : *BGP avec prévention des cycles dans les histoires de construction de chemins est auto-stabilisant. La mémoire utilisée sur chaque nœud est de* $(n^3 + n^2+2n) \times T_p$ *(où n est le nombre de nœuds du système, et où* T_p *est la longueur maximum d'un chemin dans G).*

4 Conclusion et perspectives

Le problème général de savoir si un système BGP est stable on non est NP-complet ou NP-difficile. Dans cet article, nous avons montré que lorsque qu'il n'était pas fait usage des politiques de routages, le protocole BGP était auto-stabilisant, c'est-à-dire capable de revenir en un temps fini à une configuration correcte à partir de toute configuration initiale. De plus, nous avons montré que les approches [4,6,7] qui garantissent la stabilité de BGP dans des cas plus généraux, garantissent également l'auto-stabilisation du protocole résultant dans les mêmes conditions. Ce travail ouvre la voie à l'étude des protocoles stables et auto-stabilisants pour le routage dans Internet.

Références

[1] Y. Chen, A.K. Datta, et S. Tixeuil. Stabilizing Inter-domain Routing in the Internet. Europar 2002, pp. 749-752, Août 2002.

[2] E. W. Dijkstra. Self-stabilization in spite of distributed control. Communications of the ACM, vol. 17, pp. 643-644, 1974.

[3] S. Dolev. Self-stabilization. The MIT Press, 2000.

[4] L. Gao, T. Griffin et J. Rexford. Inherently Safe Backup Routing with BGP. IEEE INFOCOM'01, vol 1, pp. 547-556, Avril. 2001.

[5] T. Griffin et G.Wilfong. An Analysis of BGP Convergence Properties. ACM SIGCOM'99, pp. 277 - 288, 1999.

[6] T. Griffin et G.Wilfong. A Safe Path Vector Protocol. IEEE INFOCOM, pp. 490-499, Mars 2000.

[7] T. Griffin, F. B. Shepherd, et G. Wilfong. Policy Disputes in Path-vector Protocols. ICNP'99, pp. 21-30, Octobre 1999.

[8] T. Griffin , F. B. Shepherd , G. Wilfong. The Stable Paths Problem and Inter-domain Routing. IEEE/ACM, pp. 232 - 243 , Avril 2002.

[9] G. Huston. Interconnection, Peering, and Settlements. Http://www.potaroo.net/inet99/peering.html.

[10] M. Jarayam et G. Varghese. The Fault Span of Crash Failures. Journal of the ACM, vol. 47, no. 2, pp. 244-293, 2000.

[11] C. Labovitz, G. R. Malan, et F. Jahanian. Internet Routing Instability. ACM SIGCOM, pp. 115 - 126, Septembre 1997.

[12] C. Labovitz, G. R. Malan, et F. Jahanian. Origins of Internet Routing instability. IEEE INFOCOM, Mars 1999.

[13] R. Perlman. Interconnections : Bridges, Routers, Switches, and Internetworking Protocols. Addison-Wesley, Hardcover, pp. 299-445, Septembre 1999.

[14] Y. Rekhter et T. Li. A Border Gateway Protocol 4 (BGP-4). IETF, RFC 1771, Mars 1995.