Ecole Polytechnique De Grenoble

RICM 2

RAPPORT DE STAGE

Implémentation dans un simulateur de réseau BGP d'une méthode de résolution des oscillations

effectué au LIRMM

du 01 Juin au 31 Août 2006

par

Joseph Emeras

Directeurs de stage:

M. Clément Saad, LIRMM, Montpellier ..... Doctorant
M. Ehoud Ahronovitz, LIRMM, Montpellier ..... Maître de conférence

Remerciements

Je tiens tout d'abord à remercier particulièrement M.Ehoud Ahronovitz de m'avoir proposé ce stage au sein du LIRMM, de m'avoir accordé sa confiance et de m'avoir donné de précieux conseils. Je le remercie aussi pour sa gentillesse et son implication dans ce stage.

Je remercie également M.Clément Saad, qui a été la personne qui m'a principalement encadré. Je le remercie pour avoir pris le temps de m'aider dans la compréhension des problèmes qui m'étaient posés, pour m'avoir donné de précieux conseils durant mon stage et également pour la rédaction de ce rapport. Je le remercie également pour sa bonne humeur et son acceuil chaleureux.

Je tiens à remercier aussi M.Bruno Quoitin, de l'Université Catholique de Louvain, pour avoir créé le simulateur C-BGP. Je le remercie aussi spécialement de m'avoir tant aidé dans la compréhension de son simulateur et pour ses précieux conseils techniques sur le simulateur comme sur le langage C.

Je remercie de même M.Julien Champ, le stagiaire qui avait effectué la première partie du travail sur C-BGP et dont l'aide me permis de gagner du temps au début du stage.

Je remercie bien entendu tous les autres doctorants de la Halle Gorithme de m'avoir acceuilli au sein de leur bureau, ainsi que les personnes que j'ai pu rencontrer au LIRMM pour les discussions enrichissantes que j'ai pu partager avec eux.

Enfin je remercie Daniel Luedemann à qui je dois mes rares moments de détente au cours de ce stage.

Table des matières

Résumé

Dans le cadre de ma deuxième année d'ingénieur, j'ai eu l'opportunité d'accomplir un stage de trois mois au sein du LIRMM, ce stage avait pour but la vérification d'une méthode d'amélioration du protocole Border Gateway Protocol (BGP).

Les systèmes autonomes (AS) dans l'Internet utilisent BGP comme protocole de routage externe. Il permet aux AS de définir de façon indépendante leur politique de routage. Lorsque les politiques ne sont pas en conflit, BGP est auto-stabilisant, ce qui signifie que quelque soit la configuration du réseau, BGP converge vers une solution stable. Malheureusement les politiques des AS peuvent être incohérentes et générer ainsi des oscillations. Une méthode pour détecter et résoudre ces oscillations dans l'Internet a été proposée dans [1]. Cette méthode, basée sur l'utilisation d'un jeton n'avait pas encore été testée dans la pratique. Un précédent stagiaire, Julien Champ (voir [2]), avait déjà implémenté la première partie de cette solution au cours d'un stage d'IUP, à savoir la partie de détection et de résolution des oscillations.

Mon stage avait pour but de continuer la vérification de la validité de cette méthode et d'adapter le travail déjà effectué au fonctionnement réel de BGP dans l'Internet.Dans ce but, j'ai donc dû dans un premier temps étudier le protocole BGP, puis la méthode proposée pour résoudre les oscillations ainsi que les modifications de topologies.

Ensuite, j'ai implémenté la partie de gestion des pannes et d'apparition de liaisons entre les routeurs BGP et j'ai adapté le travail de Julien à cette amélioration.

Ce rapport présente le LIRMM, où j'ai effectué mon stage, le protocole BGP et la méthode de détection et de résolution des oscillations, ainsi que le travail que j'ai accompli, de l'adaptation des algorithmes existants pour la gestion des changements topologiques, à l'implémentation de la méthode.

Introduction

BGP (Border Gateway Protocol) est l'un des protocoles de routage externe utilisé dans l'Internet. Ce protocole permet aux systèmes autonomes (par la suite nous utiliserons l'abbréviation AS pour Autonomous System) d'échanger des informations sur le routage à effectuer.

Un AS est un ensemble de réseaux et de routeurs sous une administration unique. Chaque AS choisit son protocole de routage interne mais aussi sa politique de routage externe. BGP laisse la possibilité à chaque AS d'appliquer sa propre politique en fonction de ses préférences (critères commerciaux, de performance ou de sécurité par exemple). Or, dans certains cas, le choix de certaines politiques peut entrainer des incohérences et causer ainsi des oscillations de routes (instabilités) dont nous verrons la signification plus loin dans ce rapport.

Une solution a été proposée dans [1] pour résoudre ces oscillations. Cette méthode respecte les contraintes imposées par BGP et permet la détection et la résolution de ces oscillations. Elle est basée sur l'utilisation d'un jeton : lorsqu'un AS détecte une oscillation, il génèrera un jeton associé à une route et l'expédiera à ses voisins. Les AS le recevant auront le choix de le faire suivre ou non. Lorsqu'un jeton reviendra à l'AS qui l'a précédemment généré celui-ci interdira alors la route associée, résolvant ainsi l'oscillation. Le principe de confidentialité doit être respecté c'est pour cela que toutes les informations échangées, comme le jeton par exemple, ne doivent pas contenir des éléments permettant de reconstruire la politique d'un AS.

L'implémentation de cette méthode de résolution des oscillations au sein d'un simulateur devrait permettre d'étudier le comportement de cette méthode, afin de valider ou invalider la solution proposée.

Dans le chapitre 1 je vous présenterai le laboratoire de recherche qui m'a accueilli. Dans le chapitre 2, je vous ferai une description brève du protocole BGP et du principe des oscillations, puis dans le chapitre 3 je présenterai la méthode de résolution de ces oscillations proposée dans [1]. Dans le chapitre 4 je vous presenterai comment [1] propose de gérer les pannes et apparitions de liens entre routeurs BGP. Je vous décrirai, dans le chapitre 5, le travail que j'ai effectué, ainsi que les résultats que j'ai obtenu. Le chapitre 6 sera consacré à une brêve discussion sur les difficultés rencontrées au cours de ce stage.

Chapitre 1

Le LIRMM

1.1 Présentation générale

Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier est une unité mixte de recherche de l'Université Montpellier II (UMII) et du Centre National de la Recherche Scientifique (CNRS), département Sciences et Technologies de l'Information et de la Communication (STIC)

LIRMM / UMR 5506
161, rue Ada
34392 Montpellier Cedex 5 - France

Le spectre des activités de recherche du LIRMM est très large et va de la conception de circuits, à la modélisation de systèmes complexes à base d'agents en passant par des études algorithmiques, la bioinformatique, les interactions homme-machine, la robotique.

Ces activités sont principalement développées au sein des trois départements scientifiques de recherche qui composent le LIRMM :

- Informatique (INFO)
- Microélectronique (MIC)
- Robotique (ROB)

1.2 Département Informatique

Responsable : Marianne Huchard, Professeur à l'UM II
Adjoints : Alain Jean-Marie, Frédéic Koriche et Patrice Séébold.

Le Département d'Informatique du LIRMM regroupe quatre-vingt chercheurs et enseignant-chercheurs, soixante doctorants et dix post-doctorants et ingénieurs. Les thématiques couvrent l'essentiel de la recherche actuelle en Informatique et ses applications.Il est structuré en dix équipes-projets :

Algorithmique et performances des réseaux (APR)

Conception et analyse de protocoles de réseaux : routage et diffusion de l'information, avec garanties de qualité de service. C'est dans ce département que j'ai effectué mon stage.

Arithmétique informatique (ARITH)

L'objectif de ce projet est de développer un groupe centré autour de l'arithmétique informatique et de la combinatoire en s'appuyant sur l'arithmétique des ordinateurs, la numération, l'infographie et la géométrie discrète.

Données Objets Connaissances pour les systèmes complexes (DOC)

Ce projet allie des recherches sur la modélisation de systèmes complexes, les langages à objets et de composants, les architectures de médiation et la représentation de connaissances et de raisonnements.

Ingénierie des Données et des Connaissances (IDC)

Médiation de données dans les systèmes distribués à grande échelle : modèles d'intégration de données axés sur des techniques de fouille de données et maintien de la cohérence.

Interaction homme-machine et Hypermédias (IHMH)

Étudie les aspects informatiques et cognitifs des échanges d'information au sein de systèmes complexes lorsque ceux-ci mettent en jeu au moins un partenaire humain. Ex : production de documents, hypermédias, conception d'interfaces utilisateur, interactions naturelles, bibliothèques digitales,...

Agents, apprentissage, contraintes et logique (KAYOU)

Thèmes de l'IA orientés autour de deux axes : l'étude des agents rationnels et l'étude des sociétés d'agents. Les outils sont les systèmes multi-agents, l'apprentissage automatique, le raisonnement logique, et les réseaux de contraintes.

Méthodes et algorithmes pour la bio informatique (MAB)

Analyse des séquences (gènes, génomes, protéines), reconstruction de l'évolution (phylogénie, histoire des duplications), analyse statistique du transcriptome (puces à ADN, SAGE) et du protéome (gels 2D, spectroscopie de masse).

