ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE AVEC CONCENTRATION EN RÉSEAUX DE TÉLÉCOMMUNICATIONS
M.Ing.

PAR
OLIVIER TRUONG

ÉTUDE ET DÉVELOPPEMENT D'OUTILS D'OPTIMISATION DE GESTION DE SERVICES DANS LES RÉSEAUX MPLS

MONTRÉAL, LE 26 JUILLET 2006

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE :

Mme Maria Bennani, directrice de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Jean-Marc Robert, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Michel Lavoie, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 25 JUILLET 2006
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ÉTUDE ET DÉVELOPPEMENT D'OUTILS D'OPTIMISATION DE GESTION DE SERVICES DANS LES RÉSEAUX MPLS

Olivier Truong

SOMMAIRE

Beaucoup d'efforts se concentrent actuellement sur l'opération, administration et maintenance des réseaux MPLS, les fournisseurs de service ayant bien compris que pour générer des profits, les services doivent être gérés efficacement. Différents outils ont ainsi été développés par des équipes de recherche et par l'industrie, adressant un ou plusieurs aspects de la gestion de réseaux. Cependant, aucun outil ne permet à lui seul de visualiser la configuration d'un réseau hétérogène MPLS, avec ses applications principales, l'ingénierie de trafic et les VPN. En outre, la problématique de détection des pannes n'est abordée qu'en théorie.

Pour répondre à ce besoin, nous avons conçu un nouvel outil, permettant de fournir à l'administrateur une vue globale de son réseau MPLS via une interface Web. Il s'intègre avec un outil dédié à la qualité de service pour former QoS MPLS Assistant (QMA). La collecte d'informations est définie selon des standards établis, rendant QMA interopérable, fonctionnel quel que soit le constructeur des équipements. Par ailleurs, une méthode de surveillance de réseau a été mise au point, après étude et analyse des mécanismes actuellement proposés ou en cours de développement.

Lors du développement de cet outil, il a fallu tenir compte de l'écart constaté entre standards publiés et implémentations effectives dans les routeurs. De fait, certaines des fonctionnalités de QMA élaborées en théorie n'ont pu être concrètement mises en place. Pour autant, QMA est un outil riche, offrant à l'administrateur un moyen ergonomique et convivial de visualisation d'un réseau MPLS. Il a été testé avec succès sur une plate-forme de tests de notre partenaire industriel Bell Canada.

Conçu avec une approche modulaire, pour faciliter son évolution, QMA pourra être repris dans le cadre d'un autre projet, et inclure de nouvelles fonctionnalités, voire même s'intégrer dans une solution professionnelle de gestion des réseaux.

STUDY AND DEVELOPMENT OF NETWORK MANAGEMENT OPTIMIZATION TOOLS IN MPLS NETWORKS

Olivier Truong

ABSTRACT

Aware that services must be efficiently managed to be profitable, service providers have recently put a lot of effort into operation, administration and maintenance of MPLS networks. Various tools have been implemented in the university and industry world to address one or several network management issues. However no tool can visualize the configuration of a heterogeneous MPLS network, along with its main applications, traffic engineering and VPN. Besides, the issue of fault detection is only tackled at a theoretical level.

This need has led to the design and implementation of a new tool providing the administrator with an overview of their MPLS network through a web interface. It is combined with a QoS tool and is called QoS MPLS Assistant (QMA). Information gathering is defined according to standards, making QMA work regardless of the vendor. A network monitoring method has also been set up after analysis of current or under development mechanisms.

During QMA development, the discrepancy between standards and effective implementation in routers had to be taken into consideration. Thus, some features originally expected could not be integrated. Still, QMA is a powerful tool providing the administrator with a user-friendly way to visualize their MPLS network. It has been successfully tested on a platform of our industrial partner Bell Canada.

Designed with a modular approach to make its evolution easy, QMA could be improved in another project, including new features, and even be integrated into a professional network management solution.

REMERCIEMENTS

La réalisation de ce mémoire a été rendue possible grâce aux laboratoires universitaires de Bell Canada. Je remercie chaleureusement Guy Côté, Nicolas Manen et le reste de l'équipe au sein de Bell Canada, pour leur accueil, leur bonne humeur et leur passion débordante pour les réseaux.

