UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Faculté des Sciences Appliquées
Département d'Ingénierie Informatique

Conception d'un langage flexible de définition de politiques de routage BGP

Promoteur : M. BONAVENTURE

Mémoire présenté en vue
de l'obtention du grade
d'Ingénieur Civil en Informatique
par Virginie VAN DEN SCHRIECK

Louvain-la-Neuve
Année académique 2004–2005

Table des matières

Introduction

Contexte

Plus qu'une évolution technologie, l'informatique est aujourd'hui devenue un véritable phénomène de société. A peine plus d'un demi siècle après son apparition, elle est à présent omniprésente dans la majorité des secteurs d'activités des pays les plus développés, et acquiert une importance non négligeable dans la vie de tous les jours. En plus de la gestion informatisée des données, elle offre également des moyens de communications extrêmement rapides et efficaces, permettant des liaisons entre des millions de machines situées aux quatre coins du globe.

Ces liaisons sont devenues possibles grâce à la mise en réseaux des ordinateurs, localement, ou globalement comme dans le cas d'Internet. L'ampleur de ce dernier est telle qu'il nécessite des dizaines de milliers d'appareils uniquement dédiés à son bon fonctionnement, répartis entre de nombreuses entités administratives différentes et communiquant via un ensemble de protocoles en constante évolution. La gestion de ces équipements est loin d'être un problème trivial, et nécessite de configurer avec soin chacune des machines concernées. Dans un réseau de taille importante comportant plus d'une centaine de routeurs, l'ensemble des configurations représente un système distribué de plusieurs centaines de milliers, voire même plusieurs millions de lignes. Ces configurations ne sont bien sûr pas toujours optimales ni exemptes d'erreurs, et cela peut entraîner des disfonctionnements dans une partie, voire même dans l'ensemble d'un réseau.

Description du travail

Nous allons donc nous pencher sur cette problématique tout au long du présent travail, afin d'identifier les besoins qui existent dans ce domaine et de tenter d'y apporter un début de solution. Nous procéderons en deux étapes : dans un premier temps, nous al-

lons faire le tour des techniques dédiées à la gestion des réseaux, et analyser de manière critique les problèmes qu'elles permettent ou non de résoudre. Cette analyse nous permettra de mettre en évidence le besoin de techniques de gestion de haut niveau. De telles techniques commencent à être déployées, mais uniquement dans le cadre de certains aspects particuliers des réseaux, comme par exemple la gestion de la qualité de service. Nous apporterons alors notre pierre à l'édifice dans le cadre de la seconde partie, en développant un langage de définition de politiques BGP de haut niveau, BGP étant un des protocoles les plus critiques pour le fonctionnement d'Internet. Ce langage donnera à un administrateur la possibilité de configurer ses routeurs BGP de manière automatique et optimale, sur base des politiques de routage qu'il désire mettre en oeuvre. Cette configuration automatique permettra de s'affranchir des erreurs qui peuvent apparaître lors de la manipulation manuelle de ces routeurs, et diminuera d'autant les problèmes de routage Internet qui peuvent en résulter.

Objectifs

Notre objectif consiste donc à développer un outil complet de configuration BGP, suffisamment flexible pour laisser toute latitude aux administrateurs pour la définition de leurs politiques de routage, mais produisant des fichiers de configuration exempts de toute erreur. Nous désirons également que cet outil soit capable de gérer la configuration iBGP de manière efficace, en étudiant une solution qui permette d'alléger au maximum les ressources du réseau considéré tout en garantissant la robustesse du système. Nous utiliserons ensuite cet outil pour définir des politiques de routage pour le réseau GEANT, afin de démontrer la capacité du langage de définition à exprimer des spécifications de routage réalistes et à produire les configurations correspondantes. Des simulations nous permettrons enfin d'évaluer les résultats produits, aussi bien en termes de respect des spécifications que d'efficacité et de robustesse.

Structure du document

Nous commencerons par présenter, dans le premier chapitre, trois outils permettant d'agir sur les fichiers de configuration des appareils du réseau. Nous détaillerons ensuite une technique permettant de gérer un réseau à partir de politiques de haut niveau, avant de conclure cette première partie. La seconde partie commencera par un chapitre présentant le protocole BGP, avant d'aborder le langage que nous avons conçu pour configurer efficacement un ensemble de routeurs BGP. Enfin, nous présenterons un outil permettant d'analyser des topologies iBGP et de choisir celle qui convient le mieux au système étudié, avant de conclure ce travail.

Première partie

Configuration et gestion de réseaux : Etat de l’art

Chapitre 1

Outils de gestion des configurations

1.1 Généralités

Il existe actuellement dans le monde des réseaux une multitude d'équipements interagissant entre eux. Pour configurer et gérer ces appareils, les gestionnaires de ces réseaux ont le choix entre différents outils. Avant de passer en revue quelques uns d'entre eux, il peut être utile de placer cette étude dans un contexte, c'est-à-dire définir clairement les activités d'un gestionnaire, indépendamment de la taille du réseau qu'il supervise. En effet, que ce soit à l'échelle d'un réseau d'entreprise ou d'un fournisseur de service Internet, un réseau comprend des appareils destinés à permettre à des machines terminales de communiquer entre elles. C'est de la configuration de ces appareils que dépend le bon fonctionnement de l'entièreté du réseau. Nous définirons donc la gestion d'un réseau comme étant l'ensemble des activités permettant la mise en place et la maintenance de ce dernier, et plus précisément, des appareils qui en composent l'ossature.

Après avoir dans cette section présenté les deux interactions principales d'un gestionnaire avec les appareils, à savoir la surveillance et le contrôle [1], nous détaillerons dans les chapitres suivants trois outils permettant de réaliser ces interactions.

1.1.1 Surveillance

La surveillance consiste, comme son nom l'indique, à observer et à analyser le trafic et les événements survenant dans un réseau. Cela comprend notamment l'analyse du com-

portement et de la configuration de ses composants, et de l'influence de ces paramètres sur l'ensemble du réseau. Cette fonction est donc uniquement observatoire. Le monitoring requiert des outils permettant de récolter l'information sur le réseau, mais également d'analyser cette dernière de manière statistique.

Pour cela, un gestionnaire a deux options : Soit il interroge régulièrement les appareils pour en extraire l'information dont il a besoin (*polling*), soit ce sont les appareils eux-même qui se chargent d'envoyer périodiquement leur état au gestionnaire (*event reporting*). Cette seconde option permet ainsi au gestionnaire d'être averti si des événements significatifs surviennent.

L'activité de surveillance peut concerner plusieurs types de données :

- Les performances du réseau : Cela comprend le temps de réponse d'un système, sa disponibilité, son pourcentage d'erreurs, son débit et son taux d'utilisation ;
- Les pannes du réseaux : Il s'agit de pouvoir détecter les pannes pouvant surgir n'importe où dans le réseau, et d'en déterminer les causes dans la mesure du possible.
- L'utilisation des ressources : Cela consiste à observer l'usage qui est fait de chaque ressource du réseau par les utilisateurs, afin de pouvoir l'optimiser.

1.1.2 Contrôle

Le contrôle d'un réseau consiste, quant à lui, à agir sur les composants pour en modifier la configuration ou leur faire accomplir certaines actions. Les deux aspects de la gestion de réseaux qui nécessitent essentiellement le contrôle de leurs composants sont la configuration et la sécurité.

La configuration d'un appareil, d'un sous-réseau ou de l'ensemble du réseau consiste à en modifier les paramètres dans le cadre de l'initialisation, de la maintenance ou de la mise hors-tension de ces derniers. La maintenance s'effectue généralement sur base des informations fournies par le monitoring. L'aspect sécuritaire de la gestion d'un réseau consiste, lui, à être capable de répondre à toute menace sur l'intégrité du réseau, la confidentialité des données, ou la disponibilité des systèmes.

1.2 L'interface en ligne de commande

La manière la plus simple de configurer un appareil est sans conteste l'utilisation de l'interface en ligne de commande de ce dernier. Cela consiste simplement à se connecter à l'appareil via Telnet par exemple, et à éditer le fichier de configuration.

1.2.1 Avantage

Cette technique ne demande aucune infrastructure particulière, et permet de tirer profit de toutes les fonctionnalités offertes par l'appareil.

1.2.2 Inconvénients

Le désavantage principal de cette outil est qu'il n'est absolument pas inter-opérable, dans le sens où chaque vendeur offre sa propre interface sans aucune standardisation entre appareils du même type. Un gestionnaire ayant à superviser des appareils d'origines différentes se devra donc de connaître les commandes spécifiques à chacun d'entre eux. De plus, gérer un réseau de grande taille uniquement via la ligne de commande peut se révéler assez ardu, puisque chaque appareil doit être configuré individuellement. Si le réseau contient plus d'une centaine de machines, il peut s'écouler plusieurs heures avant qu'elles ne soient toutes correctement configurées, et il risque d'y avoir des incohérences dans le réseau durant ce temps de latence, sans parler des erreurs de configuration qui peuvent survenir.

La surveillance du réseau n'est pas non plus facilitée par cette méthode, puisqu'il n'y a pas moyen pour le gestionnaire d'être averti en cas d'événement important survenant dans le réseau. De plus, l'analyse d'éventuelles données disponibles via la ligne de commande n'est pas simple, puisque ces dernières ne concernent qu'un appareil. Il faut donc effectuer un travail de synthèse entre toutes les informations recueillies sur les différents appareils pour obtenir des statistiques sur l'entièreté du réseau.

1.3 Simple Network Management Protocol

Au vu des limitations inhérentes à l'utilisation de la configuration en ligne de commande, il est évident que des outils permettant la gestion centralisée des équipements du réseau sont indispensables, surtout en présence de réseaux de grande taille. Le second outil que nous allons détailler, SNMP, constitue une première solution à ce problème. La description qui est fournie ci-dessous se base essentiellement sur [1]

L'acronyme SNMP (pour *Simple Network Management Protocol*) est généralement utilisé en référence à un ensemble de spécifications incluant le protocole lui-même, la définition d'un modèle d'information et de la base de données correspondante ainsi que les concepts associés.

1.3.1 Historique

Jusqu'à la fin des années 70, il n'existait aucun protocole pour la gestion des réseaux TCP/IP dans leur ensemble. L'outil alors utilisé dans ce cadre était ICMP (Internet Control Message Protocol), qui permet l'envoi et la réception de messages sur des routeurs ou d'autres appareils, comme par exemple les messages *echo* et *echo-reply*. Cela permet notamment de vérifier s'il est possible de communiquer avec les appareils du réseau. Un autre message permet également de mesurer les délais sur les liens de ce dernier. Si ces outils étaient efficaces au début de l'expansion d'Internet, il est assez vite apparu qu'il fallait développer des solutions plus puissantes pour la gestion de réseaux de plus en plus grands et de plus en plus complexes.

A la fin des années 80, le protocole SNMP fut adopté comme solution à court terme par l'Internet Activities Board pour la gestion des réseaux, en attendant un éventuel déploiement d'une solution plus complète standardisée par l'ISO, à savoir *CMIP over TCP/IP*, ou CMOT. Néanmoins, à l'heure actuelle, le protocole SNMP est largement utilisé et supporté par de nombreux appareils, et la mise en place de CMOT ne semble pas d'actualité.

1.3.2 Architecture et représentation de l'information

SNMP a été conçu comme un protocole de la couche application de la suite TCP/IP, et est prévu pour fonctionner au dessus du protocole UDP. Le modèle SNMP suppose l'existence de gestionnaires opérant à partir de stations de gestion et d'agents localisés sur les différents équipements du réseau. Un agent est un module logiciel dont la responsabilité est de fournir les informations au gestionnaire. Une station communique avec les éléments du réseau via SNMP, et doit donc supporter aussi bien SNMP que les protocoles sur lesquels ce dernier repose, c'est-à-dire UDP et IP. Les agents doivent être capables d'interpréter les requêtes SNMP, d'y répondre et d'envoyer de manière asynchrone des informations importantes mais non sollicitées. Il faut donc que les appareils supportent aussi SNMP et les protocoles sous-jacents. Si ce n'est pas le cas, il est possible d'utiliser un proxy, qui jouera alors le rôle de l'agent SNMP en récoltant l'information

sur les équipements ne supportant pas SNMP. La figure 1.1 présente une architecture SNMP typique. Puisque SNMP utilise le protocole UDP, qui est sans connexion, il sera également lui-même sans connexion : chaque échange entre le gestionnaire et les agents est une transaction séparée.

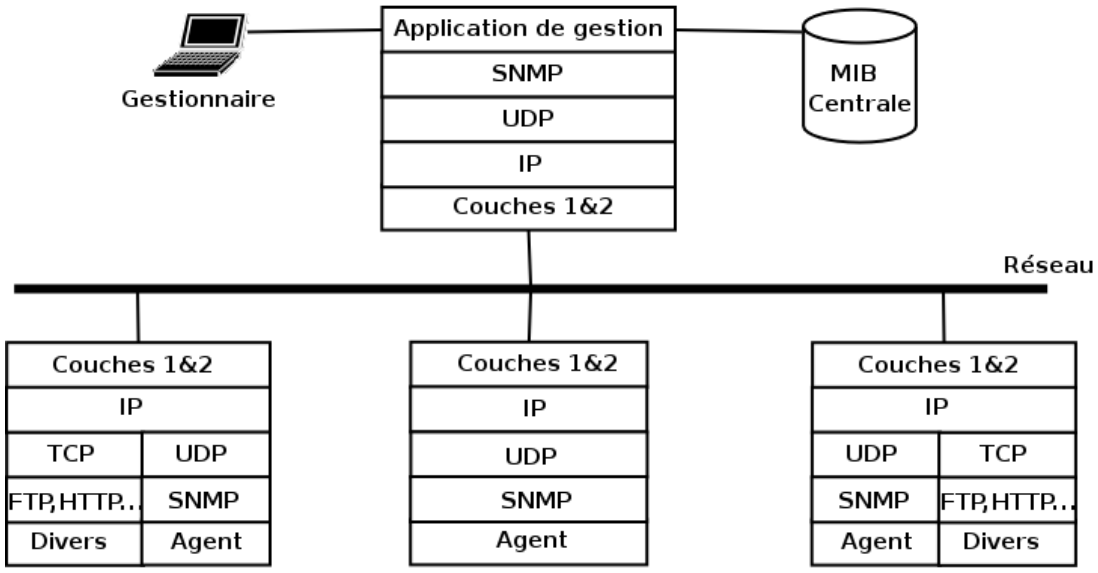

FIG. 1.1 – Architecture SNMP

Pour SNMP, les ressources du réseau sont représentées par des objets, c'est-à-dire des variables représentant chacune un aspect particulier d'un équipement. L'ensemble des informations concernant ces objets représente une base de données appelée *Management Information Base*, ou MIB. La MIB d'un agent sert donc de point d'accès à l'information pour le gestionnaire.

1.3.3 Management Information Base

Spécifications

Comme expliqué plus haut, les ressources gérées par un agent sont représentées de manière structurée par une *Management Information Base*, qui est une base de données reprenant toutes les informations utiles concernant l'appareil sur lequel l'agent

est hébergé. Sur le réseau, chaque noeud maintiendra une MIB reflétant le statut des ressources de ce noeud. Un gestionnaire pourra donc accéder aux ressources de ce noeud en lisant les valeurs des objets de la MIB, et pourra contrôler ces ressources en modifiant ces valeurs. Pour que ce système soit efficace, il faut que la MIB remplisse les deux conditions suivantes :

1. Un schéma commun et non ambigu de représentation de l'information doit être utilisé pour supporter l'inter-opérabilité. Cela est réalisé au moyen d'un langage de définition de données appelé SMI (*Structure of Management Information*).
2. L'objet ou les objets représentant une ressource particulière doivent être identiques à chaque noeud. Il faut donc pour cela que tous les objets de la MIB soient définis à priori.

SMI

La première version du langage de définition de données (SMIv1) est décrite dans la RFC 1155 [2], et définit le schéma selon lequel l'information de gestion d'une entité est organisée. Il identifie les types de données utilisables, ainsi que la manière dont les ressources sont nommées et représentées. Pour cela, SMI se base sur la notation ASN.1, ou du moins sur un sous-ensemble de cette dernière. Il n'autorise que des types de données simples, soit des scalaires ou des tableaux bidimensionnels de scalaires, afin de simplifier les implémentations. SMI doit donc fournir des techniques standardisées pour les trois points suivants :

1. La définition de la structure de la MIB : Les objets sont définis de manière hiérarchique, de telle sorte qu'ils forment ensemble une structure d'arbre. Pour cela, on associe à chaque type d'objet de la MIB un identificateur composé d'une suite hiérarchique d'entiers, chacun d'entre eux représentant un arc de l'arbre des objets. La figure 1.2 représente une partie de cette arbre, dans lequel le noeud nommé *internet* possède comme valeur d'identificateur d'objet 1.3.6.1. Cette valeur sert aussi de préfixe pour les identificateurs des objets descendants de ce noeud.
2. La définition de la structure de chaque objet, c'est-à-dire la syntaxe et la valeur de celui-ci : Pour définir un type d'objet, SMI utilise la macro OBJECT-TYPE définie dans la RFC 1212 [3].
3. L'encodage des valeurs de l'objet : Les objets de la MIB sont encodés via les règles d'encodages de bases (BER) associées à ASN.1.

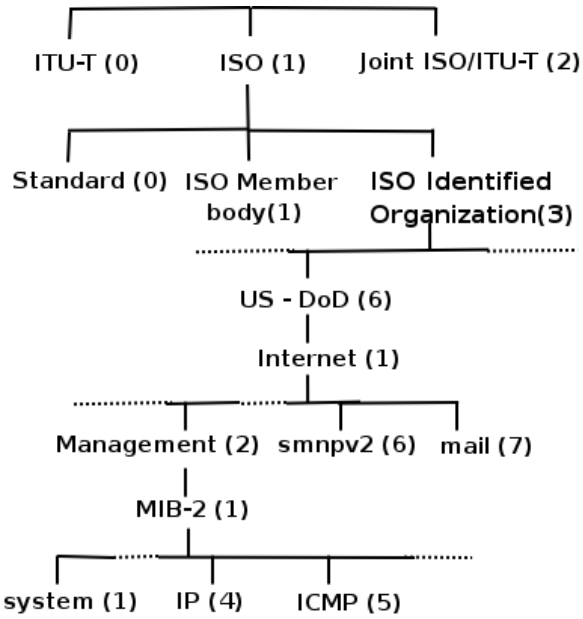

FIG. 1.2 – Arbre d'identification des objets

MIB-II

Actuellement, la première version de MIB a été supplantée par sa seconde version, définie par la RFC 1213 [4], et dont elle est un sous-ensemble. MIB-II contient donc des groupes et des objets supplémentaires. Les critères pour inclure un objet dans une MIB-II sont les suivants :

- L'objet doit être essentiel soit pour la gestion des failles, soit pour la gestion de la configuration, et avoir une utilité évidente.
- Seuls les objets non-critiques du point de vue sécurité sont admis, et ce en raison du manque de sécurité dans les protocoles actuels de gestion.
- Pour éviter les variables redondantes, un objet de la MIB ne peut être dérivé d'un autre.
- La définition des objets doit être indépendante de leur implémentation.

Le groupe des objets MIB-II est représenté partiellement par le dernier niveau de la figure 1.2, et est lui-même subdivisé en plusieurs groupes. Aucun de ces objets n'a un statut optionnel, puisque leurs concepteurs n'ont considéré que des objets essentiels.

1.3.4 Définition du protocole SNMP

Le protocole SNMP est utilisé pour transférer des informations de gestion entre gestionnaires et agents. Il permet notamment les requêtes/réponses entre ces entités. Typiquement, une requête est utilisée par le gestionnaire pour demander (*Get*) ou modifier (*Set*) les valeurs d'un objet MIB associé à une ressource. Il est également possible pour un agent d'envoyer des messages non sollicités (*Trap*), par exemple dans le cas où la congestion atteint un certain niveau sur un lien. SNMP ne permet par contre pas d'ajouter ou de modifier des instances d'objets dans la MIB, ni de déclencher des actions, et il n'a accès qu'aux objets situés dans les feuilles de la structure en arbre des identificateurs d'objets. Il peut néanmoins agir sur les tableaux bidimensionnels, en y accédant via une combinaison entre l'identificateur de l'objet et les index du tableau.

Politique d'accès

Pour que les agents puissent contrôler l'accès à leur MIB, il est nécessaire de disposer d'un moyen d'identification des gestionnaires. SNMP définit le concept de communauté, qui est une relation entre un agent SNMP et un ensemble de gestionnaires SNMP. Cette relation spécifie l'authentification du gestionnaire, le contrôle d'accès à la MIB et les caractéristiques d'un éventuel proxy. Une communauté possède un nom unique pour l'agent concerné, que les gestionnaires appartenant à la relation utiliseront dans chaque requête.

Format des messages

Un message SNMP contient plusieurs éléments : Un numéro de version, le nom d'une communauté et l'un des 5 PDUs (*Protocol Data Unit*) disponibles dans SNMPv1 :

- *GetRequest* : Contient une liste d'instances d'objets dont les valeurs sont requises par le gestionnaire.
- *GetNextRequest* : Identique au PDU *GetRequest*, excepté le fait que le gestionnaire demande la valeur des instances d'objets situés après ceux de la liste que le PDU contient, dans l'ordre lexicographique.
- *SetRequest* : Ce PDU a le même format que le PDU *GetRequest*, excepté qu'il est utilisé par le gestionnaire pour écrire des valeurs des instances des objets plutôt que de les lire.
- *GetResponse* : Contient la réponse de l'agent à un PDU *GetRequest*, *SetRequest* ou *GetNextRequest*, dont il reprend l'identifiant.
- *Trap* : PDU émis par un agent SNMP pour avertir la station de gestion d'un événement particulier.

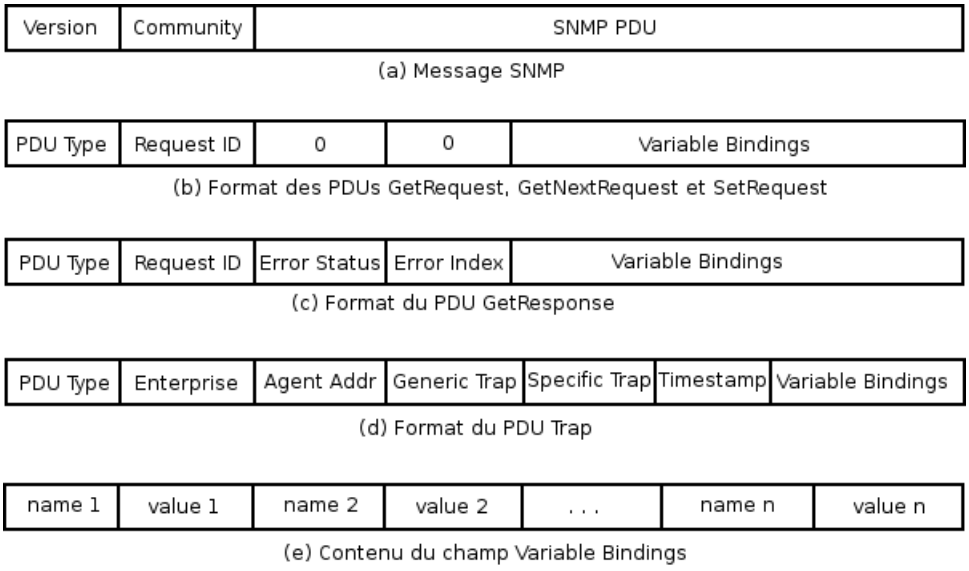

FIG. 1.3 – Format des messages

1.3.5 Avantages

Le point fort le plus évident de ce protocole est qu'il définit une syntaxe unique pour tout un ensemble d'équipements, facilitant ainsi l'inter-opérabilité. Cette dernière profite également du mécanisme permettant d'ajouter un sous-arbre privé à la MIB. Il est, de plus, largement répandu, et implémenté sur les équipements de nombreux vendeurs.

1.3.6 Inconvénients

SNMP représente une des premières solutions qui ont été proposées pour une gestion centralisée d'un réseau. Il souffre donc malheureusement de certaines faiblesses, notamment dans le cas de réseaux de grande taille. En effet, il faut envoyer un paquet de requête pour obtenir un paquet d'information concernant un équipement. Il peut donc y avoir rapidement un trafic SNMP non négligeable qui peut augmenter les délais de transmission dans le réseau. Ce problème est inhérent à la technique du *polling*, et, quant aux notifications asynchrones (PDU *Trap*), les limitations d'UDP impliquent qu'elle peuvent ne pas arriver à destination. De même, le protocole n'est pas adapté au transfert de grands volumes de données, comme par exemple le transfert d'une table de routage.

De plus, le modèle d'information définissant les MIBs reste globalement limité, malgré les possibilités d'extensions. Il ne permet ainsi que de consulter les informations concer-

nant un appareil en particulier, et non des données concernant le réseau dans son ensemble.

Enfin, du fait de sa procédure d'authentification plutôt simpliste, il est préférable de n'utiliser SNMP que pour la surveillance du réseau, et non pour son contrôle. Ce dernier n'est d'ailleurs pas facilité par la contrainte impliquant que pour exécuter une commande sur un routeur, il n'y a pas d'autre alternative que de modifier la valeur correspondante dans la MIB. De nombreux vendeurs ont par ailleurs choisi de ne pas supporter la commande SET en raison du risque d'insécurité qu'elle implique.

1.4 Le protocole NETCONF

1.4.1 Introduction

Comme nous l'avons vu plus haut, si SNMP est largement répandu, il n'est pas vraiment adapté à la gestion des configurations, et, malgré de nombreuses tentatives pour combler ses déficiences, les problèmes de fond n'ont pas été résolus. Plus récemment, l'attention semble se porter sur l'utilisation de technologies XML comme alternative à SNMP, ces dernières permettant un traitement hiérarchique de l'information de configuration. Dans cette optique, l'IETF a mis sur pied un groupe de travail chargé de définir un nouveau protocole de configuration de réseaux basé sur l'utilisation de XML. Ce groupe a donc produit, en mars 2003, la définition du protocole en question, appelé NETCONF.

Ce protocole peut être aisément utilisé dans un système de configuration automatisée grâce au mécanisme d'encodage hiérarchique flexible qu'est XML. Ainsi, à l'aide de technologies de transformation basées sur XML telles que XSLT par exemple, il est possible de créer un système de génération automatique de configurations à partir de données sur la topologie, les liens, les politiques, les clients et les services du réseau.

Nous allons donc, tout au long de ce chapitre, présenter la structure et les caractéristiques de ce protocole, en nous basant sur [5] et [6].

1.4.2 Architecture

Présentation de NETCONF

La caractéristique principale du protocole NETCONF est qu'il utilise un paradigme RPC pour interagir avec un appareil du réseau. Les paramètres des requêtes RPC des gestionnaires seront encodés avec XML avant qu'elles ne soient envoyées sur le réseau par

l'intermédiaire d'un protocole applicatif orienté connexion. L'agent concerné encodera et enverra sa réponse de la même manière.

NETCONF permet également de définir des extensions au protocole, appelées *capabilities*, de telle sorte qu'un client peut découvrir d'éventuelles caractéristiques supplémentaires supportées par le serveur et adapter son comportement pour en tirer profit.

Le protocole utilise les termes *client* et *serveur* pour désigner d'une part l'application de gestion et d'autre part l'appareil. La connexion logique entre ces deux éléments est appelée *session NETCONF*. Un appareil doit être capable de supporter au moins une session à la fois. NETCONF impose que les changements d'attributs de configuration globaux réalisés dans une session soient visibles dans toutes les autres, mais les changements spécifiques à une session donnée n'affectent que cette session.