Traitement algorithmique du langage (TAL)

Analyse morpho-syntaxique et sémantique du Francais, classification automatique de documents, recherche d'information et traduction automatique.

Visualisation et algorithme des graphes (VAG)

L'objectif du projet, axé sur la visualisation de graphes, est d'utiliser à la fois des techniques issues de l'algorithmique et la théorie des graphes ainsi que d'outils de visualisation pour des applications telles que le "clustering".

Web sémantique et e-learning (WEB)

E-learning, Systèmes d'informations, Web sémantique, ontologies, XML, métadonnées, Environnements Informatiques pour l'Apprentissage Humain, apprentissage collaboratif

Chapitre 2

BGP

BGP (Border Gateway Protocol) est l'un des protocoles de routage utilisé dans l'Internet. Nous allons voir une brève description de ce protocole. Afin d'obtenir plus de précisions sur ce protocole vous pouvez consulter des ouvrages tels que : [3] et [4].

2.1 Principe

BGP a pour objectif la transmission d'informations de routage entre systèmes autonomes à travers l'Internet. Chaque système autonome est un ensemble de réseaux et de routeurs sous une administration unique. Un AS décide lui même de son protocole de routage interne (RIP, OSPF,...) ainsi que sa politique de routage externe. BGP va devoir satisfaire tout d'abord les exigences des politiques de routage de l'AS (en fonction par exemple de préférences économiques ou sécuritaires), mais aussi respecter le principe de confidentialité de ces politiques. Les routeurs BGP ne doivent pas communiquer d'informations permettant à un AS de reconstruire la politique d'un autre AS.

Un AS contient un routeur BGP ou plus, et chacun d'entre eux appliquent la même politique de routage externe définie par l'AS. (pour une destination tous les routeurs utiliseront les mêmes chemins externes.) Pour un AS, tous les routeurs d'un autre AS apparaitront comme un seul routeur. On peut représenter un réseau d'AS comme un graphe où les sommets seront les AS et les arêtes les liens BGP. Ce graphe sera alors un multi-graphe étant donné qu'il pourra y avoir plusieurs liens BGP entre deux sommets.

Il existe plusieurs types de relations entre deux AS :

- relation "fournisseur - client" : Cela signifie qu'un AS 'fournisseur' autorise, généralement moyennant finance, le transit d'information pour un 'client' donné.
- relation de "pair à pair" : Ce type de relation existe généralement suite à une entente entre deux AS.

La figure 2.1 est un exemple de réseau. On peut imaginer qu'il y a une relation de "pair à pair" entre les deux universités et une relation "fournisseur - client" entre l'université A et le Fournisseur d'Accès Internet.

L'université A préfèrera utiliser le lien direct avec l'autre l'université si elle désire communiquer avec un ordinateur du réseau de l'université B, pour des raisons de coût.

La figure 2.2 représente la modélisation du réseau de la figure 2.1

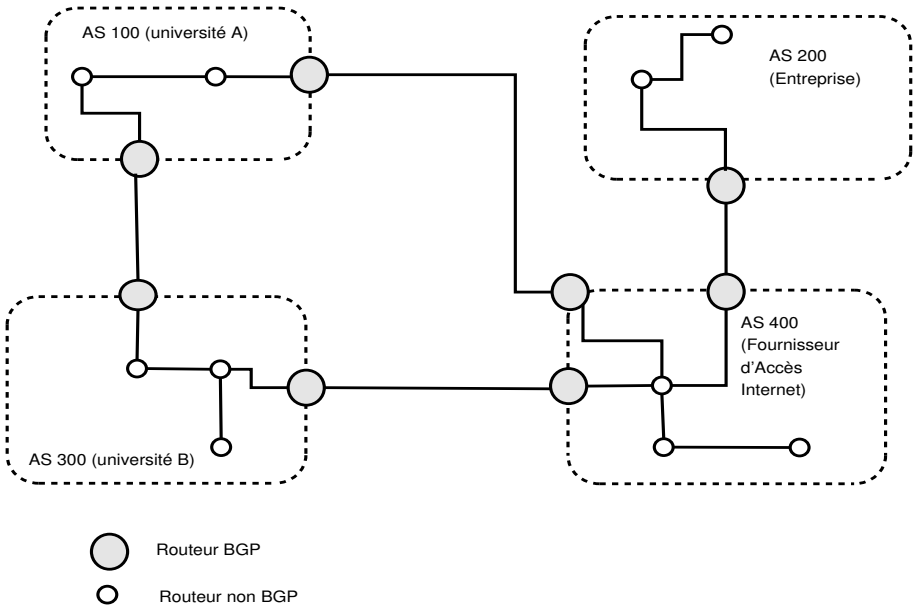

FIG. 2.1 – Exemple de réseau.

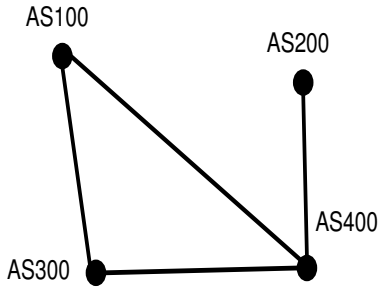

FIG. 2.2 – Modélisation du réseau.

2.2 Annonces de routes

Un réseau désirant se faire connaitre informe ses voisins BGP de son existence. Les voisins peuvent ignorer ou prendre en compte cette information. Dans le cas ou un voisin la prend en compte, il l'annoncera alors à tous ses voisins. Si un AS annonce une route, il s'engage à accepter le trafic vers cette destination ; sinon il ne l'annonce pas. BGP est capable d'agréger les routes lors de l'annonce d'une route (dans le but de diminuer la taille des tables de routage). Ainsi, si un AS s'occupe de tous les réseaux ayant pour préfixe 200.100.0 jusqu'à 200.100.255, il pourra les agréger et annoncer seulement 200.100.

Une annonce de route sera composée des attributs suivants :

- *AS_PATH* : Indique la route suivant une liste ordonnée.
- *NEXT_HOP* : indique l'adresse IP du prochain routeur BGP.
- *LOCAL_PREF* : attribut interne à un AS utilisé lors du routage externe. Il n'est jamais communiqué aux autres AS ; c'est lui qui fixe le degré de préférence accordé à chaque route.
- *ATOMIC_AGGREGATE* : indique s'il y a eu une agrégation de routes.
- *AGGREGATOR* : indique l'AS ainsi que l'IP du routeur qui a effectué l'agrégation.
- *MULTI_EXIT_DISCRIMINATOR (MED)* : permet, lorsque deux AS sont interconnectés à l'aide de plusieurs liens, d'en discriminer en associant à chaque lien un degré de préférence. (Cet attribut peut être la cause d'oscillations internes aux AS. Pour plus de détails voir : [5] et [6])

2.3 Processus de sélection

BGP peut être divisé en deux parties : E-BGP et I-BGP. E-BGP est chargé des communications entre AS et I-BGP des communications internes à l'AS. Le processus de sélection de BGP consiste à trouver le *meilleur chemin* parmi les chemins possibles suivant le schéma ci-desous ; le processus s'arrêtera lorsqu'il n'aura plus qu'un seul chemin.

1. Choisir les routes ayant le plus grand *LOCAL_PREF*
2. Choisir les routes traversant le moins d'AS.
3. Pour chaque voisin, sélectionner les routes qui ont le plus petit *MED*.
4. S'il reste au moins une route E-BGP, c'est à dire annoncé par un AS voisin et non un routeur du même AS, éliminer toutes les routes qui passent au travers de l'AS (route annoncée en I-BGP). Cette étape permet d'éviter la surcharge de son propre réseau.
5. Utiliser une information déterministe (par exemple choisir la route qui a la plus grande adresse IP dans *NEXT_HOP*).

2.4 Filtrage d'annonces

Un annonceur BGP est constitué de trois tables appellées RIB (Routing Information Base) :

1. *Adj-RIBs-In* : contient les informations de routage apprises par les routeurs BGP voisins.
2. *Loc-RIB* : contient les informations de routage que le routeur BGP a sélectionné suivant les politiques locales parmi les informations contenues dans Adj-RIBs-In.

3. *Adj-RIBs-Out* : contient les informations de routage qui peuvent être communiquées à un autre routeur BGP spécifique.

Lorsqu'un routeur BGP reçoit une annonce d'une nouvelle route, il applique alors la politique de filtrage en entrée. Les annonces respectant la politique de l'AS seront placées dans la table Adj-RIBs-In, les nouveaux chemins seront placés dans la table Loc-RIB. Ensuite ces nouveaux chemins subiront un nouveau filtrage pour savoir lesquels d'entre eux seront réannoncés aux voisins. Ce filtrage s'effectue une nouvelle fois en fonction de la politique de l'AS. Les chemins sélectionnés seront placés dans la table Adj-RIBs-Out pour être réannoncés.

2.5 Oscillations dans BGP

Nous avons vu que les politiques de routages des AS sont privées, ainsi il est impossible qu'une autorité quelconque impose quelque règle que ce soit. En conséquence, chaque AS définit sa propre règle qui lui convient sans se soucier de ce que cela peut engendrer. C'est à cause de ces politiques que des oscillations, c'est à dire une "non-convergence" du protocole BGP vers un état stable, peuvent apparaitre. S'il y a oscillation, BGP est donc incapable d'arriver à une solution stable et peut créer des situations problématiques et même incohérentes. Ces oscillations ont pour conséquences, entre autres, la surcharge des routeurs BGP et du réseau lui-même, la perte de paquets... C'est ce que l'on appelle le Stable Path Problem (SPP).