Beaucoup de gratitude va naturellement à ma directrice de mémoire Maria Bennani pour la confiance qu'elle m'a accordée en m'accueillant au sein de ce projet. Je la remercie également pour sa patience, ses conseils et son encadrement pour ce mémoire.

Je remercie aussi les autres membres du projet Bell d'avoir partagé cette aventure, en particulier Marc-André Breton, compagnon d'œuvre de QMA, Laurent Charbonnier, un compatriote d'exception, et Abderrahim Nadifi. Une reconnaissance particulière va à Abdelghani Benharref, Ilka Fauveau et Jessica Sourisseau pour leurs corrections, à Virginie Convert pour ses tests de l'outil, ainsi qu'à Jeff de Subway pour son soutien logistique. Merci également aux membres du laboratoire de gestion des réseaux informatiques et de télécommunications pour leur accueil au sein du laboratoire.

Enfin, je dédie ce mémoire à mon Seigneur Jésus-Christ que je remercie de tout mon cœur pour Sa présence, Sa patience et Sa bonté inconditionnelles. Sans Lui, rien n'aurait été possible.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS ET SIGLES

AIS	Architecture et Intégration de Solutions
ATM	Asynchronous Transfer Mode
BFD	Bidirectionnal Forwarding Detection
CE	Customer Edge
CSPF	Constraint Shortest Path First
CV	Connectivity Verification
DiffServ	Differentiated Services
DLCI	Data Link Connection Identifier
ECMP	Equal Cost MultiPath
FCAPS	Fault Configuration Accounting Performance Security
FEC	Forwarding Equivalence Class
FFD	Fast Failure Detection
FRR	Fast Reroute
GMPLS	Generalized MPLS
GRE	Generic Routine Encapsulation
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secured
IOS	Integrated Operating System
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IIS	Internet Information Services
IGP	Interior Gateway Protocol
ISC	IP Solution Center
ISO	International Organization for Standardization
IP	Internet Protocol
IPX	Internetwork Packet Exchange
ITU	International Telecommunication Union

ITU-T	ITU Telecommunication Standardization Sector
J2EE	Java 2 Entreprise Edition
JavaEE	Java Platform, Enterprise Edition
LDP	Label Distribution Protocol
LER	Label Edge Router
LSP	Label Switched Path
LSR	Label Switched Router
MAN	Metropolitan Area Network
MATE	MPLS Adaptive Traffic Engineering
MIB	Management Information Base
MPLS	Multi Protocol Label Switching
NASA	National Aeronautics and Space Administration
NGN	Next Generation Network
OAM	Opération, Administration et Maintenance
OID	Object Identifier
OSI	Open Systems Interconnection
P	Provider
PDH	Plesiochronous Digital Hierarchy
PE	Provider Edge
PoS	Packet over SDH
PPP	Point-to-Point Protocol
PHP	Penultimate Hop Popping
PVC	Permanent Virtual Connection
QMA	Qos MPLS Assistant
QoS	Quality of Service
RADIUS	Remote Authentication Dial-In User Service
RAM	Random Access Memory
RAN	Regional Area Network
RATES	Routing and Traffic Engineering Server

RFC	Request For Comments
SDH	Synchronous Digital Hierarchy
SDL	Specification and Design Language
SHA	Secure Hash Algorithm
SMS	Short Message Service
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
SSH	Secure Shell
SVC	Switched Virtual Connection
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TFTP	Trivial File Transfer Protocol
TE	Traffic Engineering
TEAM	Traffic Engineering Automated Manager
TLV	Type Length Value
TTSI	Trail Termination Source Identifier
TTL	Time To Live
UDP	User Datagram Protocol
VCI	Virtual Channel Identifier
VPI	Virtual Path Identifier
VPLS	Virtual Private Lan Service
VPN	Virtual Private Network
VRF	VPN Routing and Forwarding
WAN	Wide Area Network
WDM	Wavelength Dense Multiplexing
VPLS	Virtual Private Lan Service

INTRODUCTION

«Sans maîtrise la puissance n'est rien.» Cet adage bien connu de Pirelli s'applique à de nombreux domaines, y compris celui des réseaux informatiques. En effet, alors que les technologies se multiplient et se complexifient, offrant toujours plus, plus vite, il devient crucial de les gérer efficacement, de mieux les maîtriser.