Le modèle sépare en deux catégories les informations auxquelles il est possible d'accéder sur un système : les données d'information et les données d'état. Les premières sont l'ensemble des données modifiables nécessaires pour transformer un système de l'état initial à l'état courant, tandis que les secondes sont les données restantes, c'est-à-dire les données telles que les informations de statut, les statistiques collectées, les fichiers utilisateurs ou les bases de données.

Modèle en couche

Le modèle NETCONF est constitué de quatre couches logiques : la couche contenu, la couche opération, la couche RPC et la couche du protocole applicatif.

- Couche Contenu : Cette couche définit les données de configuration spécifiques à la gestion des appareils (par ex : adresses IP, ID/mot de passe du gestionnaire,...). Étant donné que cette couche dépend des possibilités offertes par les vendeurs, elle n'est pas traitée par NETCONF. Néanmoins, des démarches ont été entreprises pour résoudre ce problème et formuler un langage de définition de données standards, ainsi que pour définir une couche contenu standardisée. Des informations supplémentaires sur ces développements peuvent être trouvées dans [7].
- Couche Operations : Cette couche inclut les opérations de base ainsi que d'éventuelles opérations de gestion supplémentaires. Les opérations permettent d'interagir avec la couche Contenu, pour lire ou modifier une configuration, en gérer l'accès par un système de verrous, ou gérer la session NETCONF en cours.
- Couche RPC : Cette couche fournit un modèle de communication de type RPC indépendant du protocole applicatif utilisé. Elle permet d'encoder très simplement les requêtes et les réponses en XML, ainsi que les éventuels messages d'erreur.

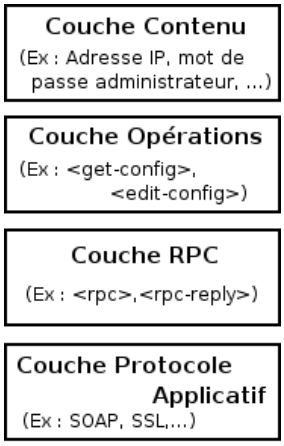

FIG. 1.4 – Modèle en couche

– Couche Protocole Applicatif :C'est la couche gérant la communication proprement dite entre le client et le serveur. N'importe quel protocole applicatif peut être utilisé, pour autant que ce dernier remplisse les exigences suivantes :

1. Il doit être orienté connexion, pour fournir un service fiable permettant de délivrer les données de manière ordonnée. Les connexions doivent être durables et persistantes entre les opérations du protocole. De plus, lorsqu'une connexion est fermée, les ressources du serveur utilisées lors de cette connexion doivent être automatiquement libérées.
2. Le protocole doit également garantir l'authenticité des interlocuteurs, ainsi que l'intégrité et le caractère privé des données.
3. Un des buts de NETCONF est de fournir une interface à l'appareil qui soit la plus proche possible de son interface native. Il faut donc que le protocole applicatif soit capable d'utiliser le mécanisme d'authentification de l'appareil.

Implémentations proposées par le groupe de travail NETCONF

Trois protocoles ont été pour le moment considérés comme protocoles applicatifs, à savoir SSH, SOAP/HTTP et BEEP.

1.4.3 Modèle RPC

Comme mentionné plus haut, NETCONF utilise un modèle de communication basé sur RPC. Le protocole définit donc un schéma de communication basé sur des requêtes

et des réponses, qui seront respectivement délimitées par les balises XML suivantes : <rpc> et <rpc-reply>. Ces deux éléments sont définis dans l'espace de nom suivant :

`urn:ietf:params:xml:ns:NETCONF:base:1.0`

L'élément <**rpc**>

Cet élément est utilisé pour délimiter une requête NETCONF envoyée par un gestionnaire à un agent. Le nom et les paramètres de la requête en elle-même seront encodés comme contenu de la balise, le nom étant défini par la balise directement interne à <rpc>, et les paramètres par les balises incluses dans cette dernière. La balise <rpc> aura un attribut obligatoire *message-id* contenant un string d'identification qui sera recopié dans l'élément <rpc-reply> répondant à la requête. Si d'autres attributs sont présents dans la balise <rpc>, ils seront également reproduits à l'identique dans la balise <rpc-reply> correspondante.

L'élement <**rpc-reply**>

Cet élément est envoyé en réponse à une opération RPC, et possède comme attribut obligatoire un *message-id* égal à l'attribut correspondant de l'élément <rpc> auquel il répond. Il recopiera également, comme déjà mentionné plus haut, tous les autres attributs de cet élément <rpc>.

L'élément <**rpc-error**>

Cet élément est envoyé dans un message <rpc-reply> chaque fois qu'une opération RPC a donné lieu à une erreur. Il contient des informations sur cette dernière, comme par exemple la couche à laquelle elle s'est produite, sa gravité, ou encore l'expression identifiant le chemin vers l'élément associé à l'erreur.

L'élément <**ok**>

Cet élément est envoyé dans un message <rpc-reply> si l'opération RPC correspondante s'est déroulée sans erreur.

Pipelining

Les requêtes <rpc> NETCONF sont traitées de manière séquentielle par l'appareil, et il est donc possible d'envoyer de nouvelles requêtes même si les précédentes n'ont pas encore été traitées.

1.4.4 Couche Opérations

Cette couche définit les opérations de bas niveau permettant d'agir sur la configuration des éléments du réseau. Ces opérations seront donc incluses dans les requêtes RPC définies à la couche inférieure. Elles offrent la possibilité de récupérer, de modifier ou de supprimer des configurations en agissant sur ce que NETCONF définit comme étant des *datastores*.

Datastore

Le protocole NETCONF comprend plusieurs datastores, qui sont des fichiers contenant chacun une version spécifique de la configuration d'un appareil. Plus précisément, un datastore de configuration est défini comme étant l'ensemble complet des données de configuration nécessaires pour amener un appareil de l'état initial à l'état opérationnel désiré. Il ne comprend pas de données d'état ou de commandes à exécuter.

Le modèle de base n'impose l'existence que d'un seul datastore, appelé $<$running$>$. Ce dernier comprend toutes les informations concernant la configuration courante de l'appareil. Elle est unique et présente à tout moment sur ce dernier.

D'autres datastores sont définis comme *capabilities* NETCONF, comme par exemple un datastore *startup* utilisé lors du démarrage, ou un datastore *candidate*, permettant de modifier et tester une configuration sans perturber le fonctionnement de l'appareil.

Opérations

Tout comme pour les datastores, NETCONF définit des opérations de base et des opérations supplémentaires qu'un appareil peut éventuellement supporter. Les opérations de base peuvent être groupées en trois catégories : les opérations agissant sur les configurations et les informations d'état, les opérations permettant de gérer l'accès aux ressources, et enfin celles permettant de contrôler des sessions NETCONF.

Etant donné qu'une opération peut échouer pour de multiples raisons, il est conseillé de vérifier la valeur retournée par la réponse RPC afin d'obtenir les éventuels messages d'erreur.

Gestion des configurations et des informations d'état : Les quatre premières opérations de cette catégorie agissent directement sur un datastore de configuration. La dernière opération, <get>, permet quant à elle d'obtenir aussi bien des informations de configuration que des informations d'état.

Si une opération s'effectue sans erreur, l'élément <rpc-reply> renvoyé contiendra soit les informations demandées, soit l'élément <ok>. Dans le cas contraire, il contiendra un <rpc-error> détaillant les raisons de l'échec de l'opération.

- <get-config> : Cette opération permet de récupérer une partie ou l'entièreté d'une configuration, en indiquant en paramètre le nom du datastore considéré ainsi qu'un filtre spécifiant la partie de la configuration à laquelle on veut accéder. Si l'opération s'effectue correctement, le <rpc-reply> envoyé en réponse contiendra un élément <config> contenant les informations demandées.

- <edit-config> : Cette opération offre la possibilité d'éditer, de remplacer, de créer ou d'effacer une partie ou l'entièreté d'une configuration dans l'appareil considéré. Elle offre de nombreuses options permettant de s'assurer du bon déroulement de l'opération, à savoir la possibilité de tester la configuration, ou encore d'indiquer le comportement souhaité en cas d'erreur : restaurer l'état initial, ignorer l'erreur, ou tout simplement arrêter l'opération.

- <copy-config> : A la différence d'<edit-config>, <copy-config> agit sur l'entièreté d'un datastore de configuration, créant ou remplaçant la totalité de la configuration. En raison de son caractère critique, un appareil peut choisir de ne pas permettre l'application de cette opération sur la configuration courante.

- <delete-config> : Cette opération a pour effet de supprimer le datastore de configuration spécifié en paramètre. Ce dernier ne peut être le datastore <running>.

- <get> : Cette opération est semblable à <get-config>, excepté le fait qu'elle permet en plus d'accéder à des informations d'état. Au lieu d'un élément <config>, <get> renverra les informations dans un élément <data> contenu entre des balises <rpc-reply>. Il est possible d'utiliser un filtre pour spécifier les informations auxquelles on veut accéder. Si rien n'est indiqué, la réponse contiendra à la fois la configuration de l'appareil et les informations d'état.

Opérations pour la gestion des ressources Afin de pouvoir modifier une configuration sans qu'un autre client n'accède en même temps à la ressource considérée, risquant ainsi de rendre la configuration incohérente, NETCONF définit des opérations permettant de verrouiller les ressources et de garantir un accès exclusif à ces dernières. Les verrous sont prévus pour être de courte durée, et garantissent l'absence d'interaction aussi bien avec d'autres clients NETCONF qu'avec des clients utilisant d'autres technologies de configuration comme CLI ou SNMP. Les éventuelles requêtes effectuées par ces derniers doivent donc automatiquement échouer. Les verrous sont globaux, et agissent sur l'entièreté d'une configuration, ce qui signifie que si un verrou d'une entité non-NETCONF est déjà posé sur une partie de celle-ci, il sera impossible d'y poser un verrou NETCONF.

Un verrou est maintenu jusqu'à ce qu'il soit explicitement relâché, ou jusqu'à ce que la session durant laquelle il a été posé se termine. Même si la session se termine de manière anormale, c'est-à-dire sans avoir été explicitement fermée par le client (par exemple suite à l'expiration d'un *timeout*), les ressources verrouillées au cours de celle-ci doivent être libérées. Il est donc possible de déverrouiller une ressource dont le verrou est détenu par une autre session en utilisant par exemple l'opération <kill-session> pour terminer cette dernière.

- <lock> : Cette opération permet d'acquérir un verrou sur le datastore de configuration spécifié par le paramètre <target>. Par défaut, ce dernier aura la valeur <running/>. L'acquisition d'un verrou échouera si la ressource est déjà verrouillée par une autre entité (NETCONF ou autre), si le verrouillage de ressources n'est pas supporté par l'appareil, ou encore si la configuration cible a été modifiée, mais que ces changements n'ont pas encore été commis. Dans le cas d'un échec de la requête, le message <rpc-reply> contiendra un <rpc-error> spécifiant éventuellement l'identifiant de session du propriétaire du verrou si la ressource est déjà verrouillée. S'il ne s'agit pas d'une session NETCONF, cet identifiant sera nul. Si le verrouillage s'effectue correctement, l'élément <ok> est envoyé.

- <unlock> : <unlock> est utilisé pour relâcher un verrou précédemment posé. Il possède le paramètre <target> spécifiant le datastore de configuration à libérer (*running* par défaut), et renvoie un <rpc-reply> contenant soit <ok> soit <rpc-error>, en fonction du succès ou de l'échec de l'opération. Cette dernière peut en effet échouer notamment pour les deux raisons suivantes : Si le verrou spécifié n'est plus actif, ou si la session effectuant l'opération de déverrouillage n'est pas la même que celle qui a effectué le verrouillage de la ressource.

Opérations contrôlant les sessions NETCONF Comme mentionné plus haut, une session NETCONF est une connexion logique entre un administrateur réseau ou une application de configuration de réseau, et un appareil d'un réseau. Si l'établissement des sessions et le processus d'identification qui en découle est laissé à la responsabilité du protocole applicatif, NETCONF fournit par contre des opérations permettant de les terminer, que ce soit de manière normale ou abrupte.

De la même manière que pour les opérations précédentes, la réponse <rpc-reply> correspondant à la requête contiendra un élément <ok> en cas de succès de l'opération, et un élément <rpc-error> en cas d'échec.

- <close-session> : Cette opération permet de terminer normalement une session NETCONF. Lorsqu'une entité reçoit ce message, elle va terminer les opérations en cours puis libérer les verrous et les ressources utilisés par la session avant de fermer toutes les connexions qui y sont liées.

- <kill-session> permet quant à lui de terminer abruptement une session à partir d'une autre. La session à terminer est identifiée par le paramètre <session-id>. Celui-ci doit être différent de l'identifiant de la session courante, sans quoi une erreur 'Bad Value' est renvoyée. Les opérations en cours dans la session cible seront immédiatement interrompues, les ressources et verrous qu'elle détient relâchés, et toutes les connexions ouvertes par la session fermées.

1.4.5 Capabilities

NETCONF définit une *capability* comme étant un ensemble de fonctionnalités implémentées au dessus des spécifications de base. Tout client ou serveur NETCONF a le choix d'éventuellement supporter certaines capabilities, ou encore d'ignorer des capabilities supportées par son interlocuteur. Elles peuvent être définies suivant une structure particulière, et être publiées comme définitions standards ou comme définitions propriétaires.

Les capabilities sont identifiées par des URI, et annoncées par les interlocuteurs d'une session NETCONF au début de celle-ci. Pour cela, chacun d'eux envoie un élément <hello> contenant une liste des capabilities qu'il supporte. Ces éléments <hello> sont envoyés simultanément dès que la connexion est ouverte, sans attendre la réception des

capabilities de l'autre interlocuteur. Les capabilities NETCONF sont toutes identifiées par l'URI suivante, le suffixe *name* représentant le nom de la capability :

```
urn:ietf:params:xml:ns:netconf:base:1.0#name
```

Les capabilities principales permettent par exemple d'autoriser la validation d'une configuration avant de l'appliquer à un appareil, de rendre possible l'édition de la configuration courante, ou encore d'ajouter une configuration supplémentaire définissant l'état de l'appareil au démarrage.

1.4.6 Avantages

Le protocole NETCONF présenté dans ce chapitre a été conçu pour fournir une nouvelle solution pour la gestion des réseaux n'ayant pas les limitations de la configuration par CLI ou SNMP. Ainsi, NETCONF assure la sécurité de ses échanges par l'utilisation d'une procédure d'identification et d'un protocole applicatif permettant par exemple le cryptage des données par SSH ou TLS. Le nouveau protocole offre également plus de flexibilité que SNMP grâce à la définition des capabilities, et améliore les opérations de configuration en permettant éventuellement de valider ces dernières avant de les appliquer, ou encore d'annuler les changements effectués en cas d'erreur. Il utilise également XML pour l'encodage, ce qui facilite le traitement des données grâce aux nombreux outils XML existants.

1.4.7 Inconvénients

NETCONF fournit un outil intéressant pour la communication entre les gestionnaires et les équipements du réseau. Il n'est cependant qu'une des pièces du puzzle consistant à obtenir un outil de gestion complet, et doit donc être utilisé en collaboration avec d'autres outils permettant par exemple de gérer les interactions entre les configurations de différents appareils. La couche Contenu du protocole doit également être complétée par un modèle de représentation de l'information.

Chapitre 2

La gestion sur base de politiques

2.1 Principe

Nous avons vu dans le chapitre 1 un ensemble d'outils permettant de modifier les configurations des appareils du réseau. Nous allons à présent aborder une technique de gestion de réseau de haut niveau, radicalement différente des interactions relativement simples autorisées par ces outils. En effet, la gestion sur base de politiques (*Policy-based Network Management*) se définit comme étant l'utilisation de règles pour gérer la configuration et le comportement d'une ou plusieurs entités [8]. Une règle, dans le cadre de cette méthodologie, lie un ensemble d'actions à un ensemble de conditions : Les conditions sont évaluées afin de déterminer s'il faut appliquer les actions. Une extension possible de cette définition est l'ajout d'un ensemble d'événements, qui peuvent déclencher l'évaluation des conditions.

L'objectif principal de cette approche est de gérer le réseau de manière unifiée : Les politiques peuvent servir aussi bien à changer les caractéristiques d'un certain type de trafic, comme les algorithmes de *queuing* ou *dropping* en fonction des valeurs de certaines mesures (par ex. le nombre de paquets reçus) qu'à contrôler l'usage qu'un utilisateur fait des ressources du réseau. Il faut donc travailler à plusieurs niveaux d'abstraction. Pour cela, il est nécessaire de définir plusieurs niveaux de politiques. Ainsi, le comportement général du réseau peut être défini par des politiques du niveau supérieur, appelé *Business* [8]. Ces politiques doivent alors pouvoir être interprétées en terme de politiques du niveau inférieur, appelé par exemple *System*. Les politiques du niveau *Business* seront explicitées en termes généraux, tandis que celles du niveau *System* seront plus techniques. Il est ainsi possible de définir 5 niveaux de politiques, comme le montre la figure 2.1. Il faut alors disposer, à chaque niveau, d'un modèle pour représenter l'information ainsi qu'un langage pour définir les règles. Il est également nécessaire d'assurer une

correspondance précise entre les modèles et les langages des niveaux successifs.

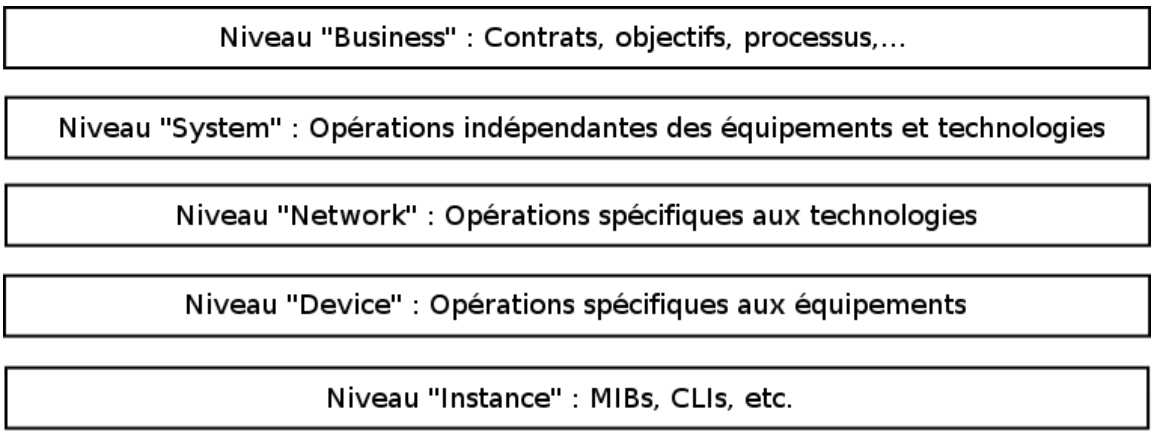

FIG. 2.1 – Niveaux d'abstractions

2.2 Représentation de l'information

Un réseau peut souvent être très complexe, et les politiques utilisées pour la gestion peuvent s'exprimer à différents niveaux de langage. Il est donc essentiel de disposer d'une représentation précise du système, afin que l'information soit bien définie et comprise par tous, aussi bien au niveau des politiques dites *Business* qu'au niveau des équipements. Il faut pour cela trois éléments : Les modèles et langages déjà mentionnés plus haut, ainsi qu'un dictionnaire de données. Ils permettront alors d'établir la correspondance entre les règles du niveau *business* et les politiques régissant l'utilisation des ressources des différentes entités.

2.2.1 Modèle d'information

Un modèle d'information définit les caractéristiques de haut niveau du système PBNM et, grâce à sa structure rigoureuse, aide à la conception et à l'implémentation de ce dernier. C'est une abstraction et une représentation des entités de l'environnement à gérer, ce qui inclut la définition de leurs attributs, des opérations qu'ils supportent, et des relations qui existent entre eux (modèle orienté-objet). Il explicite donc la sémantique, le comportement et les interactions de ces entités. Il utilise une représentation utilisant des types de données standardisés pour permettre aux différents composants de partager et échanger de l'information. Le fait que cet outil permette de décrire différents types

d'entités signifie que des relations peuvent être établies entre des entités à différents niveaux d'abstraction. Ainsi, on peut représenter dans ce modèle aussi bien des routeurs et des switches que des utilisateurs et des services.

2.2.2 Langages de politiques

Les langages permettent d'exprimer directement la sémantique qui permet à chaque niveau de politiques de traduire ses informations et concepts aux autres couches qui ont besoin de ces informations. Ils sont également compréhensibles par les machines, ce qui permet entre autres de les utiliser pour la détection de conflits. L'utilisation de langages permet surtout de lier entre elles les différentes abstractions à partir desquelles les polices sont construites, quel que soit leur niveau.

2.2.3 Dictionnaire de données

Le dictionnaire de données permet de s'assurer que les différents modèles et langages agissant à chaque niveau d'abstraction parlent bien des mêmes éléments. Il définit notamment un ensemble de synonymes par lesquels une entité est connue, de telle sorte que les différents utilisateurs soient capables de reconnaître cette entité même si elle est désignée par un autre terme que celui qu'ils connaissent. Il complète également les modèles en y ajoutant des informations supplémentaires qui permettent de mieux comprendre ces derniers.

2.3 Architecture d'un système PBNM

Un système de gestion utilisant des politiques peut s'avérer assez complexe, les différentes règles interagissant souvent les unes avec les autres. Il faut donc prévoir l'infrastructure nécessaire pour contrôler l'application de ces règles ainsi que les conflits qui peuvent apparaître entre elles. Le schéma 2.2 présente un exemple d'architecture typique de ce genre de système, dont nous allons décrire les principaux éléments.

2.3.1 L'outil de définition des politiques

C'est le composant permettant à l'administrateur d'entrer, de stocker, de consulter ou de modifier les politiques et les informations qui y sont liées, ainsi que de contrôler l'application de ces dernières. Il effectue également une validation simple des politiques entrées par l'administrateur, afin de repérer et permettre de corriger des éventuelles erreurs et conflits évidents avant que les politiques ne soient stockées dans l'élément servant de dépôt.

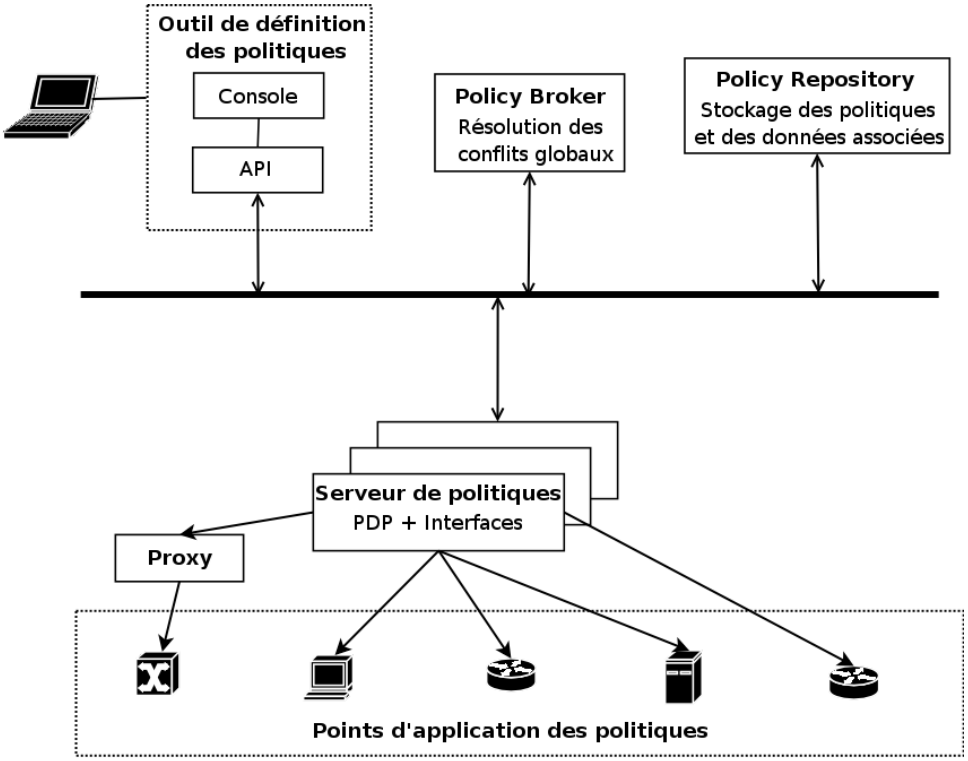

FIG. 2.2 – Architecture d'un système PBNM

2.3.2 Le dépôt de politiques (*Policy Repository*)

C'est un élément passif servant uniquement au stockage des politiques et des informations qui y sont liées. Il ne contient donc que des données, et n'effectue aucune opération sur ces dernières.

2.3.3 Le coordinateur (*Policy Broker*)

Ce composant contrôle les interactions entre les différents serveurs de politiques, d'une part pour assurer l'absence de conflits lorsque ces derniers doivent travailler ensemble, d'autre part pour coordonner l'application des politiques dans les différents serveurs. Il coordonne donc la communication et la collaboration de ces derniers.

2.3.4 Les serveurs de politiques

Ces éléments comportent plusieurs composants qui prennent les décisions et accomplissent les actions qui affecteront un ou plusieurs éléments du réseau. Ces composants peuvent être regroupés en deux groupes : L'un sera responsable de la logique de décision, ce qui signifie qu'il traduit les politiques qu'il reçoit dans le niveau d'abstraction approprié, les valide, et effectue une analyse de détection de conflits avec les autres politiques dont il est responsable. Un des éléments les plus importants de ce groupe est le *Policy Decision Point* (PDP), qui décide où et quand appliquer telle ou telle politique sur tel ou tel équipement. L'autre groupe sert d'interface avec les éléments du réseau contrôlés par le serveur.

2.3.5 Les *proxies*

Un *proxy* est utilisé pour contrôler les éléments du réseau qui ne peuvent communiquer directement avec le PDP.

2.3.6 Les points d'application des politiques

Un point d'application de politiques est un élément du système qui applique les politiques à un équipement particulier, en utilisant les commandes ou le protocole approprié pour contrôler ce dernier (CLI, NETCONF, SNMP, etc.)

2.4 Avantages

L'avantage principal de cette approche est qu'elle permet une gestion de haut niveau du réseau, en exprimant des règles dans un langage ad-hoc. Ensuite, ces règles sont traduites dans des niveaux de moindre abstraction, afin de pouvoir les appliquer aux équipements. Cette approche permet également une inter-opérabilité totale, puisque le système PBNM est capable de s'interfacer à n'importe quel équipement, qu'il soit contrôlable via CLI, NETCONF, ou n'importe quel autre outil de configuration.

2.5 Inconvénients

Le fait que cette approche soit de haut niveau nécessite une infrastructure assez importante pour pouvoir appliquer les politiques et s'assurer qu'elles ne soient pas en conflit les unes avec les autres. De plus, ces politiques doivent être définies avec soin, en utilisant les modèles d'informations ad-hocs afin que les traductions avec les autres niveaux d'abstraction soient possibles. Sa mise en oeuvre peut donc s'avérer assez lourde.

Chapitre 3

Conclusion de la première partie

Parmi les trois outils qui ont été présentés au chapitre 1, l'interface en ligne de commande apparaît clairement insuffisante dès qu'il y a plus de quelques appareils à configurer, et est à proscrire en raison des nombreuses erreurs de configuration que cette pratique entraîne. SNMP est une alternative intéressante, permettant de travailler à plus grande échelle, mais souffre encore de nombreuses faiblesses, ce qui ne le rend guère intéressant que pour la surveillance de réseaux simples. La solution idéale pour interagir avec chacun des appareils semble donc être l'utilisation du protocole NETCONF, grâce à l'utilisation de XML, permet de modifier de manière sûre et efficace n'importe quelle configuration.