Le SPP nous donne une vision simplifiée des problèmes d'instabilité des chemins dans les protocoles de routage comme BGP. Introduit par Griffin et al dans [7] , il permet de se focaliser sur les points essentiels causant les instabilités et de séparer les fonctionnalités de BGP qui ne jouent aucun role dans ce problème (par exemple le MED ou l'agrégation de routes,...).

Soit G=(V,E) un graphe tel que les éléments de V et les éléments de E représentent respectivement les systèmes autonomes et les liens BGP. Chaque AS détermine une liste de chemins ordonnés par ordre de préférence .

Une solution à SPP est un assignement d'un chemin pour chaque AS, tel que le rang de préférence de ces chemins soit le plus haut possible. Griffin et al. ont montré dans [8] que SPP est un problème NP-complet.

La figure 2.3 est une instance du problème SPP. Il s'agit d'un exemple classique d'oscillation appelé BAD GADGET très largement diffusé. Nous supposerons dans tout ce qui suit que les AS cherchent à choisir un chemin allant à la destination 1.

Voici une simulation possible du protocole BGP à partir de la figure 2.3 : au départ, tous les AS choisissent soit un chemin direct, soit le chemin vide noté ε. L'AS 2 ne connaît pas le choix des chemins de ses voisins. Il choisit le chemin 21 et le fait savoir à ses voisins. L'AS 3 prend connaissance de cette information et choisit 321 ; lui aussi fait suivre l'information à l'AS 4. La route 31 n'étant pas disponible, l'AS 4 choisit la route 41. Lorsque l'AS 2 en sera informé il pourra choisir la route 241 qu'il préfère provoquant la perte de 321,... Le processus ne s'arrêtera jamais. Cet exemple est une illustration d'une oscillation ou d'une instabilité du protocole BGP.

Des cas identiques d'oscillations ont déjà été détectés dans l'Internet et il est primordial d'implanter une méthode de résolution d'oscillation rapidement car avec l'explosion

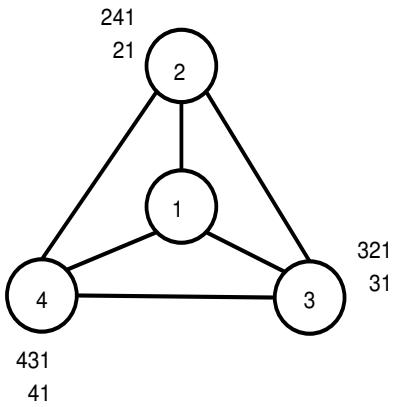

FIG. 2.3 – BAD GADGET. Problème sans solution stable.

du nombre de routeurs dans le monde et un traffic qu'on pourrait qualifier de fonction d'Ackermann, il est impossible de se contenter du routage Internet actuel.

Chapitre 3

Principe de la solution proposée dans [1]

3.1 Maintien de l'état des chemins

Voici le principe de la solution proposée dans [1]. Certaines solutions proposées s'appuyaient sur un historique ; malheureusement le principe de confidentialité n'était pas respecté et l'utilisation d'un historique nécessitait des ressources trop importantes sur un AS (voir [1] et [6]).

La solution proposée dans [1] se base sur une gestion locale, je vais maintenant vous décrire comment elle fonctionne.

Tous les chemins démarrent en ayant l'état '*'. Le principe général est, lorsqu'un nouveau chemin est sélectionné on affecte un nouvel état dans les cas suivants :

- Si le nouveau chemin sélectionné a une préférence locale supérieure à l'ancien alors son nouvel état est '+'.
- Si le nouveau chemin sélectionné a une préférence locale inférieure à l'ancien alors l'état de l'ancien chemin passe à '-'.

La méthode proposée dans [1] montre que si un chemin passe d'un êtat '+' à un état '-', alors c'est que ce chemin subit une oscillation.

Le tableau 3.1 représente une éxécution séquentielle du maintien d'êtat pour le BAD GADGET à 5 AS de la figure 3.1. Le champ rib-in correspond au chemin actuellement choisi pour un AS donné, les '+' et '-' dans les colonnes des chemins représentent les états correspondants au cours du temps. Ainsi, on peut savoir pour un chemin, s'il a permis d'améliorer le routage de l'AS par rapport à l'ancien chemin choisi ('+'), ou bien s'il a été abandonné au profit d'un chemin moins bon ('-'). Ce n'est pas représentatif de la réalité car normalement l'échange des messages se fait de façon asynchrone. Nous verrons par la suite comment gérer le cas asynchrone à l'aide d'un jeton.

3.2 Détection et Résolution des oscillations

La méthode proposée dans [1] nous permet maintenant de détecter correctement les chemins oscillants. Il est nécessaire maintenant d'identifier les chemins appartenant à un circuit et en interdire un pour le rompre.

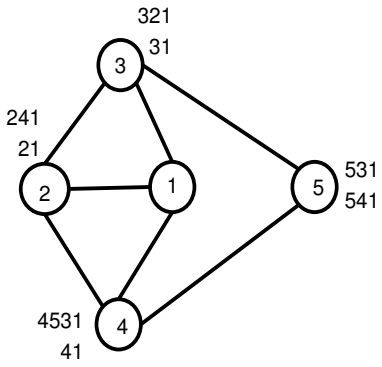

FIG. 3.1 – BAD GADGET composé de 5AS.

	AS2			AS3			AS4			AS5		
step	241	21	rib-in	321	31	rib-in	4531	41	rib-in	531	541	rib-in
1	*	*	21	*	*	31	*	*	4531	*	*	531
2	*	*	21	+	*	321	*	*	4531	*	*	531
3	*	*	21	+	*	321	*	*	4531	-	*	ϵ
4	*	*	21	+	*	321	-	*	41	-	*	ϵ
5	+	*	241	+	*	321	-	*	41	-	+	541
6	+	*	241	-	*	31	-	*	41	-	+	541
7	+	*	241	-	*	31	-	*	41	+	+	531
8	+	*	241	-	*	31	+	*	4531	+	+	531
9	-	*	21	-	*	31	+	*	4531	+	+	531

TAB. 3.1 – Exemple de maintien de l'état des chemins pour BAD GADGET.

La méthode retenue consiste à utiliser un jeton qui déterminera si un chemin, ayant subi une oscillation, appartient à un circuit oscillant. On peut noter que cette méthode n'ajoute pas de messages complémentaires mais complète les messages prévus par BGP (annonces d'un nouveau chemin) avec un jeton.

Nous allons maintenant décrire comment fonctionne cette méthode.

Un AS 'A' qui détecte une oscillation sur un de ses chemins 'X', grâce au maintien de l'état des chemins, génère un jeton, 'j.h(X)' (h étant une fonction de hachage), associé à ce chemin 'X' et l'expédie avec l'annonce de son nouveau chemin 'Y', en y ajoutant son identifiant d'AS (attribué par une autorité et unique dans l'Internet) à la liste des AS concernés par cette route et l'information OK qui indique que le jeton est prioritaire pour lui. Cette notion de priorité est basée arbitrairement sur la comparaison lexicographique entre deux jetons ; celui qui est inférieur pour cet ordre est désigné comme prioritaire.
Lorsque l'AS 'B' reçoit l'annonce d'un nouveau chemin avec un jeton, deux cas se présentent : si 'B' ne doit faire aucune modification suite à l'annonce il jette le jeton, sinon : tout dépend du jeton : s'il n'est pas prioritaire pour lui il désactivera le jeton en lui ajoutant l'information NOK, se rajoutera à la liste des AS concernés et retransmettra le jeton, sinon si 'B' modifie

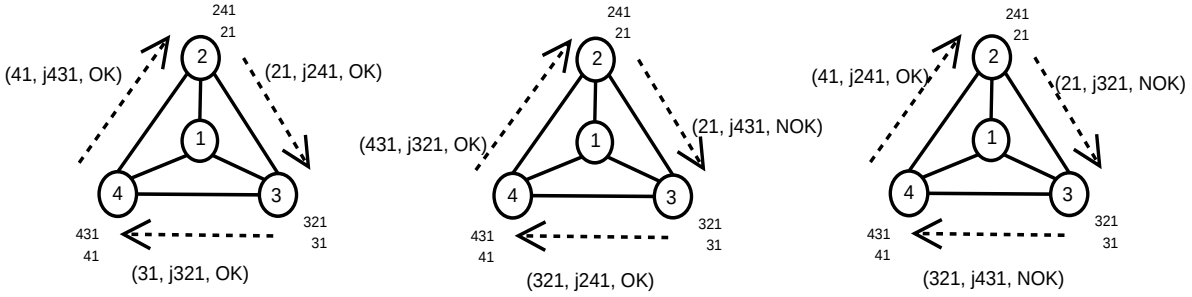

FIG. 3.2 – Echange de jeton dû à une oscillation du chemin 321.

son chemin suite à cette réception et que le jeton reçu est prioritaire, il transmettra son nouveau chemin (comme prévu par BGP) et fera suivre le jeton 'j.h(X)' en laissant le champ à OK et en se rajoutant aussi à la liste des AS concernés et ainsi de suite... Notons que si un AS reçoit un jeton ayant déjà l'information NOK alors l'AS fera suivre ce jeton mais ne modifira pas cette information ; il se rajoutera lui aussi à la liste des AS concernés.