C'est précisément l'objectif de ce mémoire : comprendre la technologie MPLS et améliorer sa gestion, pour permettre une administration de services optimisée. Pour cela, le premier chapitre expose les enjeux de la gestion des réseaux, puis la problématique de ce projet en situant bien le contexte de celui-ci. Ensuite, le second chapitre mène une étude bibliographique. Celle-ci permet au troisième chapitre de proposer un outil avec de nouvelles fonctionnalités pour MPLS. Enfin, les travaux de conception et d'implémentation sont exposés au quatrième chapitre, et l'outil est validé par l'expérimentation au cinquième chapitre.

CHAPITRE 1

PROBLÉMATIQUE

« Un problème sans solution est un problème mal posé. »
Albert Einstein

Dans ce premier chapitre, nous présentons tout d'abord les enjeux de la gestion réseau, en définissant celle-ci d'un point de vue général, puis plus technique. Forts de cette compréhension, nous pourrons alors exposer la problématique abordée dans ce mémoire. Enfin, ce chapitre se conclura par la mise en contexte de ce mémoire dans un projet plus global, afin de bien cerner dans quel cadre il intervient.

1.1 Les enjeux de la gestion réseau

En quelques décennies, l'informatique a su s'imposer comme un élément indispensable pour le traitement et l'échange d'information. D'une part, cet outil permet d'effectuer de nombreux calculs en un temps beaucoup plus court que manuellement. D'autre part, et il s'agit là de la raison majeure du développement informatique, la capacité de faire communiquer des ordinateurs, c'est-à-dire de les mettre en réseau, a ouvert la voie à de nouvelles applications, des possibilités inédites, d'abord sur le plan militaire, puis économique, et enfin personnel. Ainsi, les parcs informatiques des entreprises n'ont cessé de croître, de manière exponentielle quelquefois. Les échanges commerciaux sont facilités, la communication au sein même de l'entreprise améliorée.

Néanmoins, ce développement des infrastructures réseaux, en dépit de tous ses avantages, se heurte à un certain nombre de défis, d'importance parfois proportionnelle à sa taille. La diversité des équipements, leur complexité, ainsi que leur nombre sont autant de paramètres qu'un administrateur réseau doit prendre en compte pour une

gestion efficace. Les défis auxquels l'administrateur réseau doit faire face sont principalement : les pannes des équipements, les attaques contre le réseau, les manœuvres maladroites d'employés, mais aussi, plus fréquemment, la congestion de certains équipements.

Face à ces exigences, l'administrateur réseau doit travailler avec des contraintes opérationnelles. Les enjeux financiers peuvent en effet s'avérer substantiels. Par exemple, une panne de réseau bancaire, bloquant des transactions, pourrait coûter des milliers voire des millions de dollars à la banque et ses clients. Le droit à l'erreur est alors difficilement toléré.

Plus généralement, la gestion de réseau peut se définir comme la mise en place de moyens qui assurent le service attendu par les utilisateurs, tout en gardant une certaine souplesse permettant l'évolution du réseau selon la vision de l'entreprise. L'administrateur réseau occupe donc un poste-clé dans l'entreprise, en étant garant d'un aspect crucial au bon fonctionnement de celle-ci.

D'un point de vue plus technique, l'*International Organization for Standardization* (ISO) explicite la problématique de la gestion de réseau en cinq points, résumés par l'acronyme *FCAPS* [1]:

a. *Fault management* (gestion de fautes) : cet aspect comprend deux volets : d'une part, il s'agit ici de réagir aux problèmes quand ceux-ci se produisent : des méthodes de détection, localisation, diagnostic, isolation puis de correction doivent donc être mises en place, le tout en un temps acceptable pour le client. D'autre part, il est également nécessaire de prendre des mesures préventives (proactives), avant que des pannes n'aient lieu. L'objectif est de minimiser le temps où le réseau n'est pas fonctionnel;

b. *Configuration management* (gestion de la configuration) : elle couvre tout ce qui touche à la modification du réseau. D'abord au niveau matériel, avec l'ajout ou le retrait d'équipements, puis au niveau logiciel avec les mises à jour des logiciels et la modification des configurations réseaux. La gestion de la configuration en particulier devient d'autant plus critique que le réseau croît en complexité. Il faut toujours s'assurer que toute modification apportée répond aux besoins et ne détériore pas les services déjà offerts;