La souplesse de NETCONF permet l'intégration de ce dernier dans des systèmes de gestion de réseaux complexes et automatisés, permettant une configuration globale et cohérente de l'ensemble des appareils du réseau. Ce genre de pratique permet de s'affranchir des erreurs dues à des configurations manuelles, et peut offrir une interface de gestion de haut niveau aux administrateurs. Un exemple de tel système est la gestion par utilisation de politiques, qui a été présentée au chapitre 2. Néanmoins, cette technique n'est pas largement utilisée actuellement, et n'a été implémentée que dans certains cas particuliers, comme la gestion de la qualité de service. Il existe donc toujours d'importantes lacunes dans ce domaine.

Deuxième partie

Développement d’un outil de configuration BGP

Présentation de l’outil

Dans la première partie de ce travail, nous avons fait le point sur les techniques de gestion des réseaux. Nous en avons conclu que, malgré quelques avancées consistant à utiliser des langages de haut niveau pour une configuration globale, il restait de nombreux efforts à faire afin d’obtenir des outils efficaces permettant de minimiser voire supprimer les erreurs de configuration. Nous allons donc dans cette seconde partie tenter de concevoir une telle solution, dans le cadre particulier de la configuration BGP d’un AS de transit. Cet outil consistera en un langage pour exprimer les politiques de routage de l’AS, un compilateur pour les traduire en fichiers de configuration, et enfin, un module de simulation permettant de choisir une topologie iBGP efficace. Une fois les fichiers de configuration obtenus, ils pourront être appliqués aux routeurs du réseau via un protocole tel que NETCONF, qui a été présenté au chapitre 1.

Le choix du protocole BGP comme cadre de ce travail n’est pas anodin : il s’agit en effet du protocole responsable de l’échange inter-domaines des routes au niveau de l’Internet. Des études ont montré l’existence de nombreuses erreurs dans les configurations des routeurs BGP, et les conséquences importantes que ces dernières avaient au niveau de la connectivité Internet [9]. Ces erreurs sont souvent dues à la distraction d’opérateurs configurant leurs appareils de manière manuelle, en utilisant par exemple la ligne de commande. Un outil permettant d’obtenir de manière automatique des configurations BGP correctes serait donc à même d’améliorer la qualité du routage Internet.

Nous allons donc dans un premier temps présenter le protocole BGP en lui-même, avant d’aborder la conception du langage qui permettra de définir des politiques de routage et le compilateur permettant de générer les configurations correspondantes. Le chapitre suivant présentera quant à lui le module d’optimisation de la configuration iBGP. Enfin, pour chacune de ces deux étapes, nous présenterons une étude de cas consistant à appliquer l’outil ainsi construit au réseau GEANT.

Chapitre 4

Introduction à BGP

BGP, ou *Border Gateway Protocol*, est un protocole de routage créé en 1989 afin d'apporter une solution efficace au routage inter-domaines dans l'Internet à une époque où la croissance de ce dernier rendait inefficaces les solutions trop simplistes qui avaient été déployées jusque là. Sa quatrième version est définie par [10].

Ce protocole se base sur une division d'Internet en domaines appelés AS, ou *Administrative System*. Un AS est défini comme étant l'ensemble des routeurs réunis sous le contrôle d'une même entité administrative. Le routage BGP est alors scindé en deux parties : l'une, dénommée eBGP, concerne le routage inter-domaines, c'est-à-dire la transmission de routes entre AS, et l'autre, dénommée iBGP, s'occupe du routage intra-domaine, c'est-à-dire la diffusion des routes apprises des voisins eBGP à tous les routeurs de l'AS. La diffusion des routes internes à l'intérieur de l'AS n'est quand à elle pas gérée par BGP, mais par un autre protocole dédié au routage IGP (*Interior Gateway Protocol*).

Afin de clarifier la suite du travail, nous allons commencer par présenter BGP en trois étapes : Après avoir décrit les mécanismes du protocole lui-même, nous aborderons certains concepts et problèmes liés à iBGP. Nous introduirons en dernier lieu quelques techniques couramment utilisées pour la gestion BGP d'un AS. Ce chapitre se base essentiellement sur White [11] et Zhang [12].

4.1 Présentation du protocole

4.1.1 Routage par vecteurs de chemin

Un protocole de routage par vecteurs de chemin tel que BGP fonctionne sur base de l'enregistrement de chaque routeur par lequel transite un paquet d'information de routage. Ainsi, lorsqu'un routeur désire annoncer une de ses routes à ses voisins BGP, il émet un message *Update* contenant le préfixe de destination de la route, ainsi que le numéro qui l'identifie en tant qu'AS. Chacun de ses voisins enregistrera cette route avant d'éventuellement l'annoncer à ses propres voisins en ajoutant son numéro d'AS à celui du routeur d'origine, formant ainsi un *AS Path*, et ainsi de suite jusqu'à ce que la route soit connue dans tout le réseau.

L'exemple de la figure 4.1 montre l'échange d'une route BGP annonçant le réseau 1.0.0.0/8. La route est initiée par le routeur 1.0.0.1, qui l'annonce à son deux voisins. Le routeur 2.0.0.1 introduit alors cette destination dans sa table de routage, et annonce ensuite cette même route à son voisin de l'AS 3.

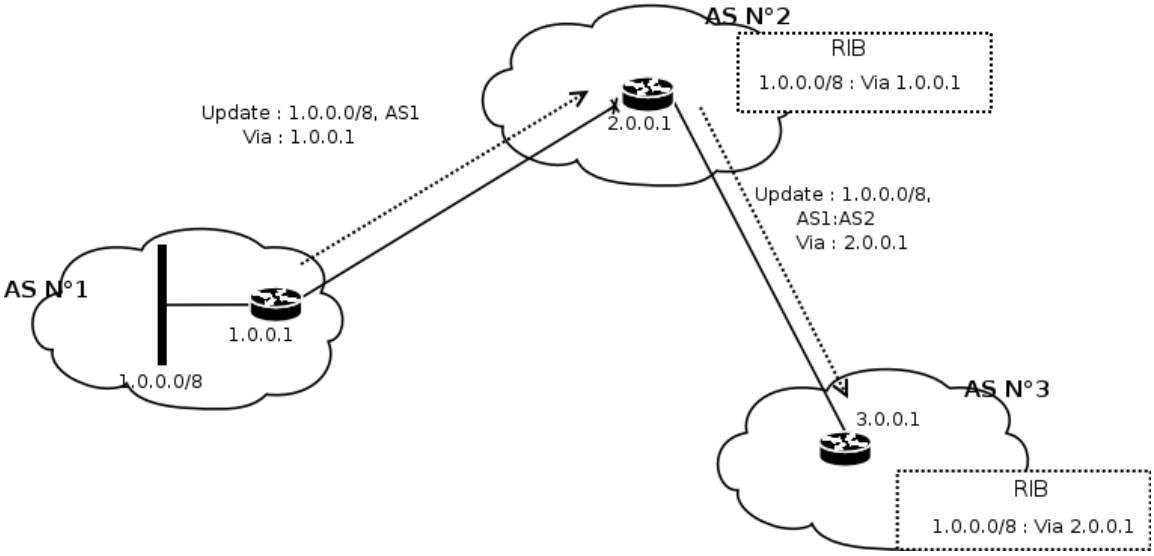

FIG. 4.1 – Échange d'une route BGP

L'*AS Path* permet de garantir l'absence de boucles de routage, puisqu'un routeur peut détecter ces dernières en vérifiant la présence de son numéro d'AS dans l'*AS Path* des

routes qu'il reçoit. Il lui suffit alors de ne pas redistribuer la route en question pour éviter toute boucle de routage. Le nombre de numéros d'AS dans l'*AS Path* sert également d'indication quant à la distance en terme de nombre de routeurs à traverser entre le routeur courant et la destination.

4.1.2 Architecture

Deux interlocuteurs BGP désirant échanger des routes doivent établir entre eux une session BGP avant d'entamer l'échange en lui-même. Une telle session s'établit au dessus de TCP. Les messages qui seront alors échangés sont essentiellement de deux types : Soit il s'agit de messages *Update* servant à annoncer de nouvelles informations de routage, soit il s'agit de messages *Withdraw*, pour retirer explicitement des routes devenues injoignables. L'échange est incrémental : tant que la connexion reste active, un message est envoyé dès qu'une route est modifiée, remplaçant dès lors implicitement tout message précédent concernant la même destination. Ainsi, si la réception d'une nouvelle route par un voisin entraîne une modification de la meilleure route vers cette destination, des messages *Update* seront alors envoyés aux autres voisins pour les informer de ce changement.

Lorsqu'une route est reçue d'un voisin, elle est stockée dans une base de données interne regroupant toutes les routes reçues de ce voisin et appelée *Adj-Rib-In*. Il y a donc autant d'*Adj-Rib-Ins* que de voisins BGP. Chaque *Adj-Rib-In* est ensuite filtrée de manière à appliquer des politiques d'importation aux routes reçues, avant de voir ses routes soumises à un processus de décision choisissant parmi toutes les routes menant à un préfixe donné celle qui est estimée la meilleure selon une série de critères spécifiques.

Les meilleures routes résultant de ce processus sont ensuite stockées dans la table de routage du routeur (*Routing Information Table*, ou *RIB*). La dernière étape du processus BGP consiste alors à annoncer les meilleures routes sélectionnées par le routeur aux voisins BGP de celui-ci, après application de filtres implémentant ses politiques d'exportation. Les routes destinées à être envoyées sont stockées dans les *Adj-Rib-Outs*, chacune correspondant à un des voisins BGP.

4.1.3 Les attributs de routes BGP

En plus de l'*AS Path* introduit ci-dessus, une route peut être caractérisée par différents attributs qui influencent le processus de sélection lorsque les routeurs doivent choisir entre plusieurs routes vers une même destination.

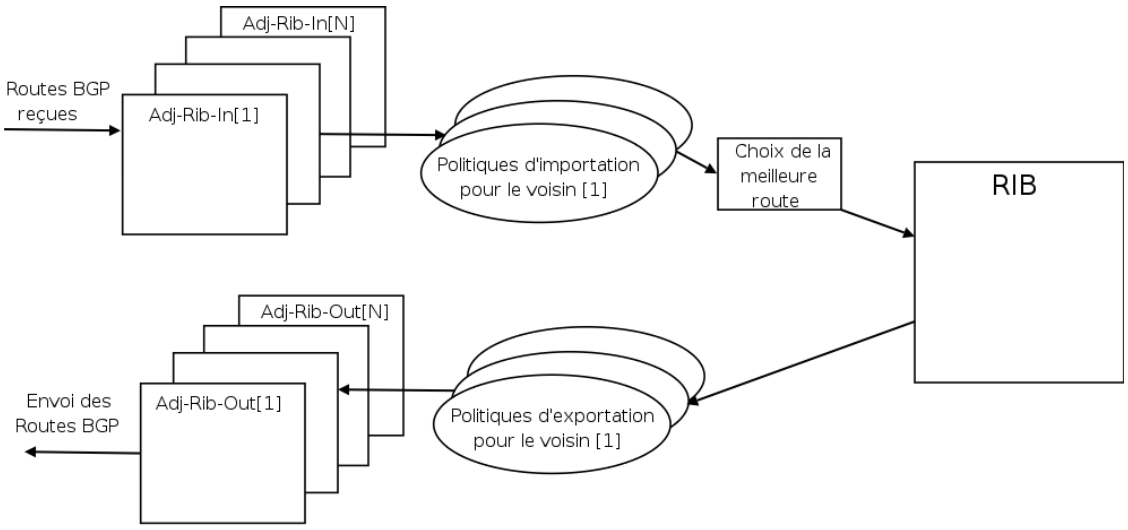

FIG. 4.2 – Structure d'un routeur BGP

L'attribut *Local-Pref* permet ainsi d'indiquer un niveau de préférence de routes interne à l'AS : un routeur choisira systématiquement la route possédant le *Local-Pref* le plus élevé. Cet attribut n'est pas transitif, ce qui signifie qu'il n'est utilisé qu'à l'intérieur de l'AS et n'est pas transmis aux AS voisins lors de l'échange de la route.

Un autre de ces attributs consiste en un *code d'origine* : Il permet de savoir si une route est originaire d'un protocole IGP, d'un protocole EGP autre que BGP ou encore d'une source inconnue qui sera caractérisée par la valeur *INCOMPLETE*.

L'attribut *Next-Hop*, quant à lui, est un élément indispensable au bon fonctionnement du routage : En effet, c'est lui qui indique le routeur vers lequel il faut envoyer les paquets afin qu'ils arrivent à destination. Le *Next-Hop* d'une route est modifié chaque fois qu'une route est annoncée sur une session inter-domaines (eBGP). Il peut soit indiquer le routeur qui a annoncé la route, soit un autre vers lequel on désire dévier le trafic.

Enfin, le *MED* (*Multiple Exit Discriminator*) est quant à lui un attribut optionnel, permettant à un administrateur d'indiquer à un voisin un point d'entrée préférentiel à l'intérieur de son AS. Il est donc transitif, contrairement au *Local-Pref*. Ainsi, lorsqu'un voisin reçoit de cet AS deux routes vers un même préfixe, il choisira celle ayant le *MED* le moins élevé. Par défaut, les *MEDs* venant d'AS différents ne sont pas comparés, mais comme

cette pratique peut entraîner des oscillations de routes dans certains cas, il est parfois préférable de comparer systématiquement tous les *MEDs*, quelle que soit leur origine (option *always-compare*).

Les derniers attributs que nous aborderons ici sont les *communautés* [13]. Ce sont des attributs optionnels et transitifs, ce qui signifie qu'ils sont transmis d'AS en AS, à moins d'être explicitement retirés par un routeur. Ils n'influencent pas le processus de décision directement, mais représentent une caractéristique commune à un ensemble de routes. Une route peut être caractérisée par plusieurs communautés. Via des mécanismes de filtrage à l'entrée ou à la sortie d'un routeur, une communauté peut donner lieu à une action, comme l'attribution d'un *Local-Pref*, ou encore l'exclusion ou l'acceptation d'une route. Des exemples d'utilisation de cet attribut seront présentés à la section 4.2.2. Enfin, en pratique, une communauté est représentée par 32 bits, dont les 16 premiers représentent en général le numéro de l'AS qui l'a attribuée à une route.

Il existe également des communautés prédéfinies : Il s'agit des communautés *No-Export*, *No-Export-Subconfed* et *No-Advertise*. La première et la seconde spécifient qu'il ne faut pas annoncer ces routes à des voisins eBGP, la seconde incluant dans cette catégorie les AS membres de la même confédération BGP que l'AS local (les confédérations sont définies par la RFC 3065 [14]). Enfin, la dernière communauté indique qu'il ne faut annoncer cette route à aucun voisin BGP.

4.1.4 Algorithme de choix de la meilleure route

L'algorithme standard pour la sélection de la meilleure route (*Decision Process*) effectue en premier lieu son choix sur base des critères mentionnés ci-dessus, à savoir le *Local-Pref*, la longueur de l'*AS Path* et le *MED*, afin d'appliquer les politiques mises en oeuvre par les gestionnaires via des filtres d'entrée. S'il reste plusieurs routes en compétition à l'issue de ce premier choix, d'autres critères seront alors examinés, se basant quant à eux sur la topologie de l'AS. On considère tout d'abord le type de la session BGP sur laquelle la route a été apprise : une route apprise via eBGP sera ainsi préférée à une route apprise via iBGP. Ensuite, la route dont le point de sortie de l'AS est le plus proche en termes de coût IGP sera sélectionnée. Enfin, s'il reste toujours plusieurs routes candidates, l'une d'elles sera choisie arbitrairement sur base par exemple de l'identifiant du routeur l'ayant annoncée.

4.2 Techniques utilisées pour la configuration BGP

Il a été mentionné plus haut que les attributs des routes BGP permettaient au gestionnaire d'influencer l'annonce et la réception des routes dans l'AS, ainsi que le processus de sélection des routes. Nous allons donc à présent détailler quelques-unes des techniques utilisées à ces fins. Ces techniques se basent sur l'utilisation de filtres, permettant d'isoler un ensemble de routes à partir d'une ou de plusieurs caractéristiques communes avant d'effectuer des actions sur cet ensemble.

4.2.1 Définition de filtres

Il existe de nombreuses manières d'exprimer des filtres, en fonction de l'implémentation de BGP déployée dans les routeurs. Nous allons donc nous contenter de mentionner de manière générale quelques types de filtres existants, indépendamment de toute syntaxe particulière. Dans cette optique, un filtre est composé de deux éléments : Un ensemble de conditions, et un ensemble d'actions.

Les conditions servent à isoler certaines particularité des routes. Ces particularités peuvent aussi bien être au niveau du préfixe de destination qu'au niveau des communautés caractérisant les routes. En fonction de l'expressivité disponible dans le routeur, la condition peut s'appliquer sur base d'un préfixe ou d'une communauté particulière, ou encore être définie par des expressions régulières. Enfin, les conditions sont généralement combinables entre elles via des opérateurs logiques afin d'obtenir d'autres conditions plus complexes.

Les actions applicables aux ensembles de routes sélectionnées consistent soit à accepter ou refuser l'importation ou l'exportation de ces routes, soit à modifier certains de leurs attributs. Les attributs modifiables sont typiquement les communautés, le *Local-Pref* ou le *MED*.

Les filtres sont appliqués à l'entrée et à la sortie d'un routeur, c'est-à-dire aux routes contenues dans les *Adj-Rib-Ins* et *Adj-Rib-Outs*. L'exemple suivant présente un filtre permettant d'attribuer un *Local-Pref* de 100 aux routes apprises d'un voisin BGP identifié par l'adresse IP 1.0.0.0, et un autre servant à empêcher le routeur d'annoncer les routes possédant la communauté 1 à ce même routeur. Le premier s'applique donc aux routes de l' *Adj-Rib-In* du routeur 1.0.0.0, et l'autre à celles de l'*Adj-Rib-Out* de ce même routeur. La syntaxe utilisée est celle du simulateur C-BGP qui sera utilisé plus loin [15].

```
peer 1.0.0.0
```

```
filter in
  add-rule
    match any
    action "local-pref 100"
    exit
  exit

filter out
  add-rule
    match "community is 1"
    action deny
    exit
  exit
```

4.2.2 Exemples d'utilisation de filtres

Les possibilités d'utilisation de ces filtres sont infinies, et les politiques qu'il est possible de mettre en oeuvre à partir de là sont donc très variées. Nous allons en décrire ici quelques-unes parmi les plus courantes, qui seront ensuite utilisées dans la suite de ce travail.

Modèle *Client*/*Pair*/*Fournisseur*

Internet est composé de plusieurs catégories d'AS, formant entre eux une sorte de hiérarchie basée sur les accords commerciaux passés entre les différents voisins BGP. Ces catégories peuvent être regroupées en un seul modèle définissant deux types de relations entre AS [16] : une première relation modélise un accord commercial spécifiant qu'un AS paye un autre afin que ce dernier se charge d'acheminer le trafic qui lui est destiné. Le premier AS est appelé *Client*, et le second *Fournisseur*. La seconde relation existe entre deux AS ayant en commun un lien dont les frais sont partagés par les deux deux parties, et chacune d'elles accepte d'annoncer à l'autre les routes menant à ses clients. Les deux AS sont alors appelés *Pairs*. Comme le montre la figure 4.3, ce modèle définit une hiérarchie entre les AS, puisqu'un fournisseur peut lui-même être client d'un autre fournisseur, tout en ayant des relations à frais partagés avec des pairs.

En pratique, l'implémentation de ce modèle implique pour un AS d'annoncer les routes d'un client à tous les voisins BGP, et de transmettre à ce dernier toutes les routes qui ont été apprises, tandis qu'il n'annoncera à un pair ou à un fournisseur que ses propres routes ainsi que celles de ses clients. De la sorte, on évite de faire transiter du trafic

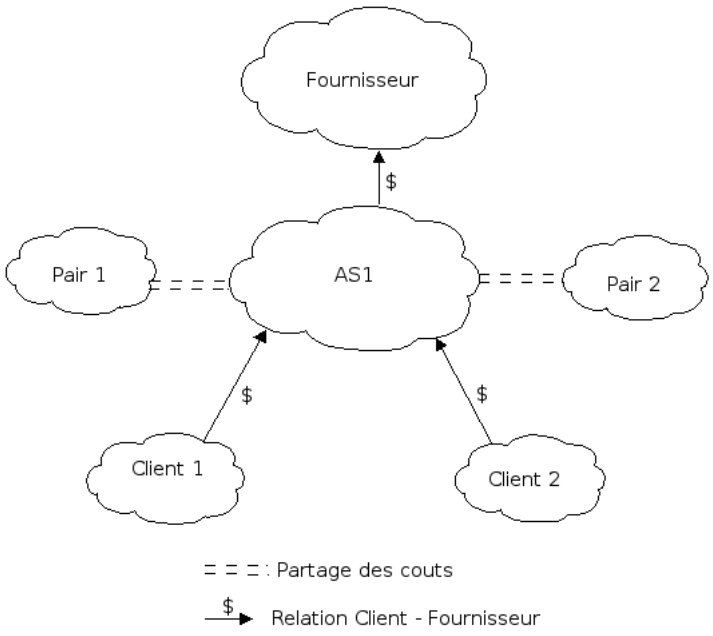

FIG. 4.3 – Relations entre AS

indésirable venant d'un pair ou d'un fournisseur et dont la destination n'est ni l'AS local, ni un client qui, lui, paye pour recevoir le trafic qui lui est destiné. De plus, afin de minimiser les frais d'acheminement du trafic, les routes des clients seront systématiquement préférées aux routes venant de pairs, ces dernières étant elle-même préférées à celles des fournisseurs. Cette dernière pratique permet en outre de garantir la convergence inter-domaines de BGP [17].

Pour ce faire, il sera donc nécessaire d'appliquer des filtres afin de limiter les annonces de routes vers les pairs et les fournisseurs et d'instaurer des préférences entre les routes reçues des différents voisins. La première étape peut être effectuée en attribuant une communauté à chaque type de relation commerciale afin d'identifier l'origine des routes. Ainsi, lorsqu'un routeur doit annoncer des routes à un fournisseur ou à un pair, il ne sélectionnera que celles possédant la communauté identifiant des routes venant de clients. La deuxième étape consistera simplement à attribuer un *Local-Pref* plus élevé aux clients qu'aux pairs, et plus élevé aux pairs qu'aux fournisseurs.

Sélection des routes valides

Parmi les préfixes annoncés dans l'Internet, il en existe un certain nombre qui ne correspondent pas à des destinations valides. Ces destinations appartiennent généralement à des groupes d'adresses IP correspondant à des adresses privées (comme par exemple les adresses comprises dans 192.168.0.0/16, 172.16.0.0/12 ou 10.0.0.0/8), des adresses Multicast (224.0.0.0/5), ou encore des adresses non allouées [11]. Une liste reprenant ces adresses peut être trouvée dans [18]. Il est également courant de limiter le nombre et la longueur des préfixes reçus d'un voisin BGP, et ce afin d'éviter une surcharge des tables de routage [19].

Communautés annoncées par des voisins BGP

Il arrive fréquemment que des fournisseurs d'accès permettent à leur client d'utiliser des communautés particulières afin d'influencer la manière dont leurs routes seront exportées en dehors de l'AS. Cette pratique, décrite dans [20], nécessite que le client et le fournisseur se mettent d'accord sur un ensemble de communautés qui seront définies à cette fin. Une communauté peut ainsi indiquer à un AS de ne pas exporter ces routes vers tel ou tel fournisseur d'accès, ou encore de l'annoncer, mais après avoir effectué de l'*AS-Path prepending*, technique qui consiste à ajouter plusieurs fois le numéro de l'AS local à la fin de l'*AS-Path* d'une route afin d'influencer la sélection des routes par les routeurs qui la recevront ultérieurement. Enfin, ces communautés peuvent également déclencher l'attribution d'un *Local-Pref* particulier à ces routes par le routeur qui la reçoit.

4.3 iBGP

Maintenant que les bases du protocole ont été présentées, nous allons nous attarder quelque peu sur le déploiement de BGP au niveau intra-domaine, c'est-à-dire iBGP. La caractéristique principale de cette partie du protocole est que les routeurs qu'elle concerne sont tous reliés les uns aux autres par des sessions iBGP, créant ainsi une topologie de type *Full Mesh*. Une session BGP entre deux routeurs d'un même AS n'a alors pas le même comportement qu'une session entre deux routeurs d'AS différents. Ainsi, une route apprise sur une session iBGP ne sera normalement pas retransmise à d'autres voisins iBGP. Cette pratique permet d'éviter les boucles de routages dans l'AS, puisque, lors d'un échange iBGP, l'*AS PATH* n'est pas modifié, et il n'est donc pas possible pour un routeur de reconnaître une route qu'il reçoit pour la seconde fois. De même, afin de garantir une sélection de routes cohérente à l'intérieur de l'AS, les attributs des routes apprises via iBGP ne sont pas modifiés. Une route n'aura alors son *AS Path* complété avec le numéro de l'AS local que lorsqu'un routeur l'annoncera sur une session eBGP.

Il en est de même pour le *Next-Hop*, qui reste inchangée tout au long de son parcours dans l'AS, et qui sera changé par l'adresse IP du routeur par lequel la route sort de l'AS.

4.3.1 Problématique du *Full Mesh*

Dans une topologie de type *Full Mesh*, chaque routeur possède une session iBGP vers les autres routeurs du même domaine, ce qui n'est guère efficace en terme d'utilisation des ressources et peut conduire à d'importants problèmes d'échelle lorsque l'AS possède un grand nombre de routeurs. Le nombre de sessions dans un AS à n routeurs est en effet de $\frac{n(n-1)}{2}$, et augmente donc en $O(n^2)$.

4.3.2 Les réflecteurs de routes

Deux alternatives ont été proposées pour résoudre la problématique du *Full Mesh* : Les confédérations [14] et les réflecteurs de routes [21]. Nous ne nous intéresserons dans ce travail qu'à cette seconde solution, et n'aborderons pas les confédérations. Ces dernières sont en effet beaucoup moins répandues, et sont surtout déployées dans le cadre d'une fusion entre AS. Après avoir présenté les caractéristiques des topologies iBGP avec réflecteurs de routes, nous mentionnerons certains problèmes qui découlent de leur utilisation.

Présentation des réflecteurs de routes

Le principe de la réflection de routes consiste à introduire une hiérarchie entre les routeurs iBGP de l'AS. Deux groupes de routeurs iBGP sont définis : les réflecteurs de routes et les clients. Chaque client est lié à un réflecteur de routes, et un réflecteur de routes peut être lui-même client d'un autre réflecteur de routes. Comme le montre la figure 4.4, cette pratique permet de réduire nettement le nombre de sessions iBGP nécessaires.

Les règles de diffusion iBGP sont alors modifiées pour correspondre à cette nouvelle architecture. Un réflecteur de routes aura alors le comportement suivant : Il annonce à tous ses clients une route reçue d'un routeur non-client, et annonce aussi bien aux clients qu'aux non-clients les routes reçues d'un de ses clients.