Lorsque 'A' reçoit un message contenant le chemin 'Z' en provenance d'un de ses voisins, avec le jeton 'j.h(X)' associé. Si 'Z' entraine le choix de X et que le jeton reçu est toujours à OK alors l'AS 'A' peut conclure que 'X' (qui était à l'origine de la création du jeton j.h(X)) appartient à un circuit. Ainsi 'A' va interdire son chemin 'X' et garder les numeros des AS concernés par cette route dans une table associative. Dans les deux cas ("OK","NOK"), les AS générateurs de jetons, connaissent les identifiants de tous les AS impliqués dans la même oscillation (cela sera utile pour la gestion des pannes et apparitions de liens).

Remarque : L'utilisation d'une fonction de hachage qui affectera une valeur au jeton en fonction du chemin oscillant permet de ne pas dévoiler le chemin oscillant. Ainsi la confidentialité est respectée. Seul l'AS générateur du jeton, 'A', est capable de retrouver le chemin à l'origine de la création de ce jeton.

La figure 3.2 est un exemple de transmission de jeton dans le cas du BAD GADGET de la figure 3.3.

Voici le déroulement de cette transmission de jeton.

L'AS 3 détecte une oscillation du chemin 321, l'AS 4 sur le chemin 431 et l'AS 2 sur le chemin 241. L'AS 3 génère alors un jeton associé à ce chemin (j321) et envoie son nouveau chemin (31, j321, OK) en joignant le jeton. L'AS 2 et l'AS 4 font de même. A la réception du jeton j321, l'AS 4 modifie son chemin en adoptant 431, et fait donc suivre le jeton avec l'annonce (431, j321, OK). A la réception du jeton j431, l'AS 2 désactive le jeton et le retransmet avec l'annonce (21, j431, NOK). A la réception du jeton j241, l'AS 3 modifie son chemin en adoptant 321, et fait donc suivre le jeton avec l'annonce (321, j241, OK). Ainsi de suite, le jeton j321 sera aussi désactivé car non prioritaire pour l'AS 2 (son jeton est inférieur pour l'ordre lexicographique à celui de l'AS 2). Au final, 2 jetons auront été désactivés et il ne restera plus que le jeton j241. L'AS 2 reçoit alors le jeton qu'il avait précedemment émis, et le chemin associé entraine le choix de 241. Donc 241 est considéré comme un "mauvais chemin". L'interdiction de ce chemin aboutira à une solution.

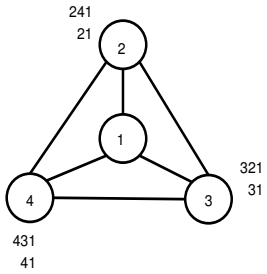

FIG. 3.3 – BAD GADGET.

Bilan et objectifs

Une fois que cette première phase, consistant a étudier le protocole BGP et la méthode de détection et de résolution des oscillations, a été bien assimilée, j'ai dù me pencher sur la compréhension du mécanisme de pannes et ajout de liens proposé dans [1] puis sur l'implémentation de ce mécanisme.

Dans les chapitres suivants, je vais vous présenter tout d'abord cette méthode de gestion des modifications topologiques, puis je vous présenterai le travail accompli et les résultats obtenus avant de brièvement traiter les difficultés que j'ai pu rencontrer.

Chapitre 4

Pannes et apparitions de liaisons entre routeurs BGP

4.1 Principe

BGP est un protocole autostable, c'est à dire que quelque soit la configuration dans laquelle se trouve le réseau, les AS vont s'échanger leurs chemins jusqu'à arriver à stabilité (si les politiques sont cohérentes entre elles, cf. chapitre sur BGP). Donc en cas de modification de la topologie, BGP converge vers un état stable.

Il existe deux sortes de pannes : les pannes de routeurs et les pannes de liaisons. Nous nous intéresserons à la gestion des pannes de liaisons car elle s'étend facilement à celle des pannes de routeurs.

Les apparitions des liens peuvent se produire lorsqu'un AS apparaît pour la première fois dans le réseau ou bien lorsqu'un lien est rétabli à la suite d'une panne.

Lorsqu'une panne de liaison ou une apparition d'un lien se produit, il est possible qu'un chemin préalablement interdit puisse être réhablité.

La gestion des pannes et apparitions de liaisons est similaire : tout d'abord les AS touchés par ce changement (par exemple les AS situés aux extrémités de la liaison en panne) vont détecter la panne, puis vont avertir tous les AS concernés (grace à la sauvegarde de leurs identifiants, voir chapitre sur le principe de la solution proposée) du changement sur ce lien. Si un AS possédant un chemin interdit reçoit cette information, il lance le processus de test de réhabilitation sur ce chemin afin de savoir s'il peut être réhabilité ou non.

4.2 Panne de liaison

Lorsqu'une panne de liaison se produit, les deux AS à l'extrémité de cette liaison vont s'en rendre compte. Si ces AS avaient généré un jeton suite à une oscillation, ils savent que l'interdiction du chemin qui avait eu lieu pour résoudre cette oscillation peut être maintenant illégitime. Malheureusement, BGP n'impose aucune annonce si aucun changement de route n'a lieu. Donc ces AS se doivent d'en informer les autres AS afin de réhabiliter éventuellement ce chemin. C'est l'AS possédant le plus petit identifiant entre les deux qui se chargera d'annoncer cette information (ce choix est totalement arbitraire, il permet de transmettre une seule fois l'information). Grâce aux identifiants joints avec le jeton, il connaît tous les AS impliqués et les prévient en leur envoyant un message les informant de la panne. Les

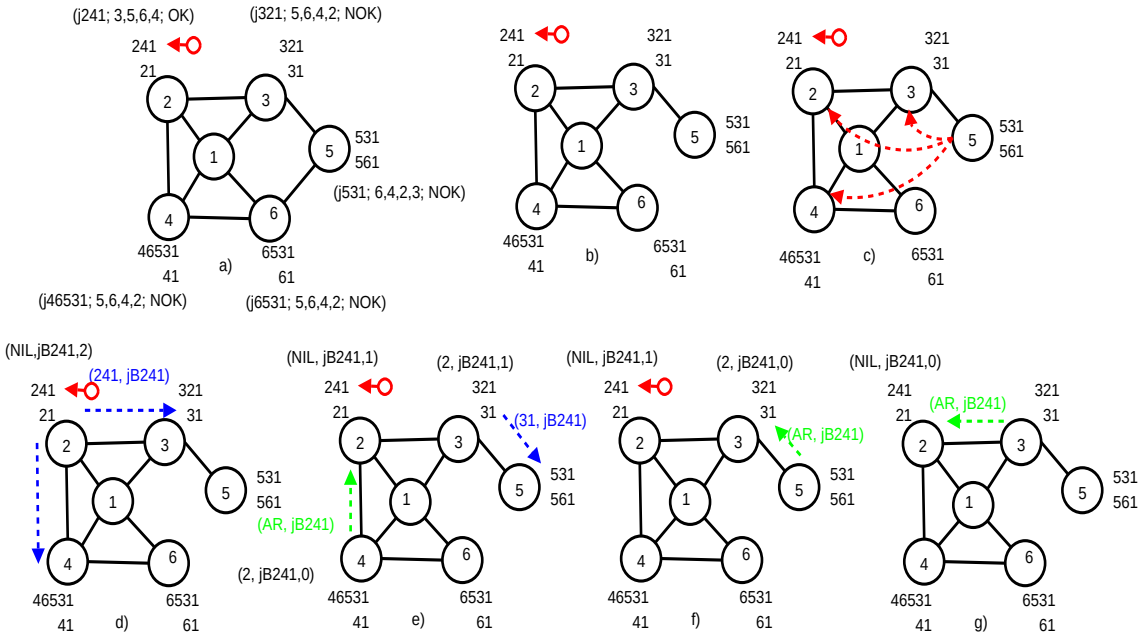

FIG. 4.1 – Illustration d'une panne dans un BAD GADGET à 6 sommets

AS, possédant des chemins interdits, vont lancer le processus de test de réhabilitation sur ces chemins et réhabiliteront ceux ayant réussi le test.

Le processus de test consiste à tester si un chemin peut être réhabilité ou non en simulant sa réhabilitation. Il génère un jeton blanc et l'envoie avec sa route à tester. Lorsque un AS reçoit un message contenant un jeton blanc, il ne met pas sa table de routage à jour mais simule un nouveau choix de route dû à l'arrivée d'un message de mise à jour de route. Dans ce cas si sa route n'est pas modifiée, il acquite le message reçu, si au contraire elle est modifiée, il retransmet le jeton avec sa "nouvelle route" simulée. Lors du parcours du jeton blanc il n'y a jamais de modification effective de route. Lors de cette simulation, si aucune oscillation n'est générée, le processus de test a réussi et le chemin est réhabilité.
Prenons l'exemple de la figure 4.1 a). Il s'agit d'un BAD GADGET (exemple de réseau provoquant des oscillations du protocole) avec six AS où la méthode a abouti à l'interdiction du chemin 241. Supposons que le lien reliant les AS 5 et 6 tombe en panne (cf. figure 4.1 b)).