c. *Accounting management* (gestion de la comptabilité) : ce point a trait à l'utilisation des ressources par les divers clients/usagers du réseau. On s'assure notamment que chaque utilisateur reçoit bien le service pour lequel il paie, que les ressources du réseau sont bien distribuées équitablement;

d. *Performance management* (gestion de performance) : s'assurer que le réseau respecte les performances (délai, perte de paquets...) définies pour les utilisateurs. Des méthodes de mesures doivent donc être mises en place avec comme compromis d'être les plus précises possibles tout en minimisant leur impact sur justement la performance;

e. *Security management* (gestion de la sécurité) : cet aspect est vu principalement dans une optique de confidentialité. Il est crucial dans le sens où les autres points sont menacés si celui-ci est compromis. Il s'agit donc de se protéger des attaques externes (*hackers*), mais aussi de l'interne, des personnes non autorisées, tant au niveau logiciel que physique. Néanmoins, la sécurité comprend plus que la confidentialité : une donnée ne doit être ni altérée, ni modifiée lors de son acheminement (intégrité). Enfin, l'information ou le service doivent être accessibles à toute personne autorisée, à tous moments et tous lieux convenus (disponibilité).

Beaucoup de recherches ont été menées sur des technologies et des outils abordant un ou plusieurs de ces points. Par exemple, dans le monde IP, le protocole *Internet Control Message Protocol* (ICMP) [2] permet de signaler différentes erreurs, et des outils exploitant cette technologie forment le célèbre duo *ping/traceroute* [3]. Le premier permet de tester la connectivité entre deux équipements, le second détermine le chemin emprunté. Le protocole *Simple Network Management Protocol* (SNMP) [4-6] a été développé pour obtenir des informations sur les différents composants du réseau avec la possibilité de les modifier.

En matière de gestion de réseaux, l'idéal à atteindre serait une automatisation complète de tous ces aspects [7], s'affranchissant ainsi de tout risque d'erreur humaine. En pratique, un tel objectif paraît difficilement réalisable, mais il assure une bonne orientation de recherche.

La croissance du réseau Internet a conduit l'*Internet Engineering Task Force* (IETF) à proposer en 2001 une technologie résolvant certaines des limites imposées par IP, le *Multi Protocol Label Switching* (MPLS) [8]. Plus de flexibilité est notamment apportée au routage ; ce point sera explicité au chapitre suivant.

Après cette présentation de la problématique générale que présente la gestion réseau sur un plan général, considérons donc le cas du MPLS, sur lequel se concentre notre projet.

1.2 Problématique du projet

Alors que la popularité de la technologie MPLS ne cesse de croître, notamment au sein de fournisseurs de service, ceux-ci ressentent la nécessité d'une gestion efficace comme un facteur clé du succès. Des efforts se concentrent ainsi sur l'opération, l'administration et la maintenance (OAM) des réseaux MPLS [9].

Dans ce contexte, notre projet consiste à étudier puis développer un outil d'optimisation de gestion de services dans les réseaux MPLS. Cet outil doit permettre une gestion plus efficace du réseau, en présentant à l'administrateur de manière claire et structurée les informations cruciales, mais aussi en l'assistant par l'automatisation de certaines fonctionnalités.

Pour cela, ce mémoire se décompose en plusieurs parties : tout d'abord, il s'agira de faire l'état de l'art de MPLS au moyen d'une étude bibliographique approfondie, afin de bien saisir et maîtriser cette technologie, ses motivations, son fonctionnement, ses applications et ses défis. Cette phase permettra de mieux appréhender la problématique de gestion de réseau, à la lumière des connaissances acquises.

Ensuite, nous étudierons les approches, démarches et outils existants dans le domaine de la gestion des réseaux MPLS, pour relever leurs forces et faiblesses respectives ; voir notamment quels aspects ne sont pas encore traités, ou pas assez convenablement. Ceci nous permettra alors de proposer de nouvelles approches, approfondissant certaines parties de la gestion réseau, ou en abordant d'autres.