Alors que les réflecteurs de routes du sommet de la hiérarchie seront toujours reliés selon un *Full Mesh* entre eux et avec les routeurs non-clients du même niveau, il existe deux possibilités pour les clients : Soit ils ne possèdent de sessions iBGP qu'avec leur réflecteur de routes, soit les clients d'un même réflecteur maintiennent un *Full Mesh* de

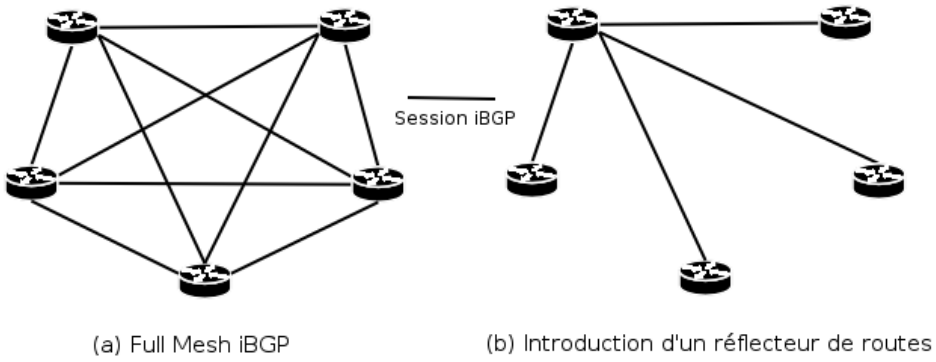

FIG. 4.4 – Comparaison d'une topologie *Full Mesh* (a) avec une topologie à un réflecteur (b)

sessions iBGP entre eux. Dans la suite de ce document, nous ne travaillerons qu'avec des topologies du premier type, c'est-à-dire sans sessions iBGP entre des clients.

Inconvénients liés à l'introduction de réflecteurs de routes dans une topologie

L'introduction d'un réflecteur de routes dans un AS n'est pas sans influencer le contenu des tables de routage de chacun des routeurs. En effet, avec un *Full Mesh*, chaque routeur reçoit l'ensemble des routes et peut donc effectuer sa propre sélection de la meilleure route vers une destination donnée. Lorsqu'on ajoute un réflecteur de routes, ce dernier va effectuer son choix avant de transmettre la route qui en résulte à ses clients. Ces derniers ne recevront donc que la route que le réflecteur a jugé meilleure, et, comme le choix de ce dernier peut être différent du choix des clients, on aura donc au final des tables de routage différentes de celles qu'on aurait obtenues avec un *Full Mesh*.

La différence de choix entre les différents routeurs vient de ce que chacun d'eux cherche à choisir le point de sortie du réseau le plus proche en fonction des coûts IGP, et ce point de sortie peut différer entre un réflecteur de routes et ses clients. Certains clients auront donc une route non optimale vers une destination, c'est-à-dire que les paquets qu'ils transmettront n'emprunteront pas le point de sortie le plus proche possible, et cela résulte en un accroissement du trafic puisque les paquets restent plus longtemps dans l'AS.

Il y a un second problème lié à l'utilisation des réflecteurs de routes. En effet, ces derniers ne transmettent à leurs clients que leur meilleure route vers un préfixe. Si cette route est invalidée, les clients se retrouvent alors sans possibilité de rejoindre ce préfixe,

du moins tant que le réflecteur n'a pas choisi une nouvelle route et retransmis cette dernière à ses clients.

Enfin, comme nous le verrons plus loin, les réflecteurs de routes sont connus pour empêcher dans certains cas la convergence du protocole, et peuvent parfois provoquer des déflections voire même des boucles lors de la transmission des paquets. Ces boucles peuvent être dues à l'utilisation de l'attribut *MED* dans le choix des meilleures routes [22], ou bien sont le résultat de certaines particularités de la topologie iBGP modifiée. Enfin, la réduction du nombre de sessions rend l'AS beaucoup plus vulnérable aux pannes de liens.

Chapitre 5

Langage de définition de politiques

5.1 Introduction

Actuellement, les configurations BGP d'un AS se font souvent de manière manuelle, technique pour laquelle il est difficile de garantir aussi bien la cohérence des politiques mises en oeuvre et du système que l'absence d'erreurs de configuration [9, 23]. Il existe quelques outils [24] permettant d'implémenter des politiques de routage de manière automatisée, se basant généralement sur RPSL [25], qui est un langage de définition de politiques d'importation et d'exportation de routes. Les politiques ainsi exprimées sont dès lors analysées et traduites en fichiers de configuration pour chacun des routeurs de l'AS, garantissant alors la cohérence et l'absence de conflit entre elles. Cela permet notamment d'éviter les erreurs de configuration qui résultent en un non-respect des politiques de routage, comme par exemple des erreurs dans les filtres d'exportation provoquant l'annonce de routes qui n'auraient pas du l'être. Le travail des gestionnaires de l'AS se voit donc allégé, leur permettant alors de se concentrer essentiellement sur la définition des politiques qu'ils désirent utiliser puisque ces dernières sont mises en oeuvre automatiquement.

Le langage que nous proposons dans ce chapitre a également pour but d'automatiser la mise en oeuvre de politiques BGP, mais, contrairement à RPSL, il permet plus de richesse dans l'expression de ces politiques en se basant non pas sur une granularité par AS, mais plutôt par session eBGP. Il utilise également le concept de *classe*, qui permet aux gestionnaires du réseau de travailler à un niveau d'abstraction plus élevé qu'avec RPSL. De plus, alors que les outils existants se contentent d'implémenter directement les politiques d'importation ou d'exportation de routes, l'outil que nous allons concevoir dans le cadre de ce mémoire effectuera en plus une optimisation de la configuration iBGP de l'AS sur base des routes qu'il reçoit.

Nous allons dans un premier temps décrire les principes sous-jacents au langage que nous proposons, avant de le définir de manière plus formelle. Nous exposerons ensuite la manière dont d'éventuels conflits entre politiques sont réglés, avant de finalement présenter un exemple d'expression de politiques dans le langage qui a été défini.

5.2 Spécifications du langage

Le langage qu'il nous faut définir doit permettre d'exprimer des politiques de routage BGP. La question qui se pose dès lors est : qu'est-ce qu'une politique BGP ? Il s'agit en fait d'un ensemble de règles dont le but est de contrôler le trafic entrant et sortant d'un AS en jouant sur l'importation et l'exportation des routes échangées via BGP avec les AS voisins. Un des exemples les plus connus de politique BGP est le modèle *Client*/*Pair*/*Fournisseur*, présenté au chapitre 4. Notre langage devra non seulement être capable d'exprimer ce modèle, mais également n'importe quelle autre politique d'importation ou d'exportation de routes.

Si on considère qu'une route vers une destination donnée est annoncée dans l'AS aux points d'entrées formés par un sous-groupe de routeurs frontières, une politique influencera alors la manière dont cette route sera transmise aux autres routeurs frontières du réseau, qui seront les points de sortie de la route considérée. Une politique peut alors se modéliser comme un graphe orienté dont les noeuds sont les sessions eBGP entre l'AS et ses voisins, et dont les arcs représentent la transmission d'une route d'une session eBGP à une autre. Par exemple, dans le cas du modèle *Client*/*Pair*/*Fournisseur*, une session eBGP de type *Client* aura des arcs en direction de toutes les autres sessions eBGP, tandis qu'une session de type *Fournisseur* ou *Pair* n'aura des arcs qu'en direction des sessions eBGP de type *Client* (voir figure 5.2). La modélisation ainsi définie présente néanmoins certaines simplifications. Ainsi, les arcs du graphe induisent une relation de type binaire entre les sessions eBGP, c'est-à-dire qu'il y a transmission de la route ou non. Il est possible d'ajouter une plus grande expressivité au langage en ajoutant la notion de priorité aux sessions eBGP. Ainsi, une route venant d'une session cliente aura une plus grande priorité qu'une route annoncée par un pair, ce qui aura pour conséquence qu'elle sera choisie préférentiellement à cette dernière.

5.3 Définition des politiques

Le langage se compose essentiellement de deux types d'éléments : des *classes* et des *types de classes*. Une classe définit un type de session eBGP ayant un comportement d'annonce et de réception de routes donné, tandis qu'un type de classes exprime les

relations qui peuvent exister entre classes. Nous allons tout d'abord présenter la syntaxe pour définir une classe et les règles qui lui sont appliquées, avant d'aborder le concept de type de classes. Nous montrerons enfin comment attribuer les classes aux différentes sessions eBGP.

Une classe représente un ensemble de sessions eBGP possédant une ou des caractéristiques communes, et dont les politiques d'annonce et de réception de routes sont semblables. Ainsi, dans le modèle *Client*/*Pair*/*Fournisseur*, chacune de ces trois catégories sera représentée par une classe distincte. On peut ainsi, de la même manière, définir une classe *Europe*, qui s'appliquerait à toutes les sessions eBGP reliant l'AS considéré à un voisin européen. Cela permet par exemple de fournir un service limité géographiquement à un client qui ne payerait que pour avoir accès à des destinations européennes. Notons qu'une session eBGP peut être caractérisée par différentes classes. La syntaxe pour déclarer une ou plusieurs classes est la suivante :

```
Class {<classname>[,<classname>]*}
```

Une fois l'ensemble des classes défini, il faut exprimer les règles qui régissent la transmission des routes à travers l'AS. Si l'on reprend l'image du graphe présentée plus haut, il s'agit donc de définir les arcs reliant chacun des noeuds. La syntaxe correspondante est la suivante :

```
define rule for <classname> :{
        priority is <value>
        AnnounceTo {[<classname>,]*<classname>}| ALL | NONE
        acceptFrom {[<classname>,]*<classname>}| ALL | NONE
}
```

Les arguments <classname> doivent bien entendu avoir été définis en tant que classes avant d'exprimer des règles les concernant. Si aucune règle n'est spécifiée, la règle par défaut sera de ne pas annoncer ni accepter de routes. Quant à la ligne commençant par 'priority', elle définit la priorité propre à la classe. Cette priorité est relative, et s'exprime par une valeur représentant un décalage par rapport à un niveau de référence. Pour simplifier l'implémentation du prototype, cette valeur sera pour le moment un entier compris entre -9 et +9, et le niveau de référence sera 0. Ainsi, pour calculer la priorité à assigner

aux routes annoncées sur une certaine session, il suffira d'additionner l'ensemble des priorités des classes assignées à cette session pour obtenir la valeur totale. Cette valeur sera alors transformée en une valeur de *Local-Pref* qui sera assignée à toutes les routes annoncées à ce point d'accès particulier.

5.4 Gestion des conflits et types de classes

Le problème du langage ainsi défini est que, si une session eBGP est caractérisée par plusieurs classes, il risque d'y avoir des conflits entre les règles régissant son comportement. Ainsi, si une session fait partie de la classe *Fournisseur* et d'une classe *Restricted-ToEurope* limitant l'accès d'une session aux seuls AS européens, doit-on annoncer ses routes à un fournisseur ne se situant pas en Europe ? Logiquement, ce ne doit pas être le cas. Il faut donc trouver un critère permettant de solutionner le problème en définissant le comportement d'une session eBGP appartenant à plusieurs classes. Pour cela, nous allons d'abord définir ce que devrait être ce comportement.

- Certaines classes sont mutuellement exclusives. Ainsi, une session eBGP ne peut être à la fois de classe *Client* et de classe *Fournisseur*.
- Certaines classes ont des règles qui peuvent éventuellement définir des sessions eBGP cibles mutuellement exclusives. Une session caractérisée par de telles classe doit sélectionner ses sessions-cible de manière à ne contredire aucune règle.

La solution proposée pour répondre aux contraintes que nous venons d'exprimer est de simplement rajouter un attribut supplémentaire à chacune des classes, attribut que nous nommerons *type de classes*, avec comme règle qu'une session eBGP ne peut se voir assigner deux classes du même type. Il reste néanmoins possible qu'une classe n'ait aucun type particulier, ce qui est d'ailleurs le comportement par défaut si rien n'est indiqué. Il n'y a dans ce cas aucune contrainte pour une session possédant une classe sans type. Pour ce qui est du traitement des sessions eBGP possédant plusieurs classes, la solution logique est de considérer comme ensemble de sessions eBGP cibles l'intersection des ensembles cibles définis par les règles de chacune des classes.

Pour définir un type de classe, il faut d'abord le déclarer :

```
Type {<typename>[,<typename>]*}
```

Ensuite, il faut définir les classes appartenant à un type donné, c'est-à-dire les classes mutuellement exclusives :

```
belongsTo <typename> : {<classname>[,<classname>]*}
```

5.5 Instanciation des classes

Une fois le comportement des classes défini, il faut instancier les politiques qui ont été ainsi exprimées à l'AS considéré. Pour cela, on se base sur des identifiants de sessions eBGP représentés par ¡session-id¿, et, pour chacune de ces sessions, on indique les classes qu'il faut leur attribuer :

```
Session <session-id> hasClass :{<classname>[,<classname>]*}
```

5.6 Exemple

Afin d'éclaircir un peu les concepts présentés tout au long de ce chapitre, nous allons présenter un petit exemple de mise en place de politiques dans un AS de taille relativement réduite, AS1. L'AS en question, ainsi que ses liens avec ses voisins sont représentés par la figure 5.1.

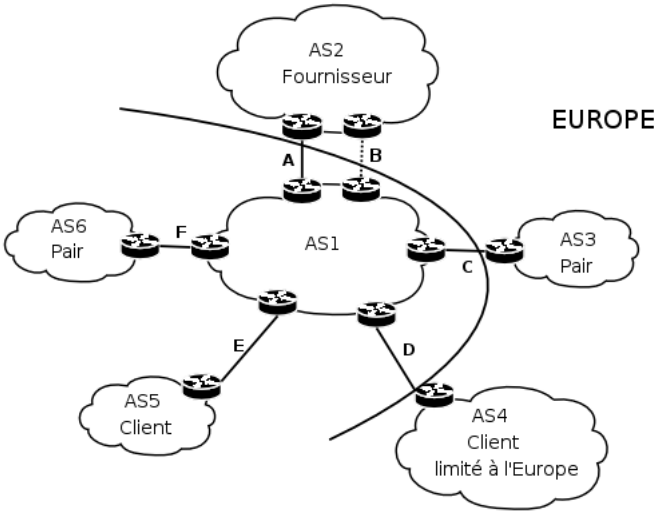

FIG. 5.1 – Relations entre l'AS 1 et ses voisins

Imaginons que les administrateurs de l'AS1 veuillent implémenter les politiques suivantes :

– AS2 est un fournisseur d'accès : AS1 lui annonce ses routes personnelles ainsi que les routes apprises par ses clients.
– Il y a deux liens avec AS2. AS1 aimerait que le lien identifié par B soit utilisé comme backup, c'est à dire qu'il ne soit utilisé que dans le cas où la session A tombe en panne.
– AS3 et AS6 sont des pairs : AS1 ne leur annonce que ses routes personnelles ainsi que celles apprises de ses clients.
– AS4 est un client : AS1 lui annonce toutes les routes qu'il connaît.
– AS5 est un client qui ne demande accès qu'à des destinations situées en Europe.
– AS1 préfère de manière générale les routes annoncées par des AS européens.

On peut modéliser ces politiques par un graphe tel que défini plus haut, avec les noeuds représentant les sessions eBGP et les arcs, la transmission de routes entre deux sessions. Les arcs en pointillés représentent des routes avec une priorité plus faible.

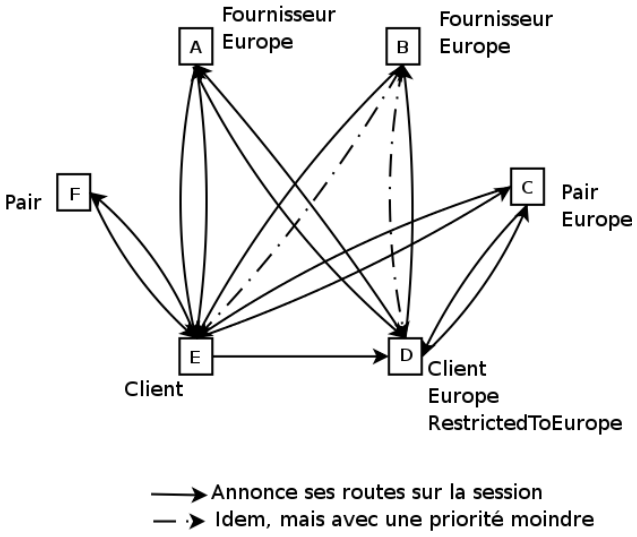

FIG. 5.2 – Représentation sous forme de graphes de l'échange des routes à travers l'AS

Pour écrire la configuration correspondant, on procède de la manière suivante :

1. Définition des classes et des types : nous allons définir les 6 classes suivantes, chacune correspondant à une caractéristique d'une session eBGP : *Client*, *Fournisseur*, *Pair*, *RestrictedToEurope*, *Europe*, *Backup*. Il nous faut également définir des types de classes pour exprimer le fait que certaines classes sont mutuellement exclusives. Ainsi, les classes *Fournisseur*, *Client* et *Pair* seront de type *CommercialPeering*. La classe *RestrictedToEurope* n'aura pas de type particulier, tandis que la classe *Europe* sera de type *Continent*, pour exprimer le fait qu'une session

eBGP ne peut appartenir géographiquement qu'à un seul continent. Cet attribut *Continent* n'aura pas réellement d'importance dans notre exemple puisque *Europe* est la seule classe de ce type que nous utilisons.

```
Class {Client, Fournisseur, Pair, Europe,
       RestrictedToEurope, Backup}
Type {CommercialPeering, Continent}

belongsTo CommercialPeering :
     {Client, Fournisseur, Pair}
belongsTo Continent : {Europe}
```

2. Définition des règles : En plus des règles de transmission de routes établies plus haut lors de la spécification des politiques, il nous faut déterminer les priorités de chacune des classes. Ainsi, la classe *Backup* aura une priorité négative pour traduire le fait qu'une session de cette classe ne doit être utilisée qu'en cas de panne du lien principal. Étant donné que le lien de backup de notre exemple est défini par rapport à un autre lien vers le même AS, les sessions correspondantes seront identiques au niveau des classes autres que celles spécifiant la relation de backup, et la valeur de la priorité à assigner n'a pas d'importance, du moment qu'elle est négative. Nous utiliserons donc une valeur de -4, bien que n'importe quelle autre valeur eût fait l'affaire. Il nous faut également définir des valeurs de priorité afin d'exprimer le fait que les routes annoncées par un client doivent être préférées à celles annoncées par un pair ou un fournisseur. Il nous faut cependant cette fois considérer les autres relations de priorité possibles afin de choisir une valeur ad-hoc. En effet, l'AS1 déclare préférer les routes européennes. Que faire s'il dispose d'une route annoncée par un client, et d'une autre route, identique mais annoncée par un pair européen ? S'il accorde plus d'importance à la prioritarisation des routes en fonction du type de peering qu'à celle déterminée par l'emplacement géographique, il accordera au groupe de classes correspondant au peering un décalage mutuel de priorité plus grand qu'au groupe de classes géographiques. Cela implique de choisir par exemple +9, 0 et -9 pour les clients, pairs et fournisseurs, et +5 pour la classe *Europe*.

```
define rule for Client{
        priority is +9
        announceTo {Client, Fournisseur, Pair}
        acceptFrom ALL
}
define rule for Fournisseur{
```

```
        priority is -9
        announceTo {Client}
        acceptFrom {Client}
}
define rule for Pair {
        priority is 0
        announceTo {Client}
        acceptFrom {Client}
}
define rule for RestrictedToEurope{
        announceTo {Europe}
        acceptFrom {Europe}
}
define rule for Europe{
        priority is +5
        announceTo ALL
        acceptFrom ALL
}
define rule for Backup{
        priority is -4
        announceTo ALL
        acceptFrom ALL
}
```

3. Instanciation des classes : Il s'agit ici de simplement attribuer à chaque session eBGP de l'AS un ensemble de classe reflétant ses caractéristiques propres.

```
Session A hasClass {Fournisseur, Europe}
Session B hasClass {Fournisseur, Europe, Backup}
Session C hasClass {Pair, Europe}
Session D hasClass {Client, Europe, RestrictedToEurope}
Session E hasClass {Client}
Session F hasClass {Pair}
```

4. Vérification des priorités : Il est possible, à partir de l'instanciation établie plus haut, de calculer manuellement les priorités de chacune des classes pour vérifier la cohérence de nos attributions. Pour cela, il suffit d'additionner les priorités de chacune des classes et de comparer les valeurs finales :

```
Session A : -9 + 5 = -4
Session B : -9 + 5 -4 = -8
Session C :  0 + 5 = 5
```

```
Session D : +9 +5 = +14
Session E : +9
Session F : 0
```

Les valeurs obtenues sont effectivement cohérentes avec la politique spécifiée.

5.7 Génération des configurations BGP

5.7.1 Structure du compilateur

Une fois les politiques exprimées, il s'agit de les traduire en fichiers de configuration pour chacun des routeurs de l'AS. Nous allons donc construire un compilateur qui prendra comme données le fichier des politiques, ainsi qu'une liste des sessions eBGP avec leurs caractéristiques respectives, à savoir adresses IP des routeurs situés aux deux extrémités, et numéro de l'AS voisin. Il produira en sortie la configuration BGP de l'AS. Ce compilateur fonctionnera en plusieurs étapes :

1. Analyse syntaxique du fichier des politiques.
2. Définition des classes et de leurs caractéristiques, et attributions de ces dernières aux sessions.
3. A partir de ces attributions, calcul, pour chacune des sessions, des sessions vers lesquelles elles transmettront leurs routes, et desquelles elles en recevront.
4. Élimination des conflits éventuels entre acceptation et annonce des routes.
5. Production de la configuration BGP avec les filtres correspondants.

Il sera implémenté en Java, et la configuration produite sera un script C-BGP définissant l'établissement des sessions BGP entre routeurs ainsi que les filtres destinés à contrôler l'échange des routes sur les sessions eBGP. C-BGP [26] est un simulateur BGP qui nous permettra d'étudier et de valider la configuration produite par le compilateur. Cette configuration, afin d'être fonctionnelle, devra être complétée par un script C-BGP définissant la topologie du réseau, à savoir les noeuds et liens internes de l'AS, ainsi que les noeuds et liens reliant ce dernier à ses voisins et les routes IGP des noeuds.

5.7.2 Analyse syntaxique et construction des objets

La syntaxe du fichier de définition des politiques est analysée à l'aide d'un parseur qui a été généré à l'aide de Javacc [27]. Ce parseur dispose d'une liste d'objets Java de classe SessionEbgp, et complétera les attributs de ces derniers en fonction des informations qu'il obtient au cours de l'analyse. Ainsi, il va construire pour chacune des classes un objet Classe, qui possédera dans ses attributs aussi bien la priorité qui lui correspond qu'une liste d'autres objets Classe vers lesquels il annonce et accepte des routes.

Enfin, il va attribuer à chacune des sessions une liste de classes selon les attributions définies dans le fichier d'entrée, après avoir vérifié que les contraintes de type sont bien respectées. Dans le cas contraire, il génère une exception et la compilation est interrompue.

5.7.3 Calcul des relations entre sessions

Une fois que les objets SessionEbgp ont reçu en attribut la liste des classes qui leur correspondent, il s'agit de construire le graphe des relations entre sessions tel que défini plus haut. En pratique, ces graphes ne seront pas réellement définis, mais ils permettent de mieux comprendre la démarche qui est mise en oeuvre. Ce qu'il nous faut déterminer, par contre, ce sont les ensembles de classes, ou de combinaisons de classes qui caractériseront les routes annoncées sur une session eBGP.

L'ensemble des combinaisons de classes caractérisant les routes annoncées sur une session eBGP est déterminé par deux contraintes : D'une part, la politique d'acceptation des routes de la session en question, et d'autre part, les politiques de distribution des sessions ayant reçu chacune des routes. Cela revient, dans le graphe des sessions, à vérifier les arcs qui relient chacune des sessions et à ne distribuer les routes d'une session à l'autre que si deux arcs de sens opposés existent.

L'exemple suivant peut permettre de mieux comprendre la raison de cette vérification :

```
defineRuleFor C1 : {
        announceTo C2
        acceptFrom C2,C3
}
defineRuleFor C2 : {
        announceTo C1
        acceptFrom C3
}
defineRuleFor C3 : {
        announceTo C2
        acceptFrom ALL
}
defineRuleFor C4 : {
        announceTo C5
        acceptFrom C5
}
```

```
defineRuleFor C5 : {
        announceTo C4
        acceptFrom C4
}
```

Si l'on considère les relations entre une session de type C1 et une session de type C2, on remarque que la première désire annoncer ses routes à la seconde, mais que cette dernière n'accepte pas les routes de la première. Il ne faut donc pas que cette route soit distribuée entre cette paire de sessions.

Formalisons un peu cette situation. Pour une session S, il nous faut déterminer de manière univoque les combinaisons de classes qui seront ou ne seront pas annoncées aux voisins eBGP. Une classe étant une caractéristique qu'une session possède ou ne possède pas, elle peut se représenter par un symbole propositionnel. Ainsi, une session possédant les classes C1, C2 et C3 sera caractérisée par la formule $C1 \wedge C2 \wedge C3$. Les routes annoncées à l'extérieur seront quant à elles représentées par une disjonction de conjonctions. Si nous reprenons l'exemple présenté plus haut, une session caractérisée par $C1 \wedge C2 \wedge C3$ annoncera les routes venant d'un ensemble de session caractérisée par $(C2 \vee C3) \wedge C3 \wedge (C1 \vee C2 \vee C3)$. Après distribution, nous obtenons :

$(C2 \wedge C3 \wedge C1) \vee (C2 \wedge C3 \wedge C2) \vee (C2 \wedge C3 \wedge C3) \vee (C3 \wedge C3 \wedge C1) \vee (C3 \wedge C3 \wedge C2) \vee (C3 \wedge C3 \wedge C3)$

Puisque $A \wedge A$ est équivalent à A et que $(A \wedge B) \vee A$ peut également être remplacé par A, nous obtenons après simplification la formule $C3$. Cela signifie que la session S accepte d'annoncer hors de l'AS toutes les routes originaires de sessions qui possèdent au minimum la classe $C3$. Cependant, comme mentionné plus haut, le fait que S accepte d'annoncer ces routes à l'extérieur ne signifie pas que les sessions dont elles sont originaires l'autorisent. En effet, une classe ne spécifiant pas l'une des classes de S dans sa politique 'announceTo' déterminera le comportement de toute session la possédant : aucune de ces dernières n'acceptera que S n'annonce ses routes à l'extérieur. Dans notre exemple, la classe C4 ne possède dans sa clause "'announceTo"' ni C1, ni C2, ni C3. La règle nous permettant de déterminer de manière univoque les routes qu'il faut annoncer à l'extérieur est donc : *Annoncer les routes venant de sessions possédant C3 mais pas C4*. Le compilateur va donc effectuer ce calcul afin de déterminer les règles d'annonce des routes pour chacune des sessions.

5.7.4 Génération de la configuration BGP

Avec les résultats du raisonnement présenté plus haut, il est possible de générer la configuration BGP demandée. Cette dernière sera répartie en 4 fichiers : Un pour la déclaration des routeurs BGP, un pour la déclaration des sessions eBGP, un pour la configuration d'*Full Mesh* iBGP temporaire, et enfin, un dernier contenant les filtres d'entrée et de sortie. Les trois premiers ne font qu'indiquer au simulateur C-BGP les paramètres de la topologie BGP du domaine, tandis que le dernier se base sur les règles calculées par le compilateur pour implémenter les politiques définies dans le langage. Nous allons donc nous concentrer uniquement sur la génération de ce dernier fichier. Pour chaque routeur, il spécifiera donc :

- En entrée : un filtre étiquetant toutes les routes apprises sur la session par des communautés correspondant aux classes de cette session. Ces communautés seront de forme *ASNum* :*class-number*, *class-number* étant un entier attribué arbitrairement à la classe. Des commentaires en début de script donneront la correspondance entre les noms de classe et les communautés correspondantes.
- En sortie : plusieurs filtres permettant d'identifier les routes qui doivent être annoncées à l'extérieur de l'AS à partir de cette session. Ces filtres sont calculés par le compilateur suivant les règles calculées selon la méthode expliquée plus haut. Le premier de ces filtres sera un filtre rejetant toutes les routes possédant une communauté représentant une classe refusant que la session courante n'annonce ses routes. Les filtres suivant autoriseront les routes caractérisées par une clause calculée à partir de la politique d'acceptation de la session courante.