Lorsque les AS 5 et 6 prennent connaissance de la panne, l'AS 5 avait généré un jeton dû à l'oscillation du chemin 531. Il annonce l'information de la panne à tous les AS qui étaient impliqués dans cette oscillation (cf. figure 4.1 c)). Les AS 3 et 4 ne possèdent pas de chemin interdit. Par contre l'AS 2 lance le processus de test de réhabilition sur le chemin 241 : l'AS 2 génère un jeton blanc associé à 241 ($jB241$), envoie ce jeton avec le chemin 241 (241,$jB241$) (cf. figure 4.1 d)) et conserve l'information $(NIL, jB241, 2)$ où le premier champ est l'AS "père", le deuxième contient le jeton et le troisième le nombre de liaisons auxquelles on a transmis le jeton. L'AS 3 et 4 reçoivent cette information, se rendent compte qu'il s'agit d'une simulation grâce au jeton blanc et regardent si le chemin 241 les feraient changer de chemin. L'AS 4 n'aurait aucune modification à faire et par conséquent, envoie un accusé de

réception à l'AS 2 qui décrémente son compteur de messages envoyés. Par contre, l'AS 3 a adopté le chemin 321 donc il modifierait son chemin en choisissant 31. Il fait suivre le jeton blanc avec le chemin 31 (31,$jB241$) (cf. figure 4.1 e)) et conserve l'information $(2, jB241, 1)$. L'AS 5 reçoit cette information et pourrait ainsi prendre 531 mais il n'a plus de voisin donc il envoie un accusé de réception à l'AS 3 qui décrémente son compteur valant maintenant zéro ((cf. figure 4.1 f)). Lui aussi envoie un accusé de réception à l'AS 2 qui décrémente son compteur et réhabilite la route 241 puisque le compteur vaut zéro. De cette manière le système, aboutira aux choix des chemins suivants : (241 31 41 531 61). Si le jeton était revenu à l'AS 2, le chemin n'aurait pas été réhabilité.

4.3 Apparitions ou rétablissements de liens

Intéressons nous maintenant aux cas des apparitions de liens. Si un AS A, ayant généré un jeton pour un chemin X lors d'une oscillation précédente, est ammené grâce à l'apparition d'un nouveau lien, à choisir un chemin Y. Si Y est préféré à X, il sait que l'interdiction qui s'était produite pour résoudre cette oscillation n'est peut être plus légitime. Il utilise le processus d'annonce décrit précédemment pour avertir tous les AS concernés par l'oscillation. La suite du déroulement est identique à celle de la panne de liaison.

La détection d'une panne est décrite par l'algorithme 5 et celle d'une apparition de lien par l'algorithme 6. La réception d'une annonce est décrite par l'algorithme 7. Ces algorithmes se trouvent en annexe.

Chapitre 5

Travail effectué et Résultats

5.1 Objectifs

Le but de mon stage était de réaliser la deuxième phase de l'implémentation de la méthode de résolution des oscillations externes dans BGP proposée dans [1] (concernant les pannes et apparitions de liens). Cette méthode se décompose en plusieurs parties :

- Mise en place de la méthode de maintien d'état des liens (nécessaire à la détection d'oscillations)
- Ajout des jetons pour les messages de mise à jour de routes
- Gestion de la détection d'oscillation et interdiction de chemin sur retour de vague
- Mise en place d'une méthode de récupération des identifiants des AS concernés pour une oscillation donnée
- Mise en place d'un mécanisme de détection de panne et d'ajout de liaison
- Annonce de la détection de modification topologique
- Gestion des jetons blancs et du test de réhabilitation d'un chemin
- Annonce de jeton blanc sur réception d'une annonce de modification topologique

Julien Champ (voir [2]) avait effectué les trois premiers items en les implémentant sur le simulateur open-source C-BGP. L'interdiction d'un chemin était donc fonctionnelle, il me fallait donc gérer tout le reste, c'est à dire toute la partie d'adaptation aux pannes et ajouts de liens. Dans mon implémentation de la méthode, deux solutions s'offraient à moi :

- Soit reprendre le travail déjà fait sur C-BGP
- Soit tout reprendre à zéro et opter pour un autre simulateur de BGP

Il a donc fallu dans un premier temps que j'opte pour une de ces options.
Actuellement, seuls deux simulateurs sont disponibles :

- C-BGP, simulateur abouti et maintenu à jour par Bruno Quoitin
- Un autre simulateur en cours de développement, basé sur le simulateur de réseau OMNET++ et développé par le laboratoire PRISM de Paris

Après avoir rencontré le développeur du deuxième simulateur, j'ai pu voir que ce projet n'était pas suffisamment abouti pour que je commence mon implémentation sur ce support. Mon choix s'est donc porté vers l'utilisation de C-BGP et du travail déjà effectué, cependant il me semble intéressant pour le futur d'implanter la solution de [1] dans le simulateur du laboratoire PRISM lorsqu'il sera terminé. En effet, ce simulateur a l'avantage d'être plus complet et plus interactif que C-BGP.
Je vais donc maintenant présenter mon travail, à savoir la mise en place dans le simulateur

C-BGP des autres items et la terminaison de l'implémentation de la méthode de [1].

5.2 Travail effectué

5.2.1 Partie compréhension et apprentissage

Le stage a tout d'abord débuté par une grosse partie théorique de compréhension de BGP et de son fonctionnement interne ainsi que la méthode de détection et de résolution des oscillations, base du travail de Julien Champ (voir [2]). Cela m'a pris un certain temps car bien que connaissant BGP dans les grandes lignes je n'avais pas suffisamment de connaissances pour me lancer dans la partie suivante.
J'ai donc étudié le fonctionnement global de BGP : de l'annonce de routes à ses voisins pour un AS à la mise à jour de ses propres routes. Ce fonctionnement global est très simple cependant plus on entre dans les méandres du protocole, plus celui-ci devient compliqué et fait appel à des notions que je ne connaissais pas forcément. Ainsi j'ai dû me documenter sur les échanges de messages BGP selon leurs différents types, à quel moments étaient-ils envoyés...
Le problème majeur ici fut de trouver des informations claires sur ce que je voulais savoir : en effet BGP étant très complexe, il est difficile de trouver sur Internet des explications simplifiées. J'ai donc dû chercher de telles explications par moi même dans les livres avant de me lancer dans des documents plus techniques.
Ensuite j'ai dû me pencher sur le fonctionnement du simulateur C-BGP ; là mes connaissances dans le domaine Gnu/Linux me furent très précieuses car son installation n'est pas forcément évidente. Pendant cette phase, le rapport de Julien Champ [2] me fut très précieux car il expliquait bien comment prendre en main C-BGP, puis lorsque je me suis senti assez à l'aise, le manuel d'utilisation de C-BGP fut très utile (voir [9]). Dans cette partie, je suis entré en contact avec le concepteur de C-BGP : Bruno Quoitin ; celui-ci m'a énormément aidé à en comprendre le fonctionnement et m'a donné de précieux conseils qui m'ont permis de gagner un temps précieux en recherche et compréhension. Une fois que cela a été maîtrisé et que le fonctionnement du simulateur me fut familier, j'ai dû m'attaquer à la compréhension des méthodes de gestion des changements de topologies. Avec le travail de recherche que j'avais effectué sur le fonctionnement de BGP, cela a été assez facile à appréhender. De plus cette partie étant basée sur l'algorithmique distribuée, mes connaissances dans ce domaine m'ont été précieuses pour la compréhension de ces algorithmes et cela en a accéléré l'assimilation.
Ensuite j'ai adapté l'algorithme de gestion des réceptions de jetons de [1] ainsi que celui d'envoi de jetons car seule les premières versions de ces algorithmes (sans la notion de topologies dynamiques) étaient spécifiées ; puis nous avons vérifié, mon tuteur de stage et moi, la validité de ces nouveaux algorithmes.
Enfin j'ai réfléchi à la manière dont j'allais gérer le temps qui me restait pour implémenter cette méthode et j'ai pu continuer sur la partie pratique du stage : la programmation des algorithmes de gestion de changement de topologie.
J'ai suivi, assez logiquement, l'ordre dans lequel cette méthode est décrite dans [1].
Ainsi, c'est plus la compréhension du protocole BGP et la prise en main du simulateur qui me posèrent problème, les algorithmes de [1] étant assez simples lorsque l'on connait l'Algorithmique Distribuée.

5.2.2 Partie pratique

Une fois la partie théorique terminée et les fondements de mon travail posés, je me suis attaqué à la partie purement pratique : la mise en place et la programmation des algorithmes étudiés.
Pour implémenter tous les algorithmes de la méthode de [1] et tous les mécanismes correspondants il a fallu modifier le simulateur lui-même. Il m'a été très difficile, au début, de savoir où modifier le code pour pouvoir y incorporer les nouveaux algorithmes. En effet, avec plus de 130 fichiers C et plus de 53000 lignes de code, C-BGP est une application assez volumineuse. Bien que la structuration de ce programme soit bien faite, il parait évident qu'avec de telles dimension, il est difficile, au début, de se situer dans le code.
Ainsi, avant tout, il m'a donc fallu me plonger dans tout ce code C qui m'était inconnu (et d'où les commentaires sur les methodes et fonctions étaient quasi-absents). Cela m'a évidement pris un certain temps mais finalement me permis d'avancer bien plus rapidement par la suite.
Dès que j'ai commencé à comprendre à quoi correspondaient les fichiers et fonctions principales, je me suis lancé dans la partie programmation.
La toute première chose que j'ai dû faire alors a été de modifier le code existant pour l'adapter aux changements que j'allais lui faire subir, ainsi j'ai rajouté la notion d'AS impliqués par une oscillation et ai mis en place la table associative correspondante au niveau des AS. Ce mécanisme permet aux AS qui ont initié un jeton de connaitre tous les AS qui sont impliqués dans cette oscillation. J'ai rajouté dans les messages de mise à jour de routes (messages BGP 'UPDATE') la notion de priorité : 'OK' ou 'NOK' (au départ il n'y avait pas de telle notion et au lieu de désactiver un jeton on jetait le message) en plus de cette liste, car comme je l'ai expliqué plus haut, les algorithmes de vagues sur de tels messages n'étaient pas adaptés aux pannes et ajouts. Ainsi j'ai dû non seulement modifier les messages échangés mais aussi la manière dont ils étaient traités pour qu'ils respectent les nouveaux algorithmes.
Ensuite il m'a fallu implanter des messages de types bien particulier : les jetons blancs, nécessaires au test de réhabilitation d'une route après une panne ou un ajout de lien. Il a donc fallu modifier ici complètement le protocole et lui rajouter un nouveau type de message, ainsi il a évidement fallu modifier aussi tout ce qui avait attrait à l'envoi et la réception de messages.
J'ai créé les méthodes de test de panne de lien ou d'ajout de lien au niveau routeur afin de pouvoir détecter ces changements, ainsi que les annonces correspondantes aux autres AS. Ici aussi il a fallu modifier le protocole, mais comme j'avais vu auparavant en détail comment m'y prendre, cette partie fut plus aisée à programmer que la précédente.
La partie la plus difficile mais aussi la plus intéressante fut la mise en place des algorithmes distribués de gestion des vagues de messages : réception de jetons blancs et de messages de modification topologique qu'il fallait traiter de manière différente selon les cas. Cette partie m'a beaucoup plu car la mise en place d'algorithmes distribués est toujours à la fois simple dans le principe mais très compliquée dans la réalisation. Il a donc fallu que je réfléchisse au développement de ces algorithmes d'une manière différente par rapport à un algorithme "classique" et ceci est extrèmement plaisant et formateur en expérience.
Les algorithmes de cette partie distribuée sont présentés en annexe.

A quelques jours de l'échéance du stage, nous sommes en mesure de dresser un premier

bilan avec les résultats obtenus, afin de les comparer avec les objectifs initialement fixés.

5.3 Résultats

5.3.1 Récuperation des identifiants des AS concernés par une oscillation

Comme nous l'avons vu précédemment, lors du transit des jetons d'élimination d'oscillation, on collecte les numéros des AS concernés par cette oscillation dans le corps du message lui-même. Cette partie fonctionne parfaitement et des tests nous ont montrés que lorsqu'un AS reçoit le retour de sa vague, le message reçu contient bien toutes les adresses des AS qui ont retransmit le jeton. Ces adresses sont ensuite stoquées dans une table pour permettre d'avertir ces AS en cas de détection de panne. L'implémentation de cette partie fut aisée, il a suffit de rajouter un attribut au corps des messages BGP échangés : les AS concernés par l'oscillation correspondante au jeton. Ensuite, chaque AS qui retransmet ce jeton retransmet aussi la liste après s'y être rajouté.

5.3.2 Détection de Panne et d'ajout de liaison

La détection de panne se fait à l'envoi de message : lorsqu'un AS désire envoyer un message à un pair qui était connecté, si on détecte que la liaison est rompue on enclenche le processus d'avertissement. C'est ici que la liste d'AS concernés est utile : on avertit à tous les AS susceptibles de modifier leurs routes après la modification du graphe. La détection d'ajout, elle, est forcée et est appellée par le script-scénario (fichier .cli) par l'intermédiaire de la commande detecte_ajout_lien. J'ai du rajouter cette commande et traiter l'ajout ainsi car C-BGP n'étant pas "event-driven" (dirigé par évennements), il m'était impossible de lancer un test périodique d'état de liaison ou de détection de modification : il n'y a pas de notion de temps dans C-BGP ni d'ordonnancement d'évenements (sauf pour les arrivées de messages). Les tests nous ont montrés que cette partie fonctionne correctement.

5.3.3 Annonce de la détection

Lorsqu'un AS détecte une panne ou un ajout, il annonce à tous les AS concernés cette modification par un message de type spécial. Lorsqu'un AS réceptionne un tel message il va tester la réhabilitation de ses routes précédemment interdites. Après divers tests, les résultats sont cohérents et me permettent de dire que cette partie est fonctionelle.

5.3.4 Jetons blancs

Lors du processus de test de réhabilitation d'une route par un AS, celui-ci va émettre un jeton blanc. Le processus de réhabilitation est lancé après réception d'un message de modification topologique par un AS, il va enclencher une simulation de réhabilitation des routes interdites de cet AS. Ce processus de test créé un jeton blanc associé à la route à tester. Ce jeton blanc est émis à tous les voisins de l'AS. Son trajet suit un processus de vague et est donc diffusé par inondation. On est certain ainsi d'atteindre tout les AS concernés. Les AS font suivre ou acquitent la vague selon l'algorithme 4 présenté en annexe et la méthode de test de réhabilitation présentée dans le chapitre consacré aux pannes. Lorsque l'AS ayant initié la vague reçoit son dernier accusé de réception, on est certain de pouvoir réhabiliter la route ; si jamais cet AS reçoit son propre jeton blanc et que celui-ci entraine le choix d'un

autre chemin que celui à tester, alors on ne doit pas réhabiliter le chemin.
Cette méthode a pour but de ne pas réhabiliter trop tôt un chemin qui pourrait engendrer une oscillation ; avec l'implémentation de cette méthode, on est certain de ne pas recréer d'oscillation après une panne ou un ajout.
Le protocole d'émission et réception du jeton blanc est clairement un protocole distribué, c'est ce qui en fait la simplicité mais aussi la difficulté à vérifier ; c'est pour cela que j'ai dû effectuer toute une série de tests sur diverses topologies. Les tests effectués nous ont montrés que les messages échangés respectent bien le protocole et que, au final, l'AS qui a initié le test de réhabilitation, réhabilite ou non sa route en fonction des réponses reçues. S'il y a réhabilitation, l'AS diffuse bien évidement cette modification aux autres AS.
Cette partie fonctionne donc correctement.

5.3.5 Tests

Après une série de tests sur diverses topologies de réseaux et plusieurs exemples de BAD GADGET connus, les tests sont concluants et les résultats concordent avec ceux attendus. La solution proposée semble donc bien fonctionner, il faudrait maintenant la tester sur des topologies générées aléatoirement et en masse pour se faire une idée plus précise de son exactitude.

Chapitre 6

Difficultés

Comme dans tout projet, j'ai rencontré divers problèmes et difficultés tout au long de mon travail sur la méthode de résolution des oscillations. Que ce soit des problèmes de compréhension du protocole, ou des problèmes lors de l'implémentation de la méthode de [1].

Je ne vais cependant pas vous énumerer tous les problèmes auxquels j'ai dû faire face, mais principalement ceux qui m'ont demandé du temps.

6.1 De l'étude...

Durant la première partie du stage, je me suis penché sur le fonctionnement du protocole BGP.

La lecture des mémoires de [1] et [6] ainsi que celle du rapport de [2] m'a permis dans un premier temps de me faire une bonne idée du fonctionnement de BGP.

Ces deux mémoires abordant le protocole d'un point de vue théorique il m'a fallu alors bien comprendre le fonctionnement de BGP dans la pratique, c'est en cela que le rapport de stage de Julien m'a beaucoup aidé.

Bien qu'on puisse résumer assez simplement le fonctionnement de BGP, ce protocole n'est pas simple à prendre en main car il existe un grand nombre de configurations possibles. Tout le protocole n'entrant pas en jeu dans le phénomène des oscillations il n'a donc pas été nécessaire que je me penche sur l'intégralité du protocole mais j'ai tout de même dû en assimiler une partie importante.

BGP n'étant pas un protocole connu du "grand public" il n'est pas facile de trouver des informations ou de l'aide sur un point précis en cas de doute sur le fonctionnement du protocole.

6.2 ...à l'implémentation

Lorsque j'ai commencé mon travail d'implémentation de la méthode, je me suis confronté à un problème. C-BGP étant un simulateur se voulant le plus complet possible, le code source du programme commence à être assez important. Bien que le code soit un minimum commenté et que les noms de variable et de fonction soient bien choisis, il n'empêche que se plonger dans le code d'une autre personne n'est pas évident. D'autant plus que ce code

avait déjà été modifié par une tierce personne et Bruno Quoitin, le concepteur de C-BGP ne pouvait donc pas répondre à mes questions sur cette partie.