Pour l'exemple présenté plus haut, les filtres de la session S seront :

```
filter in
    add-rule
        match any
        action "community add 11537:1,
                community add 11537:2,
                community add 11537:3"
        exit
    exit
filter out
    add-rule
        match "community is 11537:4"
        action deny
        exit
    exit
```

```
filter out
    add-rule
        match "community is 11537:3"
        action accept
        exit
    exit
```

En plus de ces filtres générés sur base des politiques, chaque session aura également un filtre de sortie supprimant toutes les communautés utilisées à l'intérieur de l'AS, afin que ces dernières ne soient pas propagées à l'extérieur.

5.8 Extensions possibles

Tel que défini plus haut, notre langage nous permet de contrôler quelles sont les routes qui sont acceptées dans l'AS, et de quelle manière elles sont annoncées aux voisins eBGP de ce dernier. Grâce à l'utilisation du *Local-Pref*, il permet également d'influencer le choix d'une route dans l'ensemble du domaine. Cependant, en pratique, la gestion BGP d'un AS ne se limite pas à cela. Il existe d'autres techniques d'ingénierie de trafic ou de gestion des routes permettant d'influencer la propagation et le choix de ces dernières, parfois même en dehors de l'AS lui-même. Ces techniques ont été présentées dans le chapitre 4.

Pour que notre outil fournisse également la possibilité aux administrateurs d'utiliser ces techniques, il est nécessaire d'étendre le langage pour y ajouter des fonctionnalités. Nous allons donc présenter ci-dessous quelques propositions d'extension du langage afin de permettre chaque fois l'utilisation d'une technique supplémentaire de gestion des routes ou d'ingénierie de trafic. Ces extensions se basent toujours sur le principe de l'attribution de classes à des sessions. La différence tiendra au fait qu'au lieu de définir des classes régissant la manière dont les routes sont distribuées, il s'agit de définir des classes 'spéciales' dont le comportement consiste à déclencher une action particulière au niveau des sessions concernées.

Afin de permettre à l'administrateur de définir une classe s'appliquant à l'ensemble des sessions sans devoir explicitement attribuer cette dernière à chacune d'entre elles, nous introduisons également la notion de *classe globale*. Lors de la déclaration des classes, au début de la configuration, il y a à présent deux possibilités : Soit définir une classe en temps que *classe normale*, soit en temps que *classe globale*. Cela se fait de la manière suivante :

```
Class {<classname>[,<classname>]*}
GlobalClass {<classname>[,<classname>]*}
```

Les classes appartenant à la liste *GlobalClass* ne devront alors plus être attribuées à des sessions particulières, en raison de leur caractère global.

5.8.1 Filtres *Bogon*

Ces filtres, déjà mentionnés au chapitre 4, sont destinés à empêcher des voisins d'annoncer des routes invalides à l'intérieur de l'AS. Ils sont donc appliqués à l'entrée des routeurs BGP et refusent les préfixes qui ont été spécifiés. Dans le langage de définition de politiques, cela se traduit par l'introduction d'une classe spéciale, appelée *PrefixClass*. En pratique, cette classe sera spécifiée de la façon suivante :

```
define rule for PrefixClass[_<suffix>]{
   deny prefix : {<prefix>[,<prefix>]*}
}
```

Ainsi, si l'on désire instaurer des filtres refusant les adresses IP privées, il suffit de définir une *PrefixClass denyPrivateAddresses*, qui sera automatiquement appliquée à toutes les sessions.

```
define rule for PrefixClass_denyPrivateAddresses{
   deny prefix : {192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8}
}
```

Cette classe peut être déclarée soit globalement, soit localement, c'est-à-dire en ne l'attribuant qu'à un sous-ensemble des sessions. Dans le second cas, si l'administrateur souhaite définir différentes classes de ce type, il lui suffit d'attribuer un suffixe au nom de la classe.

5.8.2 MED

Bien qu'utilisé de manière fréquente afin d'influencer le choix du point d'entrée d'un type de trafic dans l'AS, l'attribut BGP MED présente certains inconvénients, dont celui, non négligeable, d'introduire des oscillations de routes en présence de réflecteurs de routes [22]. Étant donné que l'outil que nous cherchons à développer se base sur les réflecteurs de routes pour optimiser la topologie BGP interne du domaine, cela risque d'avoir des conséquences indésirables sur la convergence du protocole. Par défaut, le domaine sera donc configuré pour ne pas utiliser le MED. Néanmoins, il peut être intéressant de laisser à l'administrateur l'opportunité de changer ce comportement. Pour cela, nous définissons une classe MEDClass, semblable à la classe PrefixClass, dont la syntaxe sera la suivante :

```
define rule for MEDClass{
   deny | allow | always-compare
}
```

Cette classe doit obligatoirement être déclarée en temps que GlobalClass. Si l'option choisie est *allow* ou *always-compare*, la configuration C-BGP sera définie avec l'option *med deterministic* ou *med always-compare*. Si le MED n'est pas autorisé dans l'AS, le compilateur ajoutera un filtre à l'entrée supprimant toute information de type MED dans les routes.

Utilisation de communautés annoncées par un voisin BGP

Dans certains cas, des AS voisins peuvent se mettre d'accord sur un ensemble de communautés permettant d'influencer le traitement des routes. Grâce à cette technique, déjà présentée dans le chapitre 4, un client peut par exemple demander à son fournisseur de limiter l'annonce de certaines de ses routes à certains de ses pairs, ou encore de demander à celui-ci de faire de l'*AS-Path prepending*. L'extension proposée pour cette technique est d'ajouter une classe, ou plutôt un ensemble de classes dont le nom est préfixé par 'RemoteCommunity'. Le comportement de ces classes est défini de la manière suivante :

```
define rule for RemoteCommunity[_<suffix>] {
  community = <community_num>
  [To <classname> action = deny | prepend <num_of_prepending>]
}
```

Par exemple, si un client désire que certaines de ses routes, identifiées par la communauté $1234:1234$, ne soient pas annoncées aux fournisseurs de son fournisseur, mais qu'elles soient annoncées aux pairs de ce dernier avec un *prepending* de 3, la règle est la suivante :

```
define rule for RemoteCommunity_1 {
  community = 1234:1234
  To Provider action = deny
  To Peer action = prepend 3
}
```

Ces règles pouvant interférer entre elles, voire même avec les politiques de propagation des routes 'traditionnelles', il nous faut définir des priorités. Ainsi, un refus d'annoncer aura toujours priorité sur une autorisation d'annonce. Quant au *prepending*, il est évidemment conditionné à l'autorisation d'annonce d'une route. Enfin, si une session se voit imposer deux *prependings* différents sur une route, il choisira le plus grand des deux.

5.9 Application : Définition des politiques du réseau GEANT

Afin de démontrer l'efficacité de l'outil de configuration qui a été développé, nous allons l'utiliser pour définir les politiques d'un AS réel de taille moyenne. L'AS qui a été choisi est GEANT, le réseau européen de la recherche. Il est constitué de 23 routeurs connectés à divers réseaux de recherche nationaux ainsi qu'à des réseaux de recherche non-européens et à deux fournisseurs d'accès commerciaux, Telia et Global Crossing.

En plus de la topologie du réseau, reprise à la figure 5.3, nous allons nous créer un petit échantillon de routes qui seront annoncées sur les sessions eBGP. Cet ensemble de routes nous permettra alors de simuler les échanges BGP dans l'AS et vérifier la bonne application des politiques en étudiant les tables de routages de chacun des routeurs qui auront été établies au cours de la simulation.

5.9.1 Définitions des politiques

Les politiques que nous allons appliquer à GEANT sont relativement simples : En plus du traditionnel modèle Client/Pair/Fournisseur, nous allons distinguer parmi tous les voisins eBGP ceux qui sont des réseaux de recherche des autres. Il sera alors possible d'offrir deux types de services aux réseaux de la recherche européens qui constituent l'ensemble des clients : soit ils n'ont accès qu'aux autres réseaux de la recherche, soit ils ont l'opportunité d'accéder à l'Internet commercial via les fournisseurs d'accès. Enfin, certaines des sessions eBGP avec les fournisseurs d'accès Internet seront des sessions de backup, utilisées uniquement en cas de panne des sessions principales.

Le raisonnement suivi pour traduire ces politiques dans notre langage est similaire à celui qui a été utilisé dans l'exemple de la section 5.6. Le résultat obtenu est le suivant :

```
Class {Customer, Peer, Provider, Research, LimitedToResearch, Backup}
Type {CommercialPeering}
belongsTo CommercialPeering : {Customer, Peer, Provider}

defineRuleFor Customer :{
  priority is +9
  announceTo ALL
  acceptFrom ALL
}
defineRuleFor Peer : {
  announceTo {Customer}
```

```
  acceptFrom {Customer}
}
defineRuleFor Provider : {
  priority is -9
  announceTo  {Customer}
  acceptFrom {Customer}
}
defineRuleFor Research: {
  announceTo ALL
  acceptFrom ALL
}
defineRuleFor LimitedToResearch: {
  announceTo  {Research}
  acceptFrom {Research}
}
defineRuleFor Backup: {
  priority is -1
  announceTo ALL
  acceptFrom ALL
}
```

5.9.2 Classification des sessions

Les sessions reliant GEANT à ses fournisseurs d'accès se verront très logiquement attribuer la classe *Provider*. Parmi celles-ci, les sessions avec GlobalCrossing aboutissant aux routeurs IT et CH (Italie et Suisse) seront également définies comme *Backup*, ainsi que la session reliant Telia au routeur UK (Grande-Bretagne).

Etant donné que tous les pairs de GEANT sont des réseaux de recherche non-européens, ils se verront attribuer les classe *Research* et *Peer*. Il s'agit de toutes les sessions liant GEANT avec les réseaux Abilene, ESNET, SINET, CANARIE, TENET, Infonet, ARN, Clara, et enfin EUMED.

Enfin, tous les voisins eBGP restants sont des réseaux de recherche européens qui auront donc les classes *Research* et *Customer*. Parmi ces derniers, certains ne désirent qu'une connectivité avec les réseaux de recherche dans le monde, et non avec l'Internet commercial. Il s'agit en gros des réseaux de recherche d'Europe occidentale, à savoir tous les clients ayant des sessions avec les routeurs CH, FR, PT, UK, IE, NL, BE, LU, ES, DE1, DE2 et SE. Nous leur attribuons donc la classe *RestrictedToEurope*.

Par souci de simplicité, nous ne détaillerons pas ici la traduction de cette classification dans le langage, mais cette dernière peut être trouvée en annexe.

5.9.3 Résultats de la simulation

Grâce aux fichiers de configuration générés par le compilateur à partir du script que nous venons de définir, il nous est possible de simuler l'échange des routes grâce à l'outil C-BGP [26]. Il nous faut encore néanmoins disposer d'un ensemble de routes. Afin de pouvoir analyser les résultats de manière efficace, nous allons sélectionner, parmi les voisins eBGP, un fournisseur d'accès, un pair, un client normal et un client avec accès aux réseaux de recherche, de sorte que chacun d'eux annoncera une route différente. De plus, le fournisseur d'accès annoncera sa routes sur deux sessions eBGP, une session normale et une session de backup. L'attribution des routes est la suivante :

```
Fournisseur = Global Crossing (GB) => 1.0/16
Pair = SINET => 2.0/16
Client normal = IUCC => 3.0/16
Client avec accès à la recherche = BELNET => 4.0/16
```

Après simulation, la table de routage du routeur CH de GEANT, voisin du fournisseur d'accès ; contient les routes suivantes :

```
*>   4.0.0.0/16   62.40.102.3 (BE)   109     4294967295      2611    i
*>   1.0.0.0/16   62.40.102.9 (DE1)  91      4294967295      3549    i
*>   2.0.0.0/16   62.40.102.47 (NY)  100     4294967295      2907    i
*>   3.0.0.0/16   62.40.102.51 (IL)  109     4294967295      378     i
*    1.0.0.0/16   208.48.23.161 (GB) 90      4294967295      3549    i
```

Les détails de la syntaxe de ces routes peuvent être trouvés dans [15], mais nous considérerons ici essentiellement les quatre premières colonnes. Les deux premiers symboles indiquent l'état de la route, $*$ représentant la joignabilité du routeur *Next-Hop*, et $>$ signalant que la route est celle choisie par le routeur pour cette destination. La deuxième colonne représente le préfixe de la route, et la troisième le routeur qui l'a annoncée. Pour plus de clarté, nous avons ajouté entre parenthèses la dénomination de ce routeur. La quatrième colonne, enfin, nous donne le *Local-Pref* qui a été attribué à la route.

On constate qu'en tant que routeur interne à l'AS, il a reçu toutes les routes qui ont été annoncées. De plus, il a deux entrées pour la destination 1.0/16, correspondant aux annonces faites par le fournisseur sur son lien principal le reliant à DE1 et sur son lien de backup le reliant au routeur CH. Le routeur CH a choisi celle correspondant au lien principal en se basant sur l'attribut *Local-Pref*, ce qui était bien le comportement requis.

Observons à présent la table de routage du routeur d'accès d'un client à accès restreint aux réseaux de la recherche :

```
*> 2.0.0.0/16   62.40.102.5 (CH)    0   4294967295   20965 2907    i
*> 3.0.0.0/16   62.40.102.5 (CH)    0   4294967295   20965 378     i
*> 4.0.0.0/16   62.40.102.5 (CH)    0   4294967295   20965 2611    i
```

Le client, relié à l'AS via le routeur CH, a donc bien reçu toutes les routes à l'exception de celle annoncée par le fournisseur d'accès. Des observations similaires sur les tables de routage des pairs et des clients normaux montrent également que les routes ont été transmises conformément à la politique qui a été définie. Les fichiers utilisés pour cette simulation peuvent être trouvés en annexe.

5.10 Conclusion

Le langage développé tout au long de ce chapitre semble remplir les objectifs que nous avions fixés au début de cette partie, à savoir concevoir un outil permettant de configurer de manière efficace et automatique des politiques de routage BGP. Sur base d'un fichier de définition de politiques comportant une vingtaine de lignes, ainsi qu'une liste d'attributions de classes aux sessions eBGP, le compilateur génère automatiquement l'ensemble des fichiers de configuration nécessaires à l'implémentation des politiques sur les routeurs.

Le prototype actuel produit des scripts C-BGP afin de pouvoir effectuer des simulations, mais il va de soi que la traduction vers d'autres types de configuration ne présente aucune difficulté. Les simulations C-BGP ayant permis de valider l'implémentation des politiques via les fichiers de configuration qui ont été générés, il est tout à fait envisageable d'étendre le prototype de compilateur de manière à y intégrer les fonctionnalités supplémentaires qui ont été mentionnées à la section 5.8 afin d'obtenir un outil permettant à des administrateurs de disposer d'une expressivité suffisante pour la gestion BGP de leur réseau. De plus, la souplesse des concepts sur lesquels repose notre langage laisse également la possibilité d'y intégrer de nombreuses autres fonctionnalités que celles qui ont été mentionnées jusqu'ici.

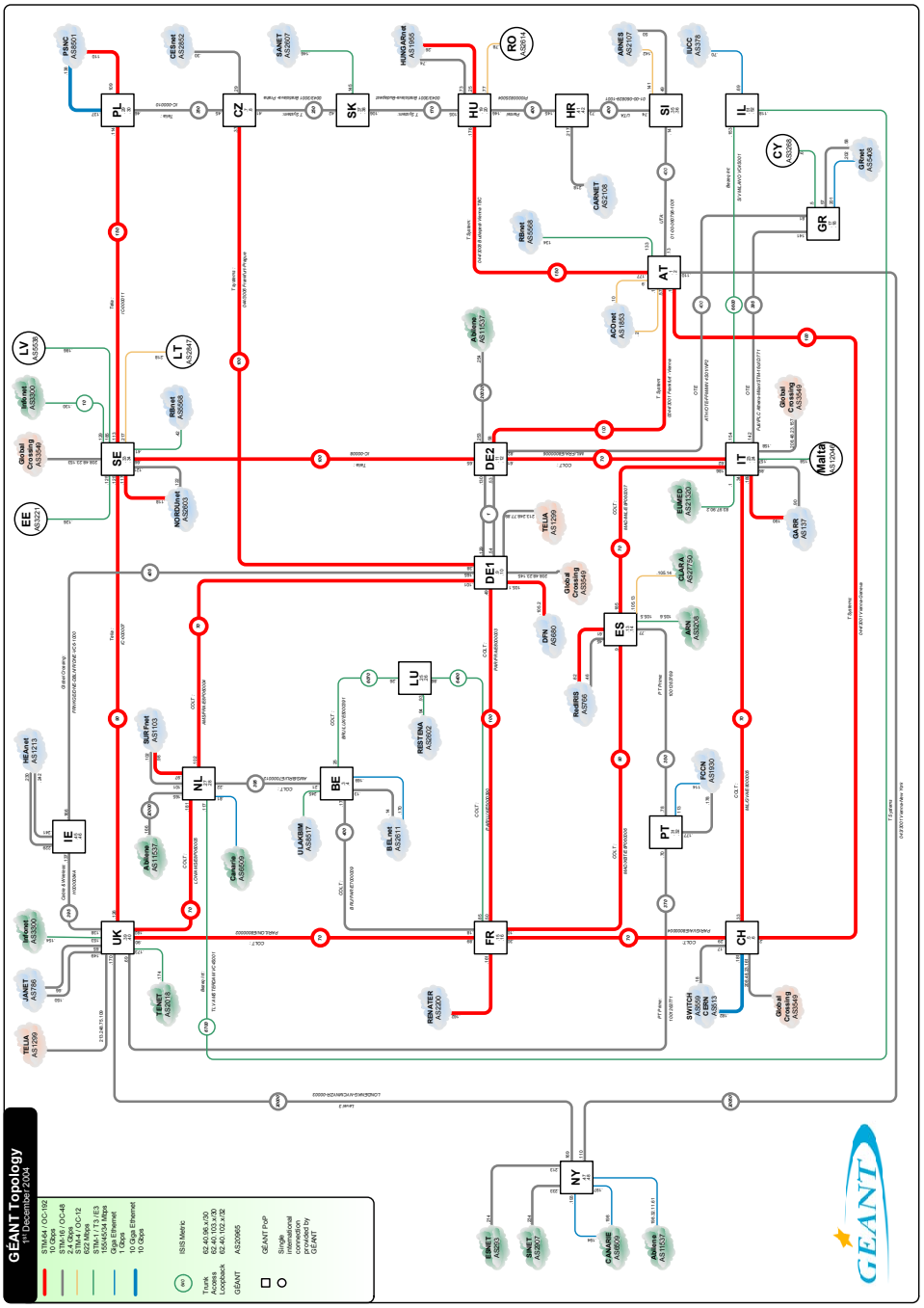

FIG. 5.3 – Topologie GEANT (source : http://www.geant.net/upload/pdf/GEANT_Topology_12-2004.pdf)

Chapitre 6

Optimisation de la topologie iBGP

6.1 Introduction

Nous avons présenté dans les sections précédentes un langage permettant de définir des politiques de routage BGP et de générer à partir de là les fichiers de configuration eBGP de chaque routeur de l'AS considéré. L'étape suivante pour atteindre l'objectif de ce travail est de trouver une topologie iBGP permettant d'optimiser l'échange des routes et le trafic à l'intérieur de l'AS. En effet, la topologie iBGP 'classique' requiert un *Full Mesh* entre tous les routeurs BGP [10], ce qui est très lourd au niveau des performances. Des mécanismes ont heureusement été trouvés pour alléger cette contrainte de *Full Mesh*, à savoir les confédérations [14] et la réflection de routes [21], présentée au chapitre 4. C'est cette dernière solution que nous utiliserons ici pour alléger notre topologie.

La première étape de notre démarche sera tout d'abord de choisir des topologies iBGP candidates, sélection qui sera effectuée en appliquant une procédure détaillée plus loin. Ensuite, pour chacune de ces topologies, nous allons simuler l'échange des routes BGP grâce à l'outil C-BGP [26], et, à partir des tables de routage qui auront été établies dans chacun des routeurs, nous allons quantifier la perte d'optimalité liée à l'utilisation de réflecteurs de routes et la mettre en balance avec le gain qu'ils entraînent en termes de nombre de sessions iBGP et de taille des tables de routage. Après comparaison des résultats des simulations pour chacune des topologies, nous pourront déterminer celle qui conviendra le mieux.

6.2 Choix des sessions iBGP candidates

Le problème qui nous est posé est, étant donné la topologie physique d'un AS, sa politique de routage et les routes BGP qui lui sont annoncées, de trouver la meilleure topologie iBGP en termes de nombre de sessions et d'optimalité des tables de routage. Nous allons ici nous restreindre à une topologie iBGP à un niveau avec un ou deux réflecteurs de route, ce qui implique qu'il nous faut déterminer l'emplacement de ces réflecteurs et les ensembles de clients et de non-clients qui correspondent.

Ce type de topologie convient à des réseaux de taille moyenne ne comportant que quelques dizaines de routeurs, mais des réseaux de plus grande envergure nécessiteront des topologies à plusieurs niveaux de réflecteurs. Nous n'aborderons cependant pas ces dernières dans le cadre de ce travail.

Si l'AS comprend n routeurs BGP, il y a alors n candidats pour l'emplacement d'un réflecteur. Le nombre total de topologies possibles augmente encore si l'on considère qu'il faut également prendre en compte les topologies à deux réflecteurs de routes en déterminant les deux ensembles de clients et de non-clients correspondants. Il devient vite fastidieux de lancer des simulations sur chacune des topologies pour en choisir la meilleure. L'idée sera donc d'effectuer un premier choix parmi les topologies candidates, en appliquant une heuristique qui sera présentée dans la section suivante. Le sous-ensemble de topologies candidates obtenu pourra alors être éventuellement encore optimisé en éliminant les topologies qui risquent d'induire des boucles lors de la signalisation ou lors de la transmission de paquets. Cette sélection s'effectuera sur base de deux critères qui seront étudiés en second point.

6.2.1 Heuristique de sélection des topologies

Présentation de la démarche

Pour trouver un ensemble de topologies iBGP à tester, il nous faut d'abord poser des contraintes afin de limiter le nombre de possibilités. Comme mentionné plus haut, nous n'allons considérer que des topologies à un et deux réflecteurs de routes. De plus, nous prenons l'hypothèse que tous les routeurs de l'AS ont les mêmes capacités en termes de mémoire et de puissance de calcul, ce qui fait que seule la disposition du routeur dans la topologie physique déterminera sa place dans la topologie iBGP. En pratique, ce seront bien évidemment les routeurs les plus performants qui seront candidats à la réflection de routes.

Pour les topologies iBGP ne possédant qu'un seul réflecteur de routes, la première étape lors de leur détermination est de situer l'emplacement du réflecteur. La seconde consistera à attribuer le statut de client ou de non-client aux routeurs restants. Nous ne travaillerons ici qu'avec des routeurs ayant le statut de client, et possédant donc une unique session iBGP avec le réflecteur.

Pour tester des topologies avec deux réflecteurs, il faut également localiser ces derniers, puis déterminer quels routeurs seront les clients de l'un ou de l'autre. Nous envisagerons chaque fois deux cas : Soit les clients ne sont connectés qu'à un seul réflecteur de routes (figure 6.1(a)), soit ils sont connectés aux deux réflecteurs(figure 6.1(b)). La seconde possibilité, couramment utilisée, nécessite plus de sessions iBGP et augmente la taille des tables de routages des clients mais peut éventuellement augmenter la redondance des routes, et donc la robustesse du système.

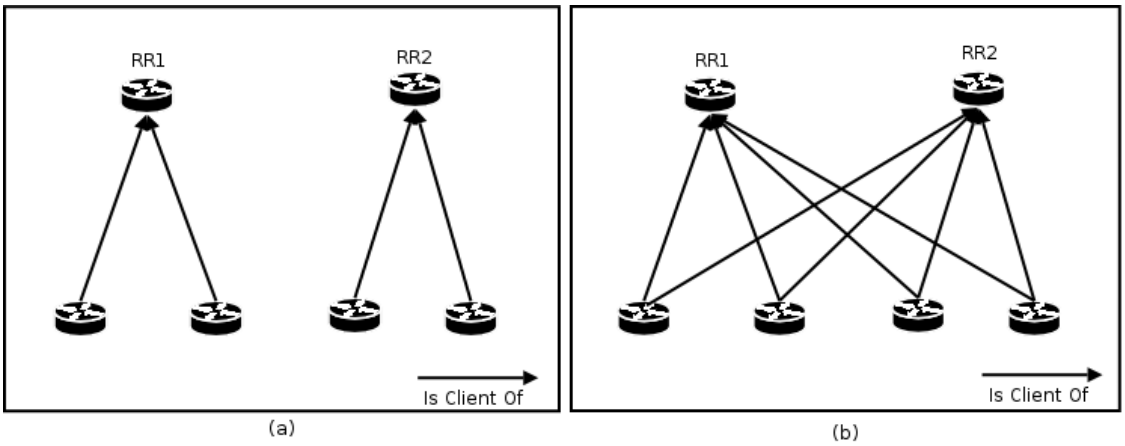

FIG. 6.1 – Topologies à deux réflecteurs de routes

Présentation de l'heuristique de localisation des réflecteurs

Pour choisir le routeur qui fera office de réflecteur de route, nous allons considérer sa position dans le réseau. En effet, plus un routeur est situé au centre du réseau, plus les sessions iBGP qu'il établira en tant que réflecteur de routes avec ses clients seront courtes. De plus, placer un réflecteur de routes au centre permet de refléter autant que possible la topologie physique du réseau. Or, les topologies iBGP proches des topologies IGP ont moins tendance à induire des boucles de routage, ce qui permet un routage plus stable et plus efficace [28].

Un routeur sera considéré comme étant au centre du réseau si la somme des distances pour joindre tous les autres routeurs du réseau est minimale. Cette somme $\sum dist$ sera donc utilisée pour déterminer les routeurs qui seront les plus efficaces en tant que réflecteurs de routes. Elle est calculée en appliquant l'algorithme de Dijkstra à la topologie physique du réseau, afin d'obtenir l'arbre des chemins les plus courts vers le routeur considéré. Il suffit alors d'additionner les longueurs de tous ces chemins pour obtenir la valeur demandée.

Une fois tous les routeurs classés selon cette méthode, il devient trivial de construire des topologies candidates avec un réflecteur de route. Il reste à considérer les topologies avec double réflection, et à choisir les paires de réflecteurs correspondantes. D'après [28], il est de bonne pratique de placer des réflecteurs de routes de telle manière qu'il n'y ait pas de session iBGP entre deux réflecteurs transitant par un client. Dans le cas contraire, la topologie physique n'est pas respectée, et des boucles de paquets peuvent facilement apparaître dans le réseau.

Cela a donc pour conséquence que, dans notre topologie à deux réflecteurs, il ne peut y avoir de routeur entre ceux-ci, à moins qu'il ne soit non-client. Étant donné que nous ne considérons pas le cas des non-clients ici, nous allons donc établir un classement des paires de réflecteurs en éliminant celles qui ne répondent pas à cette contrainte.

Nous complétons ensuite notre topologie en attribuant le statut de client à tous les routeurs qui ne sont pas réflecteurs et, dans le cas où il y a double réflecion avec des ensembles de clients disjoints, nous attribuons chaque client au réflecteur le plus proche de lui au sens IGP. Nous ajoutons ensuite une session iBGP entre chaque réflecteur de routes et chacun de ses clients.

L'étape suivante consiste alors à vérifier si les topologies que nous avons sélectionnées remplissent les conditions de correction présentées dans la section suivante, afin d'éviter des boucles soit lors de la diffusion des routes, soit lors de la transmission des paquets.

6.2.2 Conditions de correction

Nous allons étudier dans cette section deux types d'anomalies qui peuvent apparaître en présence de topologies iBGP avec réflection de routes. La première concerne la convergence du protocole, et la seconde, la transmission des paquets à travers l'AS. Ces anomalies peuvent être dues à deux choses : l'utilisation de l'attribut *MED* dans les routes [22], et la topologie iBGP en elle-même. La première cause d'anomalie peut

être éliminée en n'utilisant pas l'attribut *MED*, ou bien en ne l'utilisant qu'avec l'option *Always-Compare*. La seconde cause sera quant à elle analysée tout au long de cette section.

Correction signalétique

Avoir une topologie iBGP correcte au niveau de la signalisation des routes signifie que, quelles que soient les routes annoncées dans l'AS, les échanges de messages iBGP convergeront de manière déterministe pour donner des tables de routage stables. En effet, dans certains cas, il est possible soit qu'il n'y ait pas de convergence dans l'établissement des routes, soit que l'établissement de ces routes varie en fonction de l'ordre dans lequel les routes sont annoncées aux points d'entrée du réseau, comme nous allons le montrer dans les deux exemples suivants tirés de [17] :

Non-convergence du routage Le graphe de la figure 6.2 modélise un AS possédant trois réflecteurs de routes avec chacun un client. Les routes utilisées dans cet exemple, ainsi que dans les suivants, sont considérées comme équivalentes au niveau des attributs *Local-Pref* et *AS-Path*, ne différant donc que par l'endroit où elles ont été annoncées. Les liens physiques sont représentés par les arcs en trait plein, tandis que les sessions iBGP sont les flèches en pointillés.

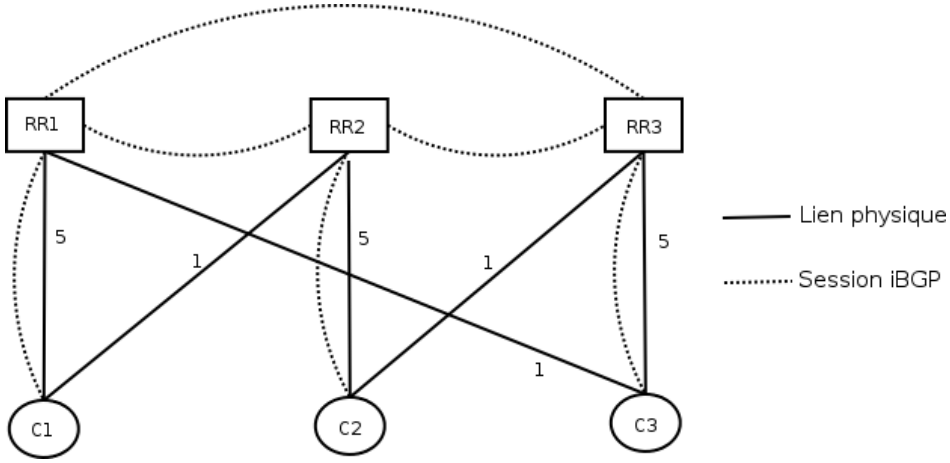

FIG. 6.2 – Exemple de topologie empêchant la convergence de BGP

Imaginons que chaque client reçoive une route vers une destination donnée. Chacun d'eux va l'annoncer à son réflecteur de route, qui, à son tour, l'annonce aux autres

réflecteurs. A cause des coûts IGP, chaque réflecteur RR_i va classer par ordre de préférence les routes passant par C_{i+1}, puis par C_i, puis par C_{i-1} (les préfixes sont à prendre modulo 3). Ainsi, le RR1 va annoncer à RR3 la route passant par C1, avant de recevoir la route passant par C2, qu'il va préférer. Mais RR2 a entre-temps reçu la route passant par C3, et va la préférer à celle précédemment annoncée. Il va donc la redistribuer à R1, qui va recommencer son choix, et ainsi de suite. L'établissement des routes ne converge donc pas.

Routage non déterministe La figure 6.3 comporte deux réflecteurs, avec chacun un client.

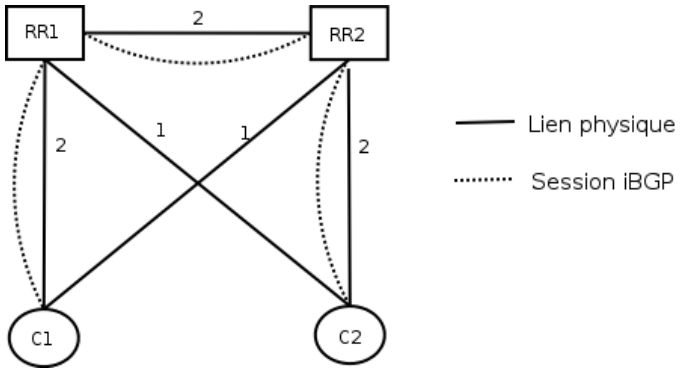

FIG. 6.3 – Exemple de topologie menant à un routage non déterministe

Imaginons qu'une route est reçue à la fois par C1 et par C2. Chacun d'eux va la transmettre à son réflecteur, qui la sélectionnera provisoirement comme meilleure route. Imaginons ensuite que RR1 est plus rapide que RR2, et lui transmet sa route via C1. RR2 va donc préférer cette nouvelle route, alors que si, à l'inverse, RR2 avait été le plus rapide, RR1 aurait choisi choisi la même route que RR2, à savoir celle passant par C2. On a donc ici deux possibilités d'établissement des routes, en fonction du routeur qui annonce sa route à son voisin en premier. Remarquons également que dans chacun des cas, le réflecteur 'annonciateur' de la route ne choisira finalement pas la route optimale, puisqu'il n'en a pas connaissance.

Il est évident que, dans ces exemples, les problèmes illustrés viennent du fait que chaque réflecteur choisira préférentiellement les routes passant par le client du réflecteur voisin, à cause des coûts IGP. Il a été prouvé que déterminer si une topologie induit une signalisation des routes convergeante et déterministe est un problème NP-complet [17]. Il existe néanmoins une condition suffisante permettant de garantir qu'une topologie n'induira pas ce genre de problèmes [17], composée des deux points suivants :

– Le graphe dont les noeuds sont les routeurs BGP et dont les arcs orientés relient un client à son réflecteur doit former un graphe orienté acyclique.

– Un réflecteur de route préférera une route annoncée par un client plutôt qu'une route annoncée par un autre réflecteur.

Cette condition est déduite par analogie avec une condition de convergence eBGP pour le modèle *Client*/*Pair*/*Fournisseur* [29]. La vérification du premier point est triviale. Par contre, en ce qui concerne le second point, étant donné qu'on ne peut influencer le processus de décision BGP pour préférer les routes d'un routeur à un autre, il faut jouer sur les paramètres dont ce processus tient compte pour faire son choix. Une solution serait de s'assurer que chacun des clients d'un réflecteur est plus proche de lui au sens IGP que n'importe quel autre routeur pour qu'il soit vérifié.

[30] propose également une condition suffisante pour la correction signalétique iBGP. Étant donné un client k, il faut que ce client ne soit pas plus éloigné de son réflecteur de route (reflect(k)) que de n'importe quel autre réflecteur j de l'AS. Cela nous donne donc que :

$$\forall(j,k), dist(reflect(k),k) \leq dist(j,k) \tag{6.1}$$

La preuve de cette condition découle de l'application des propriétés algébriques de monotonicité et d'isotonicité qui, d'après [30], garantissent la convergence optimale d'un protocole à vecteurs de chemin.

Étant donné la difficulté d'application de la condition de [17], nous allons lui préférer celle de [30]. Remarquons cependant que cette dernière ne tient pas compte du cas où un client est relié à deux réflecteurs de routes. Cela ne devrait cependant pas avoir d'impact sur l'échange des routes, car une topologie de ce type ne diffère qu'en un point d'une topologie où un client n'est relié qu'à un réflecteur de routes : Un client peut recevoir deux routes différentes vers une même destination. Dans ce cas, il effectue son choix sur base de la distance IGP jusqu'au next-hop, puis annonce la route choisie au réflecteur qui ne l'a pas annoncée. Néanmoins, ce dernier connaît déjà cette route puisqu'il possède également une session iBGP avec l'autre réflecteur, et son choix n'est donc pas influencé. Cette situation ne devrait donc pas induire de boucles si les conditions sont respectées.

Correction de la transmission de paquets

En plus d'éventuels problèmes lors de l'établissement des routes, il est également possible que des boucles apparaissent lors de la transmission des paquets. Cela vient

du fait que, sur le chemin parcouru par un paquet à travers l'AS, il est possible que ce dernier rencontre des routeurs dont les points de sortie pour cette destination sont différents. Nous appellerons ces changements de points de sortie *déflections*. Lorsque plusieurs déflections se succèdent, il est possible que le paquet se mette à boucler.

La topologie suivante (figure 6.4) est un exemple classique dans lequel des boucles peuvent s'établir : Elle consiste en 2 réflecteurs, avec chacun un client. Chaque client a un lien physique avec le réflecteur auquel il n'appartient pas, et les deux clients sont reliés entre eux. Si une route est annoncée aussi bien à RR1 qu'à RR2, on va avoir une boucle lors de la transmission des paquets. En effet, C1 va envoyer le paquet en direction de RR1, mais ce dernier va transiter par C2, qui va le rediriger vers C1 afin qu'il atteigne RR2, et ainsi de suite.

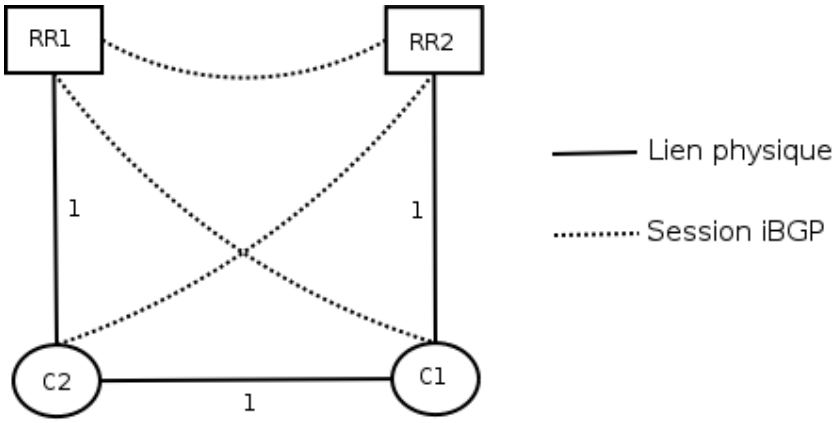

FIG. 6.4 – Exemple de topologie pouvant induire des boucles lors de la transmission des paquets

Il a été démontre que le problème consistant à déterminer si une topologie iBGP risque de mener à des boucles de transmission est NP-Complet [17] . Il existe néanmoins des conditions suffisantes permettant de garantir l'absence de ces boucles, et qui se basent sur la notion de *chemin de signalisation valide* [17]. Un chemin de signalisation représente le trajet parcouru par une route lorsqu'elle est annoncée à l'intérieur d'un AS. Un tel chemin est dit valide s'il transite d'abord 'vers le haut', c'est-à-dire de clients en réflecteurs, puis d'un réflecteur à un autre ou à un non-client, et enfin 'vers le bas', c'est-à-dire de réflecteurs à clients. Cette définition implique que le chemin d'une route dans le graphe représentant le réseau puisse se scinder en trois ensemble d'arcs distincts, dont il est utile de préciser que chacun d'eux peut être vide.

Les conditions sont alors les suivantes :

- Pour tout réflecteur u, avec v un client et w un non-client, il faut que tout chemin de signalisation valide passant par v soit plus court au sens IGP que tout chemin de signalisation valide vers la même destination passant par w. Cela revient donc à imposer qu'un réflecteur de routes préfère les routes annoncées par ses clients plutôt que celles annoncées par n'importe quel autre routeur.
- Pour toute paire de noeuds, le chemin le plus court entre ces deux noeuds constitue un chemin de signalisation valide.

Application des conditions de correction

Il est évident que, si ces conditions nous garantissent la correction de notre topologie iBGP, elles se révèlent en contrepartie assez restrictives dans leur application. Ainsi, pour garantir une transmission correcte des paquets, il nous faut imposer qu'un réflecteur de routes choisisse préférentiellement les routes annoncées par ses clients. Or, comme ce choix se fait sur base du coût IGP vers le routeur qui a reçu la route via une session eBGP, nous nous retrouvons dans la même situation que dans le cas de la condition de signalisation de [17].

De plus, la seconde partie de la condition pour une transmission correcte impose que chaque plus court chemin entre deux routeurs soit un chemin de signalisation valide. Cela implique donc qu'il faut que tous les routeurs le long d'un tel chemin soient reliés par des sessions iBGP successives. Cette condition suffisante est tellement restrictive qu'elle ne garantit pas la correction d'une topologie de type *Full Mesh* n'ayant pas un *Full Mesh* au niveau de son graphe physique. Elle risque donc de limiter de manière drastique l'ensemble des topologies iBGP candidates. Il serait donc nécessaire de trouver une autre condition à caractère moins restrictif [23].

Nous allons donc tenter d'appliquer ces conditions à chacune des topologies sélectionnées, et évaluer à partir des résultats s'il faut supprimer certaines topologies candidates ou non.

6.2.3 Algorithme de sélection des topologies candidates

La sélection des topologies candidates s'effectuera suivant les étapes suivantes :

1. Classement des routeurs sur base de leur position dans le réseau
2. Construction des topologies avec un réflecteur de route

3. Choix de paires de réflecteurs de route sur base de la distance qui la sépare, sous contrainte qu'il n'y ait pas de clients entre eux.
4. Construction des topologies avec deux réflecteurs, un par client
5. Construction des topologies avec deux réflecteurs, deux par client
6. Vérification des conditions de correction, et élimination éventuelle des topologies incorrectes

Nous obtenons alors un ensemble de topologies avec réflecteurs de routes, auquel nous ajouterons la topologie *Full Mesh*, qui servira de topologie de référence pour l'évaluation des différents candidats.

6.3 Mesure du gain en termes de sessions iBGP

A présent que le choix d'un ensemble de topologies est effectué, il nous faut arriver à déterminer, pour chacune d'elles, le gain qu'elle apporte par rapport au full mesh. Un premier avantage résultant de l'utilisation des réflecteurs de route est une réduction du nombre de routes reçues par chacun des routeurs, et donc un allègement des *Adj-Rib-Ins*. Enfin, l'utilisation des réflecteurs de routes engendre une réduction du nombre de sessions iBGP établies à l'intérieur de l'AS.

Cette réduction peut se quantifier simplement en comptant le nombre de sessions iBGP, néanmoins, [31] propose une métrique légèrement plus complexe pour mesurer l'efficacité d'une topologie iBGP, à savoir la somme du nombre de 'sauts' impliqués par chaque session iBGP. Néanmoins, puisque nous disposons des poids IGP des arcs du graphe physique, nous n'utiliserons pas la somme des 'sauts', mais plutôt la somme des distances entre deux voisins iBGP.

Afin de disposer de métriques d'ordres de grandeur comparables, nous prendrons, pour chaque critère d'évaluation, la valeur relative de chacun d'eux par rapport à ses valeurs extrêmes dans l'échantillon de topologies considérées. Cela nous permet donc d'établir un classement des topologies d'un même échantillon, mais ne représente pas une quantité absolue qui peut être comparée à des valeurs pour d'autres échantillons.

6.4 Mesure de la perte d'optimalité des tables de routage

6.4.1 Mesure de la perte d'optimalité au niveau du choix des routes

Une fois que les tables de routage se sont stabilisées dans le réseau, il devient possible de mesurer cette fameuse perte d'optimalité. Pour cela, nous allons considérer, pour chaque routeur et pour chaque préfixe de destination, la distance IGP entre le routeur et le point de sortie indiqué dans la table de routage pour le préfixe en question. Il nous suffit alors d'additionner l'ensemble de ces distances et de comparer le résultat avec ce que nous aurions obtenu s'il y avait eu un *Full Mesh* de sessions iBGP.

6.4.2 Mesure de la perte d'optimalité au niveau de la redondance des routes

Ce qui nous intéresse, ici, c'est de connaître, pour chaque routeur d'une topologie, le nombre de préfixes vers lesquels il possède plus d'une route dans ses *Adj-Rib-Ins*. Plus il y a de redondances dans les routes, plus les routeurs réagiront rapidement en cas de panne d'un lien. Typiquement, les topologies avec un ou deux réflecteurs possédant chacun un ensemble distinct de clients auront une redondance faible, puisque tous les routeurs exceptés les réflecteurs n'ont qu'un seul voisin iBGP. Les topologies avec des clients reliés à deux réflecteurs de routes auront probablement plus de choix, au détriment de la taille de leurs Ribs-In.

Dans le cas d'une topologie *Full Mesh*, lorsqu'il existe plusieurs routes vers une même destination dans les *Adj-Rib-Ins*, chacune d'elles diffère des autres par son routeur *Next-Hop*, puisqu'elles ont toutes été annoncées dans l'AS par un routeur différent. Dans cette situation, une route est invalidée lorsque ce routeur *Next-Hop* devient injoignable. Par contre, lorsque la topologie possède un réflecteur de routes, une route peut être également invalidée si le réflecteur tombe en panne, puisque toutes les routes de l'*Adj-Rib-In* correspondante sont alors supprimées. Dans ce cas, il y aura même perte de connectivité, puisque, comme chaque client voit sa seule session iBGP supprimée, il n'est plus capable de recevoir d'autres routes venant de l'intérieur de l'AS. Le réseau est dès lors très sensible aux pannes du réflecteur de routes [11].

Par contre, si l'on dispose de deux réflecteurs redondants, il peut exister dans les tables de routage deux versions d'une route vers une destination donnée possédant le même routeur de sortie de l'AS, chacune d'elles ayant été annoncée par l'un des réflecteurs de routes. Posséder deux routes avec le même routeur de sortie n'est d'aucune utilité en cas de panne le long du chemin vers ce routeur, et la seule possibilité qu'il reste pour pouvoir joindre cette destination consiste à recevoir une nouvelle route des réflecteurs.

Néanmoins, l'existence de ces doubles routes protège le réseau en cas de panne d'un réflecteur, puisque, dans ce cas, il restera toujours dans les *Adj-Rib-Ins* au moins une route vers chaque destination.

La métrique que nous allons utiliser pour ce critère ne prendra en compte que les routes vers une destination donnée possédant des *Next-Hops* différents, ce qui nous permettra de mesurer la robustesse du système aux pannes des liens vers les points de sortie de l'AS. La robustesse du système aux pannes de sessions iBGP ou aux pannes de réflecteurs étant en effet uniquement dépendante du type de topologie utilisé, il n'est pas utile de chercher à mesurer les différences individuelles entre les topologies en termes de nombre de routes doubles avec le même routeur de sortie.

Notre calcul consistera donc à dénombrer, pour chaque routeur, les routes de la *Local Rib* possédant au minimum un 'backup' dans les *Adj-Rib-Ins*, c'est-à-dire une seconde route vers la même destination mais ayant un routeur de sortie de l'AS différent. Cette valeur représentera donc le nombre de routes protégées en cas de panne du ou des liens vers leur point de sortie de l'AS. Il fournit donc une indication de la réactivité du système en cas de panne, c'est-à-dire sa capacité à réagir à cette panne sans devoir échanger de nouveaux messages BGP.

6.5 Normalisation des critères d'évaluation

Afin de pouvoir définir une métrique unique pour le choix d'une topologie, il nous faut disposer de valeurs comparables pour chacun des critères. Nous allons pour cela les modifier de telle sorte que la valeur minimum de chaque critère soit nulle, et que la valeur maximum soit égale à l'unité. Nous appellerons ces métriques *valeurs normalisées*, et les calculerons à l'aide des expressions suivantes :

$$\frac{TailleRibIn_i - TailleRibIn_{min}}{TailleRibIn_{max} - TailleRibIn_{min}} \tag{6.2}$$

$$\frac{\sum dist_i - \sum dist_{min}}{\sum dist_{max} - \sum dist_{min}} \tag{6.3}$$

$$\frac{\sum DistToNextHop_i - \sum DistToNextHop_{min}}{\sum DistToNextHop_{max} - \sum DistToNextHop_{min}} \tag{6.4}$$

$$\frac{\#RoutesRedondantes_i - \#RoutesRedondantes_{min}}{\#RoutesRedondantes_{max} - \#RoutesRedondantes_{min}} \tag{6.5}$$

En les combinant linéairement avec des poids correspondant à l'importance que l'administrateur donne respectivement à chaque critère, il est possible d'obtenir une métrique permettant de sélectionner la topologie qui se prête le mieux aux souhaits de ce dernier.

6.6 Étude d'une topologie iBGP pour le réseau GEANT

Nous allons à présent appliquer la procédure définie plus haut pour constituer un ensemble de topologies candidates, puis leur appliquer à chacune les quatre métriques qui ont été développées, afin de pouvoir étudier leurs évolutions et interactions respectives. Un outil a été implémenté à cet effet, constitué d'une représentation sous forme de classe Java d'une topologie iBGP, et pouvant interagir avec le simulateur C-BGP afin d'obtenir les tables de routage correspondant aux topologies étudiées et à un ensemble de routes annoncées par les voisins eBGP de l'AS.

Comme pour la validation du compilateur de configurations développé au chapitre 5, nous allons utiliser comme cas d'étude le réseau GEANT, qui, avec ses vingt-trois routeurs, constitue un AS de transit de taille moyenne convenant bien à la procédure d'optimisation iBGP présentée plus haut.

Nous reprendrons donc les fichiers de configuration C-BGP produits au chapitre 5 pour la validation du compilateur, à l'exception bien sûr de la configuration iBGP. Les filtres ont quant à eux été modifiés, afin de garder des valeurs de *Local-Pref* équivalentes à celles utilisées dans le réseau GEANT actuel. Pour les routes annoncées dans l'AS, nous avons reconstitué un échantillon à partir du contenu des tables de routage d'un des routeurs BGP de l'AS. Etant donné que ces tables de routage ont été établies sur base d'une topologie de type *Full Mesh*, il est alors possible de reconstituer l'ensemble de routes annoncées sur chacune des sessions eBGP établies par GEANT avec ses voisins.

Enfin, afin de limiter le temps nécessaire à la réalisation des simulations, nous avons réduit le nombre de routes afin de ne garder que maximum les deux cents premières routes annoncées sur chaque session eBGP. De plus, s'il existe plusieurs sessions eBGP avec le même AS avec des ensembles de routes non nuls, nous sélectionnons les deux cents premières routes présentes simultanément sur les deux sessions. Le cas des fournisseurs d'accès doit également être considéré, puisque ceux-ci annoncent des routes sensiblement identiques, à savoir celles de l'Internet commercial. Nous avons donc sélectionné les deux cents premières routes communes à la session de Global Crossing sur CH et à celle de Telia sur DE2, qui seront alors les routes annoncées sur les six sessions d'accès à Internet. Ces sélections ont chaque fois été effectuées à l'aide d'une classe Java qui peut être trouvée en annexe. Etant donné qu'elles ont été effectuée sans tenir compte du préfixe concerné, l'étude que nous réaliserons ne tiendra donc pas compte du trafic induit par chaque route. Si cela nous permet de faire une première analyse des topologies, il serait intéressant d'affiner ultérieurement nos conclusions en

effectuant une sélection plus précise des routes. Nous ne l'effectuerons néanmoins pas dans le cadre de ce mémoire.

6.6.1 Classification des routeurs

Suivant la procédure définie au début de ce chapitre, les routeurs ont été classés selon la somme des distances entre eux et chacun des autres routeurs. Les résultats sont repris dans le graphique de la figure 6.5.

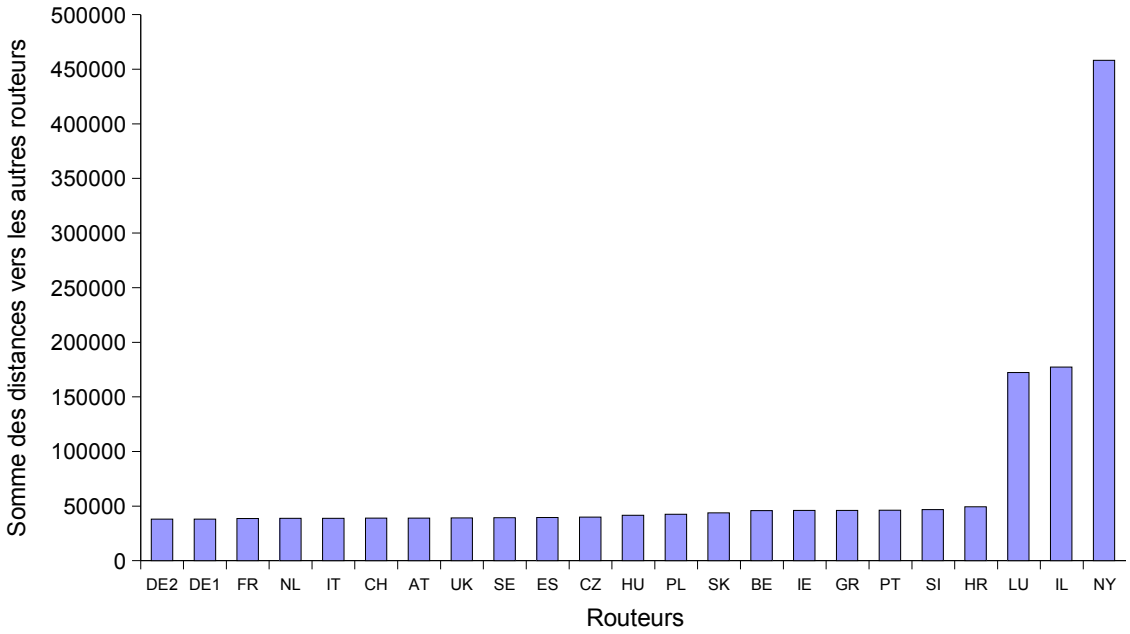

FIG. 6.5 – Classement des routeurs

Il apparaît clairement que trois des routeurs se démarquent nettement des autres par une position très décentralisée. Il s'agit des routeurs NY, IL et LU. Ces résultats sont cohérents avec la topologie présentée à la figure 5.3, qui montre que ces trois routeurs sont reliés au réseau par des liens pondérés par des coûts IGP allant de 6400 à 20000. Dans le cas du routeur NY, situé à New York, cela s'explique par le coût de la ligne inter-continentale sur laquelle on désire éviter de faire transiter du trafic. IL, situé en Israël, fait également les frais d'une connexion à grande distance. Quant au routeur LU du Luxembourg, la faible étendue géographique de cet état ne demande pas d'énormes ressources, et le coût IGP des liens correspondant s'explique donc par une infrastructure

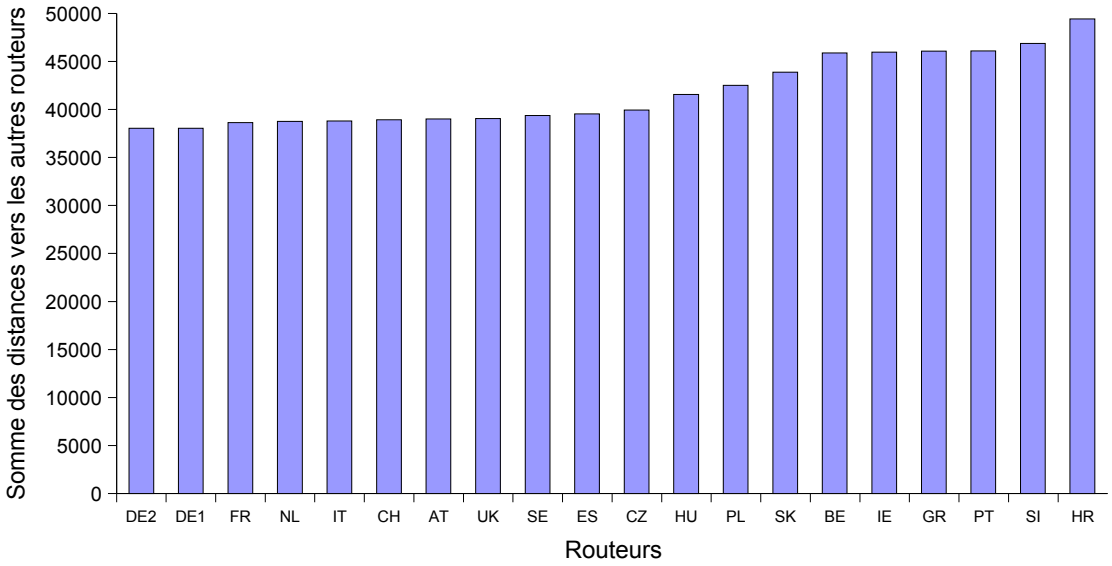

FIG. 6.6 – Classement des routeurs (2)

plus légère. Ces routeurs ne semblent donc absolument pas adaptés au rôle de réflecteur de routes.