Il a fallu que je comprenne comment le développeur avait découpé le code source dans différents fichiers, et répertoires, mais aussi, pour effectuer les modifications souhaitées que je comprenne comment fonctionnait le logiciel en "interne" et notamment l'architecture globale du programme, ce qui n'a pas été chose facile.

J'ai mis un peu de temps à m'adapter à ce code source donc, mais après un certain temps d'adaptation le code a commencé à être clair et l'implémentation est devenue subitement bien plus aisée.

Conclusion et Perspectives

Dans le présent rapport, je vous ai présenté une partie de mon travail lors de mon stage au sein du LIRMM. Ce stage avait pour but d'implémenter la méthode de résolution mise au point dans [1] dans un simulateur de réseau BGP.

Le travail que j'ai dû réaliser, m'a permis de mettre en pratique ce que j'ai appris durant mon cursus universitaire ; de l'étude de la faisabilité et d'ordonnancement au travail de développement en passant par la gestion du temps et surtout m'a permis de voir toutes les implications de l'algorithmique distribuée, matière enseignée cette année dans le cursus RICM2.

J'ai ainsi dans un premier temps dû étudier le protocole BGP et ensuite implémenter la deuxième partie de la méthode de résolution des oscillations dans le simulateur open-source C-BGP. J'ai effectué une petite série de tests, suite à l'implémentation, qui permettent de dire que cette méthode semble valide.

A titre personnel, ce stage m'a été fort bénéfique sur divers points.

J'ai pu acquérir diverses compétences informatiques, telles que la connaissance du protocole BGP, ou le perfectionnement de ma connaissance du langage C. Mon travail sur un projet de taille conséquente, C-BGP, m'a permis de me familiariser au travail et à la compréhension du code d'une autre personne.

Je souhaitais découvrir le monde de la recherche au sein d'un laboratoire, de préférence public, afin de pouvoir comparer avec l'expérience que j'avais du monde professionnel. Ce stage m'a aussi permis de découvrir et de participer à la vie sociale dans un laboratoire, où j'ai pu rencontrer et discuter avec des doctorants et des chercheurs sur divers sujets.

De plus la lecture de mémoires, tels que [1] et [6], mais aussi d'articles m'a permis de mieux comprendre les attentes dans le monde de la recherche.

En définitive on peut voir que la méthode proposée dans [1] semble fonctionner et qu'il reste à la tester sur de très grandes topologies se rapprochant de la réalité de l'Internet. Cependant cette méthode ne traite que les oscillations externes aux AS, une deuxième solution proposée dans [6] traite les oscillations internes à un même AS et il serait ainsi interessant d'implémenter ces deux solutions dans un même simulateur de BGP pour en étudier le comportement. Il me semble que les tests et la mise en place de telles solutions sont primordiales pour l'avenir de l'Internet, et les résultats que nous avons eu jusqu'à maintenant me semblent suffisamment encourageants pour continuer nos efforts dans cette voie.

Annexe

C-BGP : Présentation

C-BGP est un simulateur BGP open-source dévelloppé par l'Université catholique de Louvain et plus particulièrement par Bruno Quoitin. Il simule le comportement des AS, le processus de sélection de chemins, ainsi que les politiques des AS. C-BGP est écrit dans le langage C, et a été testé sur diverses plateformes (Gnu/Linux, FreeBSD, Solaris...).

Utilisation de C-BGP

Afin d'utiliser C-BGP il faut tout d'abord créer un réseau dans lequel la simulation va se dérouler.

Pour cela, on peut soit définir les AS, routeurs, et liens un à un dans le shell de commandes de C-BGP, soit on peut utiliser un fichier contenant la topologie du réseau, et ensuite un deuxième fichier dans lequel nous règlerons les préférences de chaque AS ainsi que les paramètres nécessaires à la simulation.

Afin de comprendre le fonctionnement du simulateur, je vais vous décrire comment créer un BAD GADGET composé de quatre AS tel que nous l'avons décrit précédemment.

Topologie du réseau (fichier .topo)

Ce fichier décrit la topologie du réseau. Il permet de définir rapidement les AS du réseau, ainsi que les liens entre chacun d'entre eux.

Chaque ligne de ce fichier sera composée ainsi : "ASX ASY REL".

ASX et ASY correspondent à un numéro d'AS et REL à la relation entre les précédents AS.

Si REL vaut 0 cela signifie que c'est une relation de pair-à-pair.

SI REL vaut 1 cela signifie que c'est une relation Fournisseur-Client (ASX est le fournisseur, ASY le client).

(voir les définitions de pair-à-pair et Fournisseur-Client dans la partie sur BGP)

Afin de créer la topologie du réseau *BAD GADGET* voici le fichier badgadget.topo correspondant :

```
1 2 0
1 3 0
2 3 0
1 4 0
```

```
2 4 0
3 4 0
```

La figure 6.1 correspond à l'interprétation que C-BGP fera du fichier de topologie.

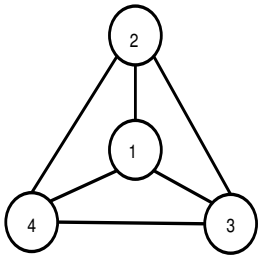

FIG. 6.1 – Interprétation de la topologie par C-BGP.

Maintenant que la topologie du réseau est décrite, il reste à créer le fichier de script pour faire une simulation avec C-BGP.

Script C-BGP (fichier .cli)

Ce fichier permet de définir le comportement du simulateur, de charger une topologie, de fixer les politiques des AS,...

Nous allons décrire les commandes principalement utilisées, pour plus de détails vous pouvez vous référer au manuel [9].

Premièrement, afin d'utiliser une topologie existante, il suffit de la charger au lancement du simulateur :

```
bgp topology load "badgadget.topo"
```

Ainsi le simulateur charge la topologie décrite dans le fichier "badgadget.topo". C-BGP construit un réseau ou chaque domaine est composé d'un unique routeur. Pour tout numéro d'AS, C-BGP attribue une adresse IP. Par exemple :

- L'AS 1 sera composé du routeur ayant pour adresse IP 0.1.0.0
- L'AS 250 sera composé du routeur ayant pour adresse IP 0.250.0.0
- L'AS 7018 sera composé du routeur ayant pour adresse IP 27.106.0.0

C-BGP configure aussi les sessions BGP entre les routeurs.

Le chargement de "badgadget.topo" aura donc pour effet de créer le réseau vu sur la figure 6.1.

Maintenant que le réseau du BAD GADGET est correctement créé, il ne reste plus qu'à fixer les politiques locales.

Les politiques de chaque AS peuvent être définies selon un grand nombre de critères, pour de plus amples renseignements vous pouvez vous référer à la section "Filters" de [9].

Nous allons voir comment définir les préférences locales d'un AS du BAD GADGET, il suffit de procéder de manière similaire pour définir les autres AS.

Prenons par exemple l'AS 2.

Pour cet AS nous voulons accorder une préférence plus haute au chemin **241** qu'au chemin "direct" **21**.

Voici comment procéder sur le routeur "0.2.0.0", correspondant à l'AS 2, pour fixer la préférence sur la route **241** à 66 :

```
bgp router 0.2.0.0
        peer 0.4.0.0
                filter in
                        add-rule
                                match "prefix is 0.1/16"
                                match "path \"^4 1$\""
                                action "local-pref 66"
                        exit
```

Maintenant, sur le même routeur "0.2.0.0" nous fixons la préférence de la route **21** à 33 :

```
bgp router 0.2.0.0
        peer 0.1.0.0
                filter in
                        add-rule
                                match "prefix is 0.1/16"
                                match "path \"^1$\""
                                action "local-pref 33"
                        exit
```

Ces quelques lignes pour définir les préférences locales auront donc pour effet sur l'AS 2 que son processus de sélection choisira de préférence le chemin 241 au chemin "direct" 21.

(les valeurs des préférences locales auraient pu être autres que 66 et 33, du moment que le chemin 241 a une préférence locale supérieure au chemin 21)

Remarque : Le chemin dont on veut fixer la préférence locale est défini à l'aide d'une expression régulière.

Pour finir de fixer correctement les préférences locales il suffit de faire la même chose sur l'AS 3 et l'AS 4.

Maintenant que le réseau est correctement décrit, et que les préférences locales sont fixées, la simulation va pouvoir être lancée.

```
bgp topology run
```

Cette commande permet d'établir les sessions BGP entre tous les routeurs définis dans le fichier de topologie(cette commande doit donc être utilisée si on a précédemment utilisé "bgp topology load").

```
sim run
```

Cette commande lance la simulation, qui ne s'arrêtera que lorsque il n'y aura plus d'élémements à traiter. A la fin de simulation, il est possible de demander au simulateur d'afficher

les tables RIB (vues dans la section traitant de BGP).

Celle qui nous intéresse principalement est la table RIB-IN. Elle contient la liste des chemins éligibles lors du processus de sélection du meilleur chemin.

A la fin de la simulation du BAD GADGET (après avoir implémenté la méthode pour résoudre les oscillations), nous pouvons par exemple demander l'affichage de la table RIB-IN de l'AS 2 pour tous les chemins à destination de l'AS 1.

```
bgp router 0.2.0.0 show rib-in * 0.1.0.0
```

Voila la table RIB-in résultante de la commande précédente, à la suite d'une simulation qui a été arrétée par la commande "sim stop at X" (arret de la simulation apres X passages dans le processus de séléction), sachant que la ligne qui commence par "*>" permet de connaître le chemin sélectionné à l'arrêt de la simulation :

```
----- Rib in AS2 -----
*  0.1.0.0/16   0.1.0.0 33      4294967295      1       i
*> 0.1.0.0/16   0.3.0.0 66      4294967295      4 1     i
*  0.1.0.0/16   0.4.0.0 0       4294967295      3 1     i
```

L'essentiel à retenir pour nous est que cette table nous permet d'apprendre que c'est le chemin **241** qui est choisi par l'AS 2 au moment ou le simulateur a été arrété.

Les détails de l'affichage résultant de la commande précédente sont traités dans la section "Commands Reference" de [9].

Pannes et Ajout de liaisons

Pannes

Exemple de panne de lien bidirectionnel entre les routeurs 2 et 3 :

```
net link 0.3.0.0 0.2.0.0 down
net link 0.2.0.0 0.3.0.0 down
net node 0.3.0.0 spf-prefix 0.3/16
net node 0.2.0.0 spf-prefix 0.2/16
bgp domain 3 rescan
bgp domain 2 rescan
sim run
```

On déclare que le lien entre les adresses 0.3.0.0 et 0.2.0.0 voit son etat passer à "DOWN". Puis comme avant on déclare le prefixe des adresses modifiées et on met à jour leur domaine. Enfin on relance la simulation.

Ajouts

Exemple de rétablissement de lien bidirectionnel entre les routeurs 2 et 3 :

```
net link 0.3.0.0 0.2.0.0 up
net link 0.2.0.0 0.3.0.0 up
net node 0.3.0.0 spf-prefix 0.3/16
net node 0.2.0.0 spf-prefix 0.2/16
bgp domain 2 detecte_ajout_lien
bgp domain 3 detecte_ajout_lien
bgp domain 3 rescan
bgp domain 2 rescan
sim run
```

On déclare que le lien entre les adresses 0.3.0.0 et 0.2.0.0 voit son etat passer à "UP". On appelle la methode detecte_ajout_lien qui force la detection du nouveau lien auprès des routeurs concernés. Puis comme avant on déclare le prefixe des adresses modifiées et on met à jour leur domaine. Enfin on relance la simulation.

Algorithm 1: Détection d'une oscillation pour un AS u (modifié et adapté à la gestion des pannes)

Data : Tableau T de l'état des chemins
Result: Génère un jeton en cas d'oscillation et le diffuse aux voisins

```
/* detectOscillation(T) retourne le chemin oscillant */
/* demandeur = vrai si u a généré un jeton, faux sinon */
/* creatToken(op) créer un jeton en fonction du chemin oscillant op */
/* choicePath() retourne le meilleur chemin selon la politique de l'AS */

demandeur ← false;
oscillantPath ← detectOscillation(T);
if oscillantPath ≠ null then
    demandeur ← true;
    token ← creatToken(oscillantPath);
    token.addPath(u);
    token.type ← OK;
    newPath ← choicePath();
    for v ∈ N(u) do
        send(newPath, token, v);
    end
end
```

Algorithm 2: Réception d'un jeton pour un AS u (modifié et adapté à la gestion des pannes)

Data : réception du message $<oscillantPath, token, v>$
Result: Fait suivre ou supprime le jeton reçu

```
/* path : chemin actuellement choisi */
/* getGenerator() retourne le chemin à l'origine de la création du jeton */
/* interdire(p) interdit le chemin p */
/* lgPath(t) retourne la longueur du chemin associé au jeton t */

newPath ← choicePath();
if path ≠ newPath then
    path ← newPath;
    /* token est prioritaire par rapport à ≺ ou la longeur du chemin associé à token
    est plus petite que la longueur du chemin associé à myToken */
    if demandeur = vrai and
    (token ≺ myToken or lgPath(token) < lgPath(myToken)) and
    (type(token)! = NOK) then
        demandeur ← false;
    end
    if demandeur = vrai and
    not(token ≺ myToken or lgPath(token) < lgPath(myToken)) and
    (type(token)! = NOK) then
        token.type ← NOK;
    end
    if demandeur = vrai and token = myToken then
        if path = myToken.getGenerator() then
            if token.type = OK then
                interdire(path);
                sauverNumerosAsImpliques(path);
                path ← choicePath();
                for v ∈ N(u) do
                    send(path, v);
                end
            end
            else
                sauverNumerosAsImpliques(path);
            end
        end
    end
    else
        for w ∈ N(u) − {v} do
            token.addPath(u);
            send(newPath, token, w);
        end
    end
end
```

Algorithm 3: Processus de test de réhabilitation

Data : Un chemin *barredPath*
Result: Retourne vrai si *barredPath* peut être réhabilité, faux sinon
/* creatWhiteToken(op) créer un jeton en fonction du chemin oscillant op */
/* involve(p) retourne le chemin choisi si u connait p */

$whiteToken \leftarrow creatWhiteToken(barredPath)$;
$d \leftarrow |N(u)|$;
for $v \in N(u)$ **do**
  $send(barredPath, whiteToken, v)$;
end
while $d \neq 0$ **do**
  $reception < message >$;
  /*u récupère son jeton et le chemin associé empêche l'adoption de barredPath*/
  if $< message >=< whiteToken, p >$ **and** $involve(p) \neq barredPath$ **then**
    $returnfalse$;
  end
  /*u reçoit un accusé réception*/
  if $< message >=< whiteToken, AR >$ **then**
    $d \leftarrow d - 1$;
  end
end
$returntrue$;

Algorithm 4: Réception du jeton blanc

Data : réception du message <p, $whiteToken$, v>
Result: Fait suivre ou non le jeton blanc
/* *currentPath* représente le chemin actuellement choisie */
/* annonceur(jB) retourne l'identifiant de l'annonceur de jB */

$path \leftarrow involve(p)$;
if $path \neq currentPath$ **then**
  $d \leftarrow |N(u) - \{v\}|$;
  for $w \in N(u) - \{v\}$ **do**
    $send(path, whiteToken, w)$;
  end
  while $d \neq 0$ **do**
    $reception < whiteToken, AR >$;
    $d \leftarrow d - 1$;
  end
  $send(whiteToken, AR, annonceur(whiteToken))$;
end
else
  $send(whiteToken, AR, annonceur(whiteToken))$;
end

Algorithm 5: Détection d'une panne pour un AS u

Result: Annonce d'une panne

/* detectPanne() retourne l'AS innaccessible */

/*listId() retourne la liste des identifiants joins avec le dernier jeton reçu */

/*id() retourne l'identifiant de l'AS */

$v \leftarrow detectPanne()$;

/* L'AS v n'est plus joignable et je suis prioritaire pour annoncer la panne */

if $v \neq null$ **and** $v.id() > u.id()$ **then**

  $List \leftarrow listId()$;

  /* On avertit tout les AS concernés*/

  for $id \in List$ **do**

    $send(id, "panne")$;

  end

end

Algorithm 6: Détection d'une apparition de lien pour un AS u

Result: Annonce d'une apparition de lien

/* detectApparition() retourne le chemin accesible grâce à l'apparition d'un lien*/

/* cheminOscillant() retourne le dernier chemin à l'origine de la création d'un jeton*/

/* prefered(p1,p2) retourne vrai si p1 est préféré à p2 */

$path \leftarrow detectApparition()$;

/* Si path est meilleur que le dernier chemin oscillant*/

if $path \neq null$ **and** $prefered(path, cheminOscillant())$ **then**

  $List \leftarrow listId()$;

  /* On avertit tout les AS concernés*/

  for $id \in List$ **do**

    $send(id, "apparition")$;

  end

end

Algorithm 7: Réception d'une annonce de panne ou d'apparition d'un lien pour un AS u

Data : réception du message <"*apparition*"> ou <"*panne*">

Result: Réhabilite les chemins illégitimement interdits

/* gestion des interdictions (avec B ensemble des chemins interdits)*/

/* rehabiliter(p) réhabilite le chemin interdit p */

/* testDeRehabilitation(p) retourne vrai si p peut être réhabilité */

for $p \in B$ **do**

  if $testDeRehabilitation(p) = true$ **then**

    $rehabiliter(p)$;

  end

end

Bibliographie

[1] Clément Saad. Instabilité du protocole bgp. 2005.

[2] Julien Champ. Rapport de stage. 2005.

[3] C. Huitema. Routing in the internet (2nd edition). *Hardcover*, Januar 2000.

[4] Luc Saccavini. Le routage bg-4.

[5] Timothy G. Griffin and Gordon Wilfong. Analysis of the med oscillation problem in bgp. *in Proc. of the 10th IEEE Int. Conf. on Network Protocols*, pages pp. 90–99, 2002.

[6] Ken Schummacher. Stabilite dans bgp. December 2004.

[7] Timothy G. Griffin and Gordon Wilfong. A safe path vector protocol. *Proc. IEEE INFOCOM*, vol.2 :pp. 490–499, 2000.

[8] Timothy G. Griffin, F. Bruce Sherpherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *Proc. IEEE/ACM Transactions on Networking*, 2002.

[9] Bruno Quoitin and Sébastien Tandel. C-bgp user's guide (version 1.1.20). 2005.