Les autres routeurs apparaissent comme relativement équivalents, par rapport aux valeurs extrêmes prises par les trois routeurs mentionnés plus haut. Il est donc intéressant de reconsidérer les valeurs prises par les vingt premiers routeurs indépendamment des trois derniers, afin de pouvoir observer plus précisément les différences qui existent entre eux. C'est ce que montre le graphique de la figure 6.6.

Les différences entre ces routeurs sont bien entendu moins marquées que celles existant avec les routeurs NY, IL et LU, mais on distingue néanmoins deux niveaux de valeurs. Ainsi, la somme des distances vers les autres routeurs des onze premiers routeurs du graphique ne dépasse pas les 40000 en termes de coûts IGP, avec une variance assez faible entre individus. Les routeurs suivants prennent des valeurs de plus en plus élevées, allant jusque 50000 dans le cas du routeur HR. La différence entre le routeur le plus central, DE2, et le routeur HR est donc d'une vingtaine de pourcents. Bien que cela paraisse presque négligeable en comparaison avec les valeurs atteintes par le routeur NY, une telle différence justifie clairement que DE2 soit choisi préférentiellement à HR pour le rôle de réflecteur de routes.

TAB. 6.1 – Identifiants des topologies

Identifiant	Description de la topologie
FM	*Full Mesh*
DE2	1 réflecteur : routeur DE2
DE1	1 réflecteur : routeur DE1
FR	1 réflecteur : routeur FR
NL	1 réflecteur : routeur NL
IT	1 réflecteur : routeur IT
DE1 & FR	2 réflecteurs sans redondance : DE1 & FR
DE1 & NL	2 réflecteurs sans redondance : DE1 & NL
DE1 & DE2	2 réflecteurs sans redondance : DE1 & DE2
DE1 + FR	2 réflecteurs avec redondance : DE1 & FR
DE1 + NL	2 réflecteurs avec redondance : DE1 & NL
DE1 + DE2	2 réflecteurs avec redondance : DE1 & DE2

Au vu de cet analyse, il paraît dès lors raisonnable de considérer des topologies dont les réflecteurs de routes figurent parmi les dix premiers routeurs du classement. Nous ne tiendrons en fait compte que des cinq premiers routeurs, afin de limiter le nombre de topologies à tester. Nous constituerons donc cinq topologies avec un réflecteur de routes, ainsi que les vingt topologies à deux réflecteurs découlant des combinaisons entre les cinq routeurs. Parmi ces vingt topologies, seules six d'entre elles répondent à la condition stipulant qu'il ne doit pas y avoir de client entre deux réflecteurs de routes, et nous éliminerons donc les quatorze autres.

6.6.2 Évaluation des topologies

Nous allons à présent comparer les topologies qui ont été sélectionnées, et qui sont de quatre types : *Full Mesh*, avec un réflecteur de routes, et enfin, avec deux réflecteurs, avec ou sans redondance. Afin de rendre plus lisibles les graphiques qui seront présentés plus loin, nous allons identifier chacune des topologies par un code. Le tableau 6.1 nous donne la correspondance entre chaque topologie et son identifiant.

Vérification de la correction des topologies

Avant de nous lancer dans des simulations, nous allons tout d'abord évaluer les topologies choisies par rapport aux conditions de correction présentées plus haut. Ces conditions ne présentent pas de difficulté d'implémentation, puisqu'elles ne se basent

que sur les topologies, aussi bien physiques qu'iBGP.

Après vérification, il apparaît que l'ensemble des topologies que nous avons définies répondent au critère de correction de la signalisation de [30]. Cela découle de la méthode employée pour construire ces topologies, qui restent finalement assez simples : Dans le cas des topologies à un réflecteur, l'application de la condition est triviale puisque tous les clients sont forcément plus proches de leur réflecteur que des autres, qui sont inexistants. Pour les topologies à deux réflecteurs sans redondance, la condition est vérifiée par construction, puisque chaque client est relié au réflecteur le plus proche de lui. Enfin, dans le cas de l'ajout de redondance, la condition est également vérifiée. Nos topologies garantissent donc la convergence du protocole lors de l'échange des routes.

La condition pour garantir l'absence de boucles dans la transmission des paquets n'est par contre vérifiée par aucune des topologies. Ainsi que nous l'avions mentionné plus haut, cela vient du fait qu'elle est extrêmement restrictive, et n'est vérifiée que si la topologie iBGP est extrêmement proche de la topologie physique, ce qui est difficile, voire impossible à garantir dans un réseau un peu complexe tel que GEANT. Nous ne sommes donc pas en mesure de garantir l'absence de bouclage de paquets dans nos topologies. Nous avons néanmoins réduit ce risque pour les topologies à deux réflecteurs en ne considérant que des paires de réflecteurs sans clients sur le chemin les reliant.

Somme des longueurs des sessions iBGP

Le but premier de l'utilisation des réflecteurs de routes est de limiter le nombre de sessions iBGP nécessaires à l'échange des routes à l'intérieur d'un AS. Le graphique 6.7 nous donne donc, pour chaque topologie, la somme des longueurs des sessions iBGP correspondantes.

Au vu de ces résultats, il est indéniable que l'introduction de réflecteurs de routes réduit efficacement la somme des longueurs des sessions iBGP. En effet, le nombre de sessions iBGP pour une topologie *Full Mesh* est proportionnel au carré du nombre de routeurs, alors qu'avec des réflecteurs, le rapport n'est que linéaire, ce qui explique la grande différence qui existe pour ce critère dans un réseau comportant vingt-trois routeurs.

Assez logiquement, les variations entre topologies du même type sont relativement faibles. Dans le cas des topologies à un réflecteur, cette métrique est équivalente à la

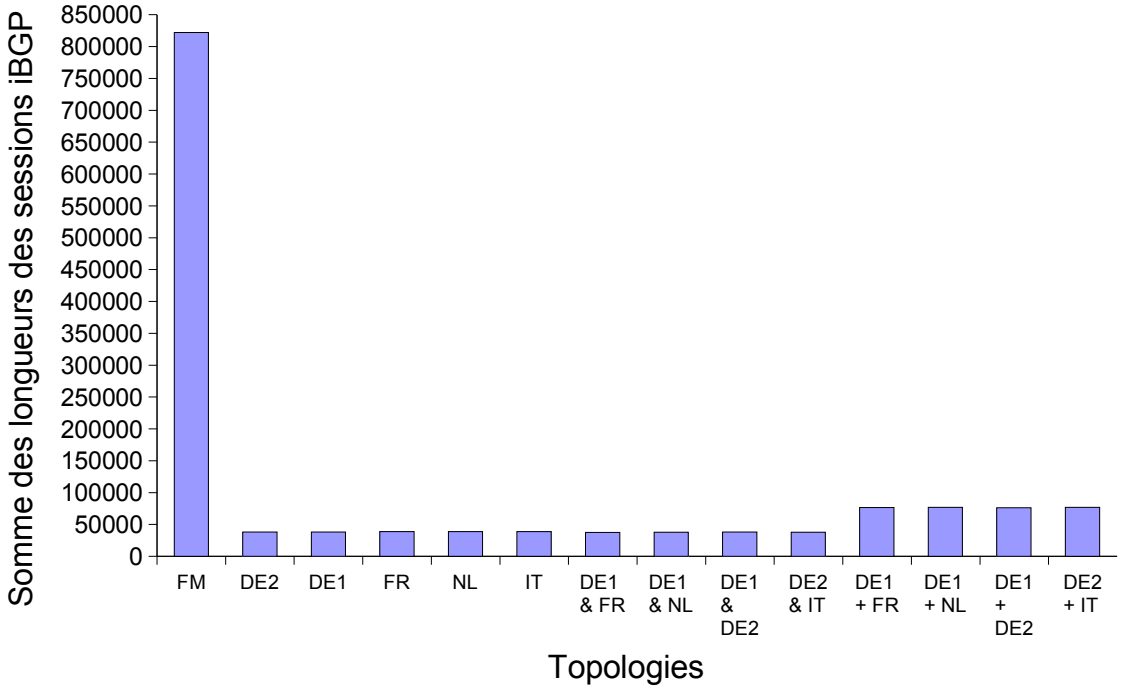

FIG. 6.7 – Somme des longueurs des sessions iBGP

valeur utilisée pour classer les routeurs. Pour les topologies à deux réflecteurs sans redondance, la somme des distances vers les clients est d'autant plus petite que les deux réflecteurs sont éloignés, tout en restant très proches des valeurs pour les topologies avec un réflecteur. S'il y a redondance, la somme des longueurs des sessions est logiquement environ deux fois plus grande que lorsqu'il n'y en a pas.

Taille des tables de routage Adj-Rib-Ins

Les différences en termes de taille des tables de routage viennent du fait que les réflecteurs effectuent une sélection sur les routes qu'ils reçoivent avant de les annoncer à leurs clients. Cela se traduit, dans notre étude de cas, par une réduction d'environ 20% de la taille des Adj-Rib-Ins entre la topologie *Full Mesh* et les topologies avec un réflecteur de routes (figure 6.8). La situation est identique en présence de deux réflecteurs indépendants, mais, si l'on ajoute de la redondance, les tables de routage doublent de taille, puisque chaque client reçoit alors deux routes pour chaque destination. Malgré la surcharge en terme d'utilisation de la mémoire que cela implique, ce phénomène reflète le fait que les topologies avec redondance sont protégées en cas de

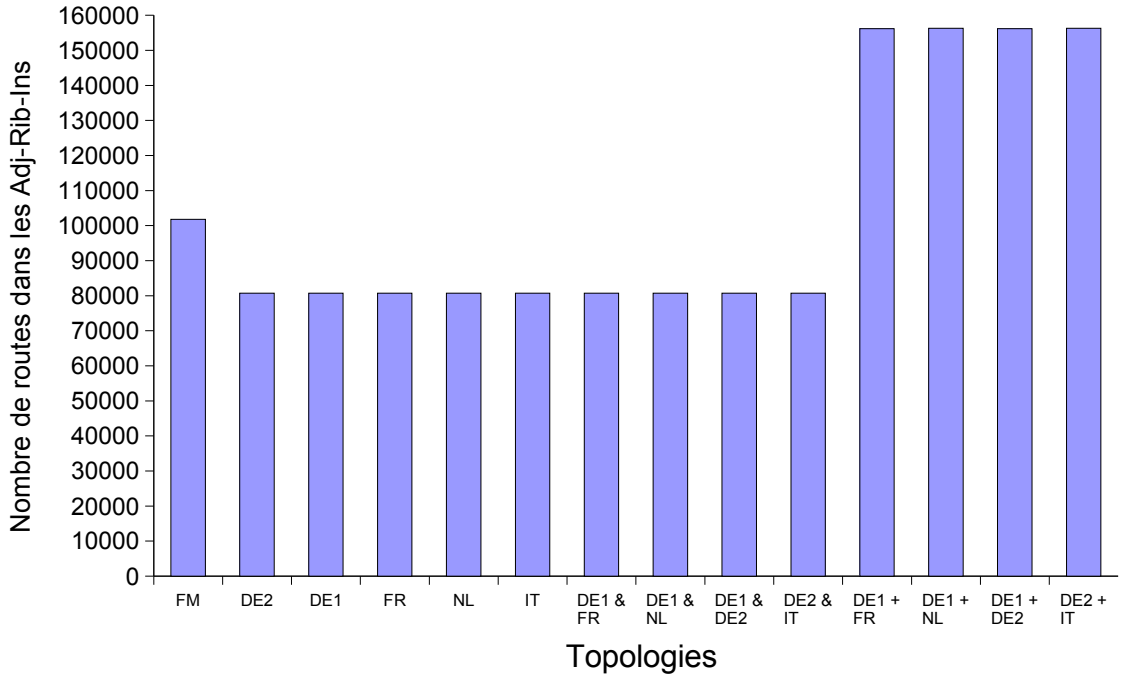

FIG. 6.8 – Taille des tables de routage Adj-Rib-Ins

panne d'un des deux réflecteurs.

Nombre de doubles routes

Comme expliqué plus haut, le nombre de doubles routes a été calculé sur base de routes vers la même destination ayant des routeurs de sortie différents. Nous pouvons immédiatement constater en observant la figure 6.9 l'énorme différence entre le nombre de doubles routes dans une topologie de type *Full Mesh* et une topologie sans redondance, puisque cette valeur passe d'environ 9000 routes de backup à seulement 1500 routes, concentrées sur un petit nombre de routeurs. Il y a donc perte de plus de 80% de la redondance du système à cause du choix effectué par les réflecteurs de routes. Dans le cas des réflecteurs avec redondance, cette perte est moindre, et varie en fonction de la localisation de ces réflecteurs. La topologie avec les routeurs DE2 et DE1 comme réflecteurs redondants est ainsi nettement moins performante au niveau de ce critère, puisqu'elle possède près de trois fois moins de routes doubles que les autres topologies avec redondance.

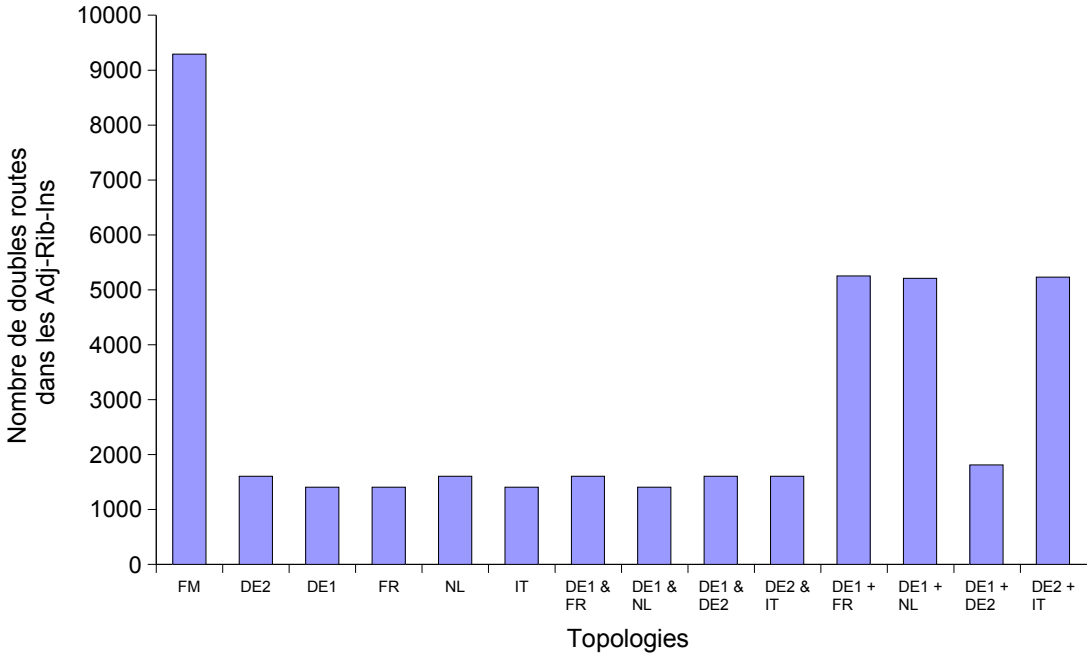

FIG. 6.9 – Nombre de doubles routes dans les Adj-Rib-Ins

Ce phénomène s'explique facilement en considérant la topologie physique de GEANT (figure 5.3) et les routes annoncées sur chacune des sessions. Etant donné que les routes des clients seront généralement uniques, elles n'influenceront en effet pas la topologie iBGP. En ce qui concerne les pairs, seules les sessions avec Abilene annoncent des routes semblables, et ce aux routeurs NY et DE2. Etant donné la distance IGP séparant NY du reste du réseau, il est logique que les routes d'Abilene qu'il annonce ne soient jamais choisies, quel que soit le réflecteur de routes, du moment qu'il ne s'agit pas de NY lui-même. Les routes qui vont influencer les performances des différentes topologies sont donc celles annoncées par les fournisseurs d'accès à Internet. Elles concernent six sessions, qui annoncent chacune les mêmes deux cents destinations.

Il est dès lors logique que la topologie avec DE1 et DE2 en redondance possède nettement moins de routes doubles que les autres, puisque les deux réflecteurs choisissent le même point de sortie vers un fournisseur, à savoir DE1. Dans le cas des autres topologies avec redondance, les réflecteurs annonceront chaque fois les routes des fournisseurs avec des points de sortie différents. Le routeur FR choisit en effet les routes annoncées

par CH, tandis que DE1 annonce celles qu'il a reçues sur ses sessions eBGP. Pour la topologie $IT + DE2$, les points de sortie sont IT et DE1, et enfin, pour $NL + DE1$, il s'agit de UK et DE1.

Somme des distances vers les points de sortie des routes

Les différences entre les valeurs de cette métrique pour chacune des topologies sont relativement faibles, de l'ordre de 1%, c'est pourquoi nous les présenterons pour plus de clarté sous forme normalisée (figure 6.10). Ces différences sont néanmoins suffisantes pour pouvoir sélectionner une topologie présentant une perte d'optimalité minimale par rapport à la topologie *Full Mesh* de référence. Il ne semble pas y avoir de différences énormes entre les différents types de topologies, et les différences de valeurs dépendent surtout de la localisation des réflecteurs dans l'AS.

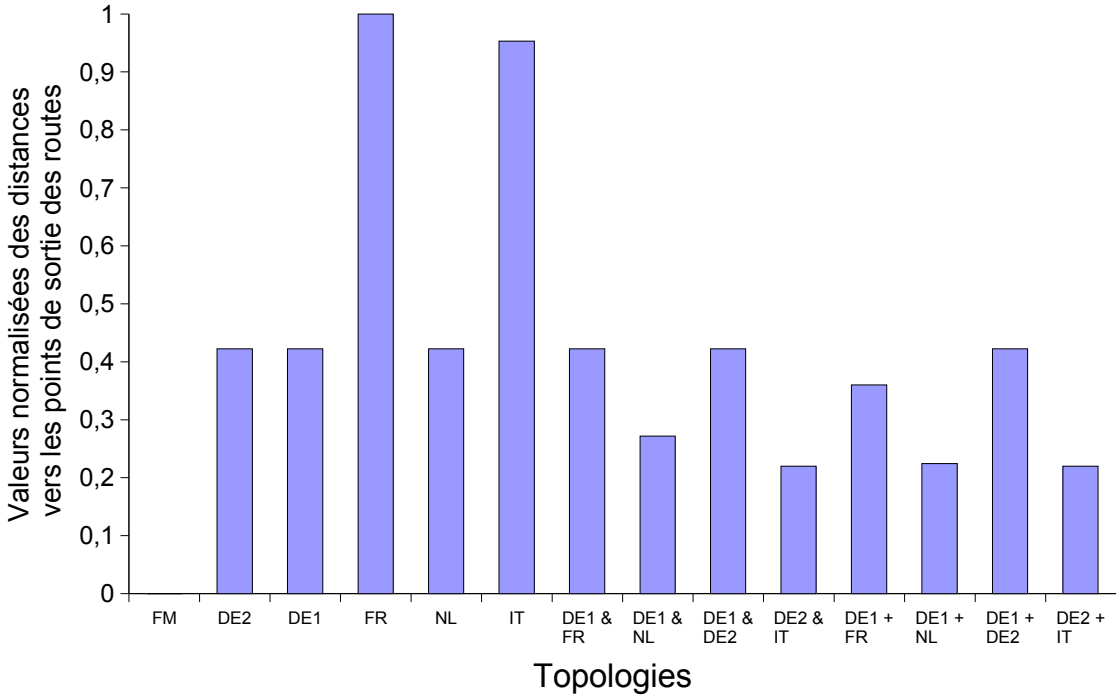

FIG. 6.10 – Valeurs normalisées des différences par rapport au *Full Mesh* des sommes des distances vers les points de sortie des routes

L'analyse de ces valeurs s'effectue suivant un raisonnement similaire à celui suivi dans le cas du nombre de routes doubles. En effet, l'augmentation de la somme des distances vers les points de sortie des routes dépend du choix du point de sortie vers les fournisseurs d'accès Internet choisi par le ou les réflecteurs de routes. Si ce choix est contraire à celui des clients, les paquets devront suivre un plus long chemin pour sortir de l'AS. Les routeurs FR et IT, en tant que réflecteurs uniques, ne sont guère performants de ce point de vue. Ils choisissent en effet les points de sortie situés en IT et CH, qui ne sont pas les routeurs les plus au centre du réseau. Par contre, les topologies avec deux réflecteurs en redondance choisissant des points de sortie différents semblent limiter la perte d'optimalité au niveau du trajet des paquets dans l'AS, puisqu'ils offrent à leurs clients l'opportunité de choisir entre deux points de sortie celui qui leur est le plus proche.

Comparaison des évolutions des métriques et des performances de chaque routeur

Après avoir normalisé l'ensemble des résultats pour que les valeurs soient comprises entre 0 et 1 et puissent ainsi être comparées, nous obtenons le graphique de la figure 6.11. Le *Full Mesh* possède évidemment les valeurs maximales pour les métriques de longueur des sessions et de nombre de doubles routes, et la valeur minimale pour la mesure de la perte d'optimalité en termes de distances vers les points de sortie des routes. Nous recherchons ici une topologie qui soit la plus proche possible du *Full Mesh* en terme d'optimalité et de robustesse, tout en diminuant au maximum la somme des longueurs des sessions. En ce qui concerne la taille des tables de routage, l'idéal serait qu'elle soit la plus faible possible.

Deux de ces métriques sont intrinsèquement liées au type de topologie utilisé. Il s'agit de la somme des longueurs des sessions et des tailles des tables de routage. En effet, d'après les figures 6.7 et 6.8, ces valeurs sont doubles dans le cas des topologies avec redondance par rapport aux cas sans redondance. Cependant, la première de ces métriques est très différente dans les topologies avec réflecteurs par rapport au *Full Mesh*, dénotant un gain appréciable en termes de nombre de sessions iBGP, tandis que la seconde ne diffère que très peu entre la topologie *Full Mesh* et les topologies sans redondance. Par conséquence, alors que, dans le cas des longueurs des sessions, l'augmentation de la métrique lors de l'ajout de redondance reste négligeable par rapport au gain par rapport au *Full Mesh*, dans le cas de la taille des tables de routage, elle reflète une perte de performances assez importante. Néanmoins, cette perte est compensée par une robustesse accrue en cas de panne d'un réflecteur de routes.

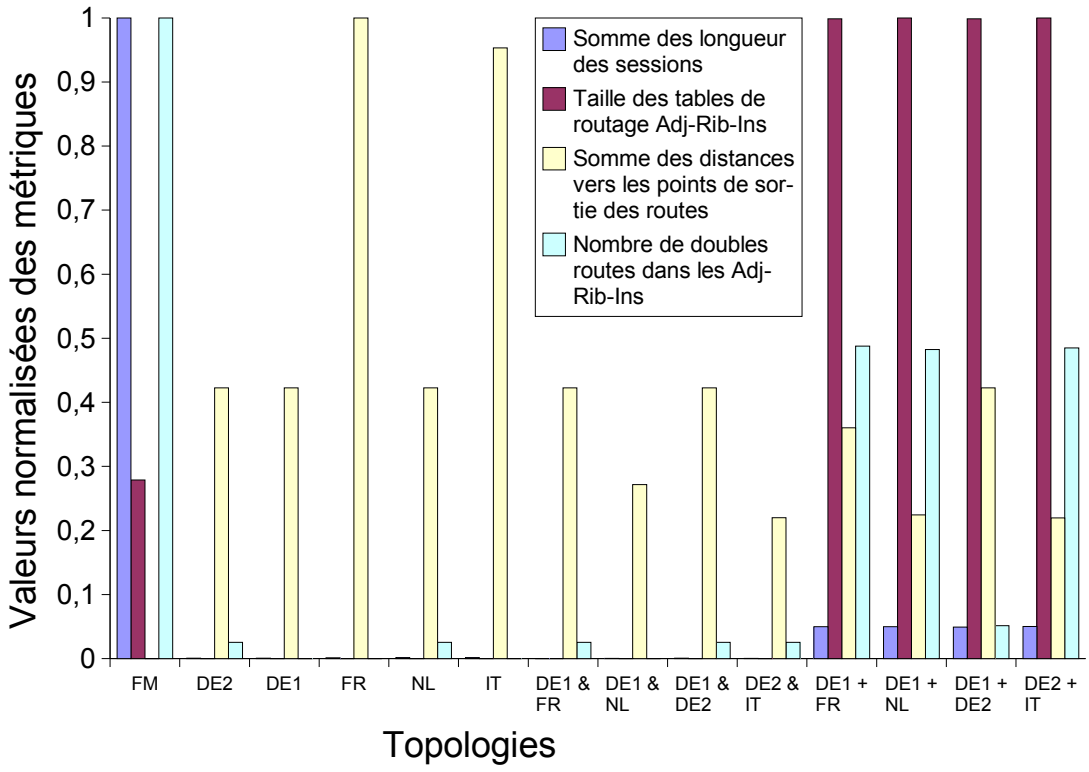

FIG. 6.11 – Valeurs normalisées des quatre métriques

En ce qui concerne la mesure de la perte d'optimalité, les variations observées entre les valeurs des différentes topologies proviennent essentiellement de la manière dont les points de sortie vers les fournisseurs d'accès à Internet sont choisis. Dans le cas des topologies avec un seul réflecteur, il apparaît que ce choix ne peut être vraiment conforme à celui qu'auraient effectués les clients, ce qui explique les performances moindres observées au niveau des distances vers les points de sortie des routes. Ce phénomène est lié au fait que les clients n'apprennent du réflecteur qu'une seule route vers chaque destination, ce qui limite leur choix de point de sortie.

Les topologies avec deux réflecteurs sans redondance sont en moyenne un peu plus performantes, toujours en fonction du choix effectué par les réflecteurs. S'ils choisissent tous deux le même point de sortie, la situation ne sera guère différente du cas avec

un seul réflecteur. Par contre, si ce choix est différent, il est plus probable que chaque réflecteur effectuera un choix conforme à celui qu'auraient fait ses clients, puisque ces derniers se trouvent plus proche de lui que de l'autre réflecteur. Les meilleures topologies en ce qui concerne le choix des routes restent cependant les topologies avec réflecteurs en redondance effectuant des choix de points de sortie différents, puisqu'elles offrent en général deux choix de routes différents à leur client dès qu'il existe dans l'AS deux routes vers la même destination provenant de fournisseurs d'accès.

Choix d'une topologie

L'analyse que nous venons de réaliser ne permet bien entendu pas d'effectuer un choix immédiat de topologie iBGP. Ce choix dépend en effet de l'importance qu'apporte l'administrateur de l'AS aux différents critères. S'il désire avant tout réduire le nombre de sessions iBGP, il optera probablement pour une topologie avec un réflecteur, et, dans ce cas, les routeurs DE1 et DE2 semblent être de bons candidats à la réflection de routes, puisque les topologies correspondantes minimisent la perte d'optimalité et les longueurs des sessions iBGP parmi les topologies de ce type. Les topologies avec deux réflecteurs de routes non redondants n'apportent quant à elles pas grand chose par rapport aux topologies avec un seul réflecteur. Elles peuvent légèrement augmenter les performances, mais impliquent que l'administrateur doive déployer un réflecteur de routes supplémentaire.

Enfin, si l'administrateur accorde plus d'importance à la robustesse de son système, il choisira une topologie à deux réflecteurs redondants. Il s'immunise ainsi contre les problèmes entraînés par la panne d'un réflecteur, avec en contrepartie un doublement de la taille des tables de routage et de la longueur totale des sessions iBGP par rapport aux topologies sans redondance. Parmi les topologies de ce type, celles avec DE1 et IT, et DE1 et NL sont les plus performantes d'après la figure 6.11, tant au niveau de la redondance des routes que de la perte d'optimalité de ces dernières.

6.7 Conclusion de l'analyse

La démarche présentée tout au long de ce chapitre nous a permis d'étudier différents types de topologies dans le cadre du réseau GEANT en nous basant sur quatre critères d'évaluation. Nous sommes dès lors en mesure de fournir aux administrateurs de réseaux un moyen de sélectionner une topologie iBGP idéale, sur base de leurs préférences en termes d'efficacité et de robustesse. Ces préférences détermineront le type de topologie à utiliser, et il sera alors possible de calculer la topologie de ce type qui convient le mieux et d'en générer la configuration.

Conclusion

Résultats

Le moment est à présent venu de conclure ce travail, et de faire le point sur les résultats qui ont été obtenus. L'objectif présenté au tout début de ce document a en effet été atteint : nous avons conçu et implémenté un outil de configuration BGP sur base d'un langage de définition de politiques de haut niveau. Il s'agit certes d'un prototype, mais il montre néanmoins que, grâce à ce langage, il est possible d'exprimer un grand nombre de politiques à partir de quelques primitives simples. Une étude de cas sur le réseau GEANT nous a de plus permis de montrer l'implémentation effective de ces politiques grâce aux fichiers de configuration produits, confirmant par là même l'obtention d'un outil simple d'application de politiques.

Nous avons également ajouté à cet outil de configuration un module permettant d'étudier une topologie iBGP idéale pour un réseau donné. Nous avons pour cela défini une procédure permettant de construire des topologies dites candidates, en nous restreignant à des topologies avec un ou deux réflecteurs de routes. Cette procédure nous permet notamment de choisir les routeurs qui seront potentiellement les plus aptes à assurer la réflection des routes, c'est-à-dire ceux pour qui la topologie iBGP résultante sera la plus proche de la topologie physique. De plus, la manière dont nous construisons ces topologies nous garantit la convergence de BGP à l'intérieur de l'AS, sur base de l'étude réalisée sur les incorrections d'iBGP pouvant résulter de l'utilisation de réflecteurs de routes.

Nous avons enfin défini quatre métriques permettant d'évaluer ces topologies candidates au niveau de leur efficacité, de leur robustesse, et de la perte d'optimalité qu'elles peuvent induire. Nous avons ensuite utilisé l'entièreté de ce module pour étudier les différentes possibilités de topologies iBGP dans le réseau GEANT. Après analyse, il en est ressorti qu'il était difficile de porter son choix sur une topologie en particulier, sans avoir d'autres indications sur les préférences du gestionnaire du réseau ou sur la qualité

des liens de ce dernier. En effet, certaines topologies sont plus efficaces que d'autres, mais sont en contrepartie beaucoup plus sensibles à la panne de certains liens, ou de certains routeurs. Il est donc nécessaire d'indiquer à l'outil quels sont les critères auxquels le gestionnaire attache le plus d'importance pour que ce dernier puisse effectuer un choix définitif de topologie, et générer le fichier correspondant pour obtenir une configuration BGP complète.

Améliorations futures

Comme nous l'avons mentionné plus haut, l'outil que nous avons développé n'est encore qu'à l'état de prototype, et, si ce dernier nous a montré son utilité, il reste encore quelques étapes à accomplir avant de le rendre totalement opérationnel, c'est-à-dire utilisable pour configurer entièrement un véritable AS.

Une première étape consistera à étudier les principales politiques utilisées par les administrateurs BGP dans leurs AS, et d'enrichir le langage pour qu'il soit capable de les exprimer. Quelques pistes ont déjà été présentées au chapitre 5, et, si elles n'ont pas encore été implémentées afin d'être supportées par le compilateur, elles démontrent qu'il est tout à fait possible d'étendre le langage pour y ajouter plus d'expressivité.

Le prototype qui a été développé ne construit pour le moment que des fichiers de configuration pour le simulateur C-BGP. Il est donc indispensable de modifier le compilateur pour qu'il utilise une syntaxe compréhensible par des routeurs BGP réels. Cette étape ne présente néanmoins que très peu de difficultés, puisque les différentes syntaxes de configuration BGP reposent toutes sur les mêmes principes de base. Il existe de plus déjà certains outils permettant de passer d'une syntaxe à une autre, et il sera assez simple de les adapter et de les utiliser avec notre outil.

Enfin, si les métriques définies pour évaluer des topologies iBGP sont tout à fait générales, nous ne les utilisons pour le moment que sur quelques topologies relativement simples, ne convenant pas à des AS de grande taille. Des recherches ultérieures pourraient donc réaliser des analyses similaires sur des topologies à plusieurs niveaux de réflecteurs de routes, ou bien sur des topologies 'hybrides', c'est-à-dire des topologies ajoutant de la redondance dans les tables de routage de certains clients, mais pas dans d'autres. Cela permettrait donc de rendre l'outil applicable à tout type de réseaux, et notamment à ceux de grande envergure.

Bilan personnel

Pour les quelques lignes clôturant ce travail, je me permettrai de contrevenir aux conventions en m'exprimant à la première personne du singulier, afin de faire le point sur la manière dont ce mémoire à été produit. Ce sont en effet près de dix mois de travail qui arrivent à présent à leur terme, parsemés de périodes de réflexion plus ou moins intenses, de moments de doutes ou de satisfaction devant les résultats produits, et surtout, d'une pile assez conséquente de notes en tout genre.

Mener à bien un travail d'une ampleur nettement plus importante que les projets réalisés lors des quatre années précédentes fut une expérience particulièrement intéressante, et même réellement passionnante. J'ai en effet énormément apprécié le défi que représentait le fait de devoir mener à bien ce travail, et cela m'a beaucoup apporté : en termes d'organisation, d'abord, car il n'a pas toujours été facile de garder les idées claires quant à la démarche à suivre. En termes de connaissances générales ensuite, car j'abordais un domaine dans lequel celles-ci étaient relativement basiques, et j'ai découvert combien les concepts et la logique liés au protocole BGP était passionnants. Et enfin, en termes de recherche, car parcourir la littérature pour tenter de trouver des éléments de réponse à certains problèmes n'est pas toujours évident.

Au delà du travail effectué, ce mémoire représente également le point d'orgue et l'aboutissement de cinq années d'étude qui se sont révélées extrêmement riches en enseignements, aussi bien au niveau des connaissances et de la méthodologie que de toutes les activités extra-académiques qui les ont accompagnées et complémentées. C'est donc avec une certaine émotion et une pensée reconnaissante pour tous ceux qui m'ont aidée et accompagnée aussi bien dans ce travail que durant l'ensemble de ces cinq années que j'appose enfin le point final à ce texte.

Bibliographie

[1] W. Stallings, *SNMP, SNMP v2, and CMIP*. Addison-Wesley, 1993.

[2] M. Rose et K. McCloghrie, “Structure and Identification of Management Information for TCP/IP-based Internets,” RFC 1155, IETF, 1990.

[3] M. Rose et K. McCloghrie, “Concise MIB definitions,” RFC 1212, IETF, 1991.

[4] M. Rose et K. McCloghrie, “Management Information Base for Network Management of TCP/IP-based internets : MIB-II,” RFC 1213, IETF, 1991.

[5] R. Enns, “NETCONF Configuration Protocol,” Internet-Draft Version 3, Internet Engineering Task Force, June 2004. Expired.

[6] M.-J. Choi, H.-M. Choi, J. Hong, et H.-T. Ju, “XML-based configuration management for IP network devices,” *IEEE Communications Magazine*, vol. 42, no. 7, pp. 84–91, 2004.

[7] NETCONF : http ://www.ops.ietf.org/netconf/.

[8] J. Strassner, *Policy-based network management - Solutions for the next generation*. Morgan Kaufmann, 2004.

[9] R. Mahajan, D. Wetherall, et T. Anderson, “Understanding BGP misconfigurations,” in *Proceedings of ACM SIGCOMM*, pp. 3–16, 2002.

[10] Y. Rekhter, “A Border Gateway Protocol 4 (BGP-4),” internet draft, IETF, 2002. draft-ietf-idr-bgp4-18.txt.

[11] R. White, D. McPherson, et S. Sangli, *Practical BGP*. Addison Wesley, 2004.

[12] R. Zhang et M. Bartell, *BGP Design and Implementation*. Cisco Press, 2003.

[13] R. Chandra, P. Traina, et T. Li, “BGP Communities Attribute,” RFC 1997, IETF, Aout 1996.

[14] P. Traina, D. McPherson, et J. Scudder, “Autonomous System Confederations for BGP,” RFC 3065, IETF, Février 2001.

[15] B. Quoitin et S. Tandel, *C-BGP User’s Guide (version 1.1.19)*, Février 2005. http ://cbgp.info.ucl.ac.be/downloads/cbgp-doc.pdf.

[16] L. Gao, “On Inferring Autonomous System Relationships in the Internet,” *IEEE/ACM Transactions on Networking*, vol. 9, pp. 733 – 745, Décembre 2001.

[17] T. Griffin et G. Wilfong, “On the Correctness of IBGP Configuration,” *Computer Communication Review*, vol. 32, pp. 17– 29, October 2002.

[18] CYMRU : http ://www.cymru.com/Documents/bogon-bn-nonagg.txt.

[19] S. Bellovin, R. Bush, T. Griffin, et J. Rexford, “Slowing routing table growth by filtering based on address allocation policies,” June 2001. preprint available from http://www.research.att.com/~jrex.

[20] O. Bonaventure et B. Quoitin, “Common utilizations of the BGP community attribute,” internet draft (expired), IETF, June 2003. draft-bq-bgp-communities-00.txt.

[21] T. Bates, R. Chandra, et E. Chen, “BGP Route Reflection - An Alternative to Full Mesh IBGP,” RFC 2796, IETF, Avril 2000.

[22] T. Griffin et G. Wilfong, “Analysis of the MED Oscillation Problem in BGP,” in *Proceedings of the 10th IEEE International Conference on Network Protocols*, pp. 90–99, IEEE Computer Society, 2002.

[23] N. Feamster, “Practical verification of wide-area routing,” in *Workshop on Hot Topics in Networks (HotNets)*, 2003.

[24] RAToolSet : http ://www.isi.edu/ra/RAToolSet/RtConfig.html.

[25] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, et M. Terpstra., “Routing Policy Specification Language (RPSL),” RFC 2622, IETF, Juin 1999.

[26] B. Quoitin. C-BGP : http ://cbgp.info.ucl.ac.be/.

[27] JavaCC : https ://javacc.dev.java.net/.

[28] R. Dube et J. G. Scudder, “Route Reflection Considered Harmful,” internet draft, IETF, November 1998. draft-dube-route-reflection-harmful-01.txt, Work in Progress.

[29] T. Griffin, F. B. Shepherd, et G. Wilfong, “The stable paths problem and interdomain routing,” *IEEE/ACM Transactions on Networking*, 2002.

[30] J. Sobrinho, “Network Routing with Path Vector Protocols : Theory and Applications,” in *Proc. ACM SIGCOMM 03*, 2003.

[31] L. Xiao, J. Wang, et K. Nahrstedt, “Optimizing IBGP route reflection network,” in *IEEE International Conference on Communications*, 2003.

Annexes

Annexe A
NETCONF

Cette annexe a pour but de présenter en détails la syntaxe des balises XML utilisées par NETCONF, notamment à l'aide de quelques exemples d'utilisation de ces dernières. Elle détaillera également les différentes *capabilities* définies dans le protocole.

Syntaxe des balises

Modèle RPC : Exemple d'échange de messages

Cet exemple permet de montrer l'utilisation de la balise <rpc>. Dans ce cas, l'absence de l'attribut *message-id* déclenche l'envoi d'un message d'erreur signalant le problème. Ce message d'erreur est contenu dans une balise <rpc-reply>

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
  </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>rpc</error-type>
    <error-tag>MISSING_ATTRIBUTE</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <bad-attribute>message-id</bad-attribute>
      <bad-element>rpc</bad-element>
```

```
    </error-info>
  </rpc-error>
</rpc-reply>
```

Opérations : Détails de syntaxe et exemples

Gestion des configurations et des informations d'état :

- <get-config> Cette opération permet de récupérer une partie ou l'entièreté d'une configuration, en indiquant en paramètre le nom du datastore considéré ainsi qu'un filtre indiquant la partie de la configuration à laquelle on veut accéder. Si l'opération s'effectue correctement, le <rpc-reply> envoyé en réponse contiendra un élément <config> contenant les informations demandées. Ses arguments sont :

  - *source* : Nom du datastore de configuration condidéré (ex : <running>)
  - *filter* : Element identifiant la partie de la configuration à laquelle on veut accéder. Si ce paramètre n'est pas spécifié, l'entièreté de celle-ci est renvoyée.
- <edit-config> Cette opération offre la possibilité d'éditer, de remplacer, de créer ou d'effacer une partie ou l'entièreté d'une configuration dans l'appareil considéré. L'appareil analyse la configuration source et la configuration cible pour effectuer les changements demandés. La configuration cible n'est donc pas simplement remplacée, comme elle le serait avec le message <copy-config> qui sera considéré plus loin. Cette opération possède un attribut et quatre paramètres :
  - Attribut '*operation*' : Cet attribut spécifie l'opération à appliquer au morceau de configuration considéré. Les valeurs possibles sont 'merge', 'replace', 'create' et 'delete'. La première, qui est la valeur par défaut, permet d'intégrer la modification à la configuration existante sans influencer les parties de cette dernières qui ne sont pas concernées. La valeur 'replace' remplace totalement la partie de la configuration considérée, 'create' crée un nouvel élément de configuration ou renvoie un <rpc-reply> contenant 'DATA_EXISTS' si il existe déjà, et enfin 'delete' supprime le morceau de configuration spécifié. Cet attribut apparait dans la balise XML de l'élément cible, permettant ainsi d'appliquer l'opération spécifiée à plusieurs sous-ensembles de la configuration. Cependant, dans un souci de simplicité, tous les attributs 'operation' apparaissant dans le même sous-arbre délimité par une balise <config> doivent avoir la même valeur.
  - Paramètre '*target*' : Ce paramètre précise le nom du datastore de configuration à éditer.
  - Paramètre '*test-option*' : Cet élément ne peut apparaître que si l'appareil a annoncé qu'il supportait la validation de configuration (validate capability, voir plus loin). Il peut prendre les valeurs 'test then set' ou 'set', indiquant s'il faut valider

la modification demandée avant de l'effectuer ou non.
  - Paramètre '*error-option*' : Il prend les valeurs 'stop on error' (par défaut), 'ignore error' ou 'rollback on error'. Il permet donc d'indiquer l'attitude à adopter par l'appareil si l'édition de la configuration produit un erreur, à savoir, arrêter l'opération, ignorer l'erreur ou supprimer les changements déjà effectués.
  - Paramètre '*config*' : Ce paramètre indique la portion de la configuration à éditer. Il doit avoir comme attribut le namespace de la configuration en question.
- <copy-config> A la différence d'<edit-config>, <copy-config> agit sur l'entièreté d'un datastore de configuration, créant ou remplaçant la totalité de la configuration. En raison de son caractère critique, un appareil peut choisir de ne pas permettre l'application de cette opération sur la configuration courante.
  Elle possède les paramètres suivants :
  - *target* : Spécifie le datastore de configuration destiné à contenir la copie
  - *source* : Spécifie le datastore de configuration à copier. Il doit être différent du datastore cible.
- <delete-config> Cette opération a pour effet de supprimer le datastore de configuration spécifié par le paramètre target. Ce dernier ne peut être le datastore <running>.

Exemple : L'exemple présenté ci-dessous utilise l'opération <edit-config> pour modifier le MTU d'une interface nommée 'Ethernet0/0'. L'argument 'operation' est utilisé avec la valeur 'merge' bien que celle-ci soit la valeur par défaut, afin de montrer l'utilisation de ce dernier comme argument d'une balise <config>. La structure de la configuration de l'appareil à qui est destiné la requête est définie par le namespace XML identifié par l'URI 'http ://example.com/schema/1.2/config'.

```
<rpc message-id ="'107"'
     xmlns="'urn:ietf:params:xml:ns:NETCONF:base:1.0"'>
  <edit-config>
    <target>
      <running/>
    </target>
      <config xmlns="'http://example.com/schema/1.2/config"'
              xmlns:xc="'urn:ietf:params:xml:ns:NETCONF:base:1.0"'>
        <interface xc:operation="'merge"'>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </config>
```

```
  </edit-config>
</rpc>
```

La réponse RPC contiendra le même identifiant de message que la requête qui lui correspond ; à savoir 107, et reprendra également l'attribut xmlns de cette dernière. Puisqu'il s'agit d'une opération <edit-config>, la réponse attendue est la balise <ok/>. Dans le cas d'une opération <get-config>, elle aurait contenu le sous-arbre de la configuration demandé.

```
<rpc-reply message-id ="'107"'
           xmlns="'urn:ietf:params:xml:ns:NETCONF:base:1.0"'>
  </ok>
</rpc-reply>
```

– <get> Cette opération est semblable à <get-config>, excepté le fait qu'elle permet en plus d'accéder à des informations d'état. Au lieu d'un élément <config>, <get> renverra les informations dans un élément <data> contenu entre des balises <rpc-reply>. Il est possible d'utiliser un filtre pour spécifier les informations auxquelles on veut accéder. Si rien n'est indiqué, la réponse contiendra à la fois la configuration de l'appareil et les informations d'état.

Opérations pour la gestion des ressources Afin de pouvoir modifier une configuration sans qu'un autre client n'accède en même temps à la ressource considérée, risquant ainsi de rendre la configuration incohérente, NETCONF définit des opérations permettant de verrouiller les ressources et de garantir un accès exclusif à ces dernières. Les verrous sont prévus pour être de courte durée, et garantissent l'absence d'interaction aussi bien avec d'autres clients NETCONF qu'avec des clients utilisant d'autres technologies de configuration comme CLI ou SNMP. Les éventuelles requêtes effectuées par ces derniers doivent donc automatiquement échouer. Ils sont globaux, et agissent sur l'entièreté d'une configuration, ce qui signifie que si un verrou d'une entité non NETCONF est déjà posé sur ne fut-ce qu'une partie de celle-ci, il sera impossible d'y poser un verrou NETCONF.

Un verrou est maintenu jusqu'à ce qu'il soit explicitement relâché, ou jusqu'à ce que la session durant laquelle il a été posé se termine. Même si la session se termine de manière anormale, c'est-à-dire sans avoir été explicitement fermée par le client (par exemple suite à l'expiration d'un timeout), les ressources verrouillées au cours de celle-ci doivent être libérées. Il est donc possible de déverrouiller une ressource dont le verrou est détenu par une autre session en utilisant par exemple l'opération <kill-session> pour terminer cette dernière.

– <lock> Cette opération permet d'acquérir un verrou sur le datastore de configuration spécifié par le paramètre <target>. Par défaut, ce dernier aura la valeur <running/>. L'acquisition d'un verrou échouera si la ressource est déjà verrouillée par une autre entité(session NETCONF ou autre), si le verrouillage de ressources n'est pas supporté par l'appareil, ou bien encore si la configuration cible a été modifiée, mais que ces changements n'ont pas encore été commis. Dans le cas d'un échec de la requête, le message <rpc-reply> contiendra un <rpc-error> spécifiant éventuellement l'identifiant de session du propriétaire du verrou si la ressource est déjà verrouillée. S'il ne s'agit pas d'une session NETCONF, cet identifiant sera nul. Si le verrouillage s'effectue correctement, l'élément <ok> est envoyé.

– <unlock> <unlock> est utilisé pour relâcher un verrou précédemment posé. Il possède le paramètre <target> spécifiant le datastore de configuration à libérer (running par défaut), et renvoie un <rpc-reply> contenant soit <ok> soit <rpc-error>, en fonction du succès ou de l'échec de l'opération. Cette dernière peut en effet échouer notamment pour les deux raisons suivantes : Si le verrou spécifié n'est plus actif, ou si la session effectuant l'opération de déverrouillage n'est pas la même que celle qui a effectué le verrouillage de la ressource.

Exemple Pour poser un verrou sur le datastore <running/>, les messages suivant sont échangés :

```
<rpc message-id ="108"
     xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <lock>
    <target>
      <running/>
    <target/>
  </lock>
</rpc>

<rpc-reply message-id ="108"
           xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <ok/>
</rpc-reply>
```

Si la session 454 possède déjà un verrou sur le même datastore, l'échange sera le suivant :

```
<rpc message-id ="109"
```

```
     xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <lock>
    <target>
      <running/>
    <target/>
  </lock>
</rpc>

<rpc-reply message-id ="109"
           xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>LOCK_DENIED</error-tag>
    <error-severity>error</error-severity>
    <session-id>150</session-id>
    <error-message>Lock failed, lock is already held</error-message>
      <error-info>
        <session-id>454</session-id>
      </error-info>
  </rpc-error>
</rpc-reply>
```

Opérations contrôlant les sessions NETCONF Comme mentionné plus haut, une session NETCONF est une connexion logique entre un administrateur réseau ou une application de configuration de réseau, et un appareil d'un réseau. Si l'établissement des sessions et le processus d'identification qui en découle est laissé à la responsabilité du protocole applicatif, NETCONF fournit par contre des opérations permettant de les terminer, que ce soit de manière normale ou abrupte.

De la même manière que pour les opérations précédentes, la réponse <rpc-reply> correspondant à la requête contiendra un élément <ok> en cas de succès de l'opération, et un élément <rpc-error> en cas d'échec.

- <close-session> Cette opération permet de terminer normalement une session NETCONF. Lorsqu'une entité reçoit ce message, elle va terminer les opérations en cours puis libérer les verrous et les ressources utilisés par la session avant de fermer toutes les connexions qui y sont liées.

- <kill-session> permet quant à lui de terminer abruptement une session à partir d'une autre. La session à terminer est identifiée par le paramètre <session-id>. Celui-ci doit être différent de l'identifiant de la session courante, sans quoi une er-

reur 'Bad Value' est renvoyée. Les opérations en cours dans la session cible seront immédiatement interrompues, les ressources et verrous qu'elle détient relachés et toutes les connexions ouvertes par la session fermées.

Capabilities

NETCONF définit une *capability* comme un ensemble de fonctionnalités implémentées au dessus des spécifications de base. Tout client ou serveur NETCONF a le choix d'éventuellement supporter certaines *capabilities*, ou encore d'ignorer des *capabilities* supportées par son interlocuteur. Elles peuvent être définies suivant une structure particulière, et être publiées comme définitions standards ou comme définitions propriétaires.

Les *capabilities* sont identifiées par des URI, et annoncées par les interlocuteurs d'une session NETCONF au début de celle-ci. Pour cela, chacun d'eux envoie un élément <hello> contenant une liste des *capabilities* qu'il supporte. Ces éléments <hello> sont envoyés simultanément dès que la connexion est ouverte, sans attendre la réception des *capabilities* de l'autre interlocuteur.

Exemple Dans l'exemple suivant, un interlocuteur annonce les *capabilities* NETCONF de base, une *capability* NETCONF supplémentaire (startup), et une *capability* propriétaire.

```
<hello>
<capabilities>
<capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
<capability>urn:ietf:params:xml:ns:netconf:base:1.0#startup</capability>
<capability>http://example.net/router/2.3/core#cool-feature</capability>
</capabilities>
</hello>
```

Les *capabilities* NETCONF sont toutes identifiées par l'URI suivante, le suffixe *name* représentant le nom de la *capability* :

```
urn:ietf:params:xml:ns:netconf:base:1.0#name
```

Les principales capabilities sont reprises ci-dessous :

Writable-Running Cette *capabilities* indique que l'appareil qui la supporte autorise les modifications sur le datastore de la configuration courante, <running/>. Elle modifie donc les opérations <edit-config> et <copy-config> en leur permettant d'avoir <running/> comme valeur du paramètre <target>.

Candidate Configuration Cette extension autorise l'utilisation du datastore de configuration 'candidate', qui est utilisé pour modifier des éléments de configuration sans influencer le fonctionnement courant de l'appareil. Il sert donc de brouillon avant d'effectuer les changements définitif grâce à l'opération <commit>, qui applique la configuration définie dans le datastore 'candidate' au datastore <running/>. Lorsque cette *capability* est supportée, le datastore <candidate> peut être utilisé comme source ou destination de n'importe quel opération agissant sur un datastore.

Cette configuration peut être partagée entre plusieurs sessions, et il faut donc tenir compte de l'éventuelle présence d'autres clients travaillant sur le même datastore en utilisant des verrous pour éviter les incohérences. L'élément <candidate> peut donc être utilisé comme valeur du paramètre <target> des opérations <lock> et <unlock>, et devient également la valeur par défaut de ces dernières.

Un client peut annuler tous les changements effectués depuis le dernier <commit> en utilisant l'opération <discard-change>.

Confirmed Commit Cette *capability* ne peut être supportée que si la *capability* Candidate Configuration l'est également. Elle permet l'utilisation des paramètres <confirmed> et <confirm-timeout> avec l'opération <commit>. Le premier de ces paramètre indique que l'opération doit être annulée s'il n'a pas été confirmé dans les 10 minutes. Le second permet de changer la valeur par défaut en minutes de ce timeout. Cette *capability* risque d'influencer d'autres sessions dans le cas de configuration partagées, et il est donc recommendé d'utiliser des verrous pour éviter tout problème.

Rollback on Error Cette extension indique que l'agent est capable d'annuler les changements effectués si jamais l'opération qui les a initié échoue. Elle permet donc au paramètre <error-option> de l'opération <edit-config> de prendre la valeur 'rollback-on-error'. Etant donné que cette *capability* peut avoir des répercussions sur d'autres sessions, il est nécessaire d'utiliser des mécanismes de verrouillage.

Validate La validation consiste à vérifier une configuration candidate afin de détecter d'éventuelles erreurs syntaxiques ou sémantiques avant de l'appliquer à l'appareil. La *capability* validate requiert que l'appareil qui la supporte supporte au minimum la vérification syntaxique. Elle définit également une nouvelle opération, <validate>, qui possède un paramètre <source> indiquant le nom du datastore sur lequel il faut effectuer la vérification. Cette opération peut échouer si la configuration contient des erreurs de syntaxe, si des paramètres sont manquants, ou s'il y a des références à des données de configuration indéfinies.

Startup Un appareil supportant cette *capability* supporte un datastore de configuration supplémentaire, <startup>, qui définit la configuration par défaut lors de la mise en route de l'appareil. Pour mettre à jour cette configuration, il faut utiliser l'opération <copy-config> et copier la configuration courante dans la configuration startup.

URL Un agent NETCONF ayant annoncé cette *capability* peut utiliser l'élément <url> dans les paramètres <source> et <target>. L'URI de la *capability* doit contenir un argument "protocol" ayant comme valeur la liste des protocoles supportés par l'agent.

Annexe B
Fichiers de données et sources des programmes