Nous pourrons dès lors établir des spécifications sur un nouvel outil et définir une architecture. Puis, pour chacune de ses fonctionnalités, nous examinerons les différentes alternatives pour choisir la plus appropriée. Nous justifierons notamment nos choix technologiques.

Par la suite, nous implémenterons cet outil et l'expérimenterons sur la plate-forme de test *Architecture et Intégration de Solutions* (AIS) de Bell Canada. Nous observerons alors concrètement son efficacité. Enfin, nous conclurons par quelques recommandations quant à l'évolution de l'outil.

Dans la section suivante, nous placerons les objectifs de ce projet dans un contexte global de gestion des réseaux avec Bell Canada.

1.3 Contexte du projet

Ce projet s'intègre dans un partenariat entre la Direction Architecture et Intégration de Solutions (AIS) de Bell Canada et l'École de Technologie Supérieure, qui contribue à l'évolution d'une plate-forme expérimentale vers une plate-forme de service intégrant un service de transport IPv4 avec qualité de service (QoS pour *Quality of Service*), et un service de réseau privé virtuel sécurisé, à l'aide des technologies IP/MPLS. Il permettra le développement d'un *middleware* entre outils de gestion et équipements de réseau, avec pour perspective la commande de ressources et la réduction de la complexité d'administration des équipements de réseau. Ainsi, dans ce projet global, d'autres technologies sont étudiées : la qualité de service, le *Virtual Private Lan Service* (VPLS), et les problématiques de sécurité.

CHAPITRE 2

ÉTAT DE L'ART ET BESOINS

« Étudie, non pour savoir plus, mais pour savoir mieux. »
Sénèque

Ce chapitre présente tout d'abord la technologie MPLS dans son contexte et ses applications. Puis, dans le cadre de notre problématique, les approches et outils déjà existants en matière de gestion de réseaux MPLS sont examinés. Ceci nous permettra d'identifier et de discuter les limitations des approches actuelles pour la gestion des réseaux MPLS.

2.1 Multi Protocol Label Switching

Depuis quelques années, MPLS est de plus en plus plébiscité par les fournisseurs d'accès. Cette technologie prometteuse retient ainsi toute l'attention des différents acteurs du monde des réseaux.

2.1.1 Terminologie

Afin de bien saisir les concepts de MPLS, nous listons ici son vocabulaire principal. Certains termes sont spécifiques à cette technologie, d'autres non. Dans tous les cas, la définition proposée ici est avant tout valide dans le cadre de MPLS.

a. **CE** : *Customer Edge* : dans un *Virtual Private Network* (VPN), il s'agit du routeur du client qui communique avec le réseau du fournisseur;

b. **Downstream** : en aval;
c. **Egress** : désigne un routeur de sortie du réseau;
d. **FEC** : *Forwarding Equivalence Class* : terme décrivant un groupe de paquets IP qui sont commutés de la même manière;
e. **FRR** : *Fast Reroute* : liés au TE, mécanismes permettant de minimiser la perte de paquets en cas de bris de LSP;
f. **Ingress** : désigne un routeur d'entrée du réseau;
g. **Label** : valeur, placée dans l'en-tête MPLS, d'après laquelle la commutation se fait. Un label est associé à une FEC;
h. **LDP** : *Label Distribution Protocol* : protocole de distribution des labels MPLS le plus utilisé;
i. **LER** : *Label Edge Router* : routeur MPLS de bordure de réseau;
j. **LSP** : *Label Switched Path* : chemin suivi par un paquet dans un réseau MPLS. Il est composé d'une succession de segments ;
k. **LSR** : *Label Switch Router* : routeur MPLS;
l. **P** : *Provider* : dans un VPN, routeur du fournisseur d'accès;
m. **PE** : *Provider Edge* : dans un VPN, routeur qui fait l'interface entre le réseau client et celui du fournisseur d'accès;
n. **PHP** : *Penultimate Hop Popping* : fait de ne pas imposer de label sur le dernier segment, afin de minimiser le temps de traitement du dernier routeur, en lui retirant l'opération de *Pop*;
o. **Pop** : action de retirer un label sur un paquet MPLS;
p. **Push** : action d'apposer un label sur un paquet MPLS (ou IP);
q. **Segment** : partie d'un LSP entre deux routeurs voisins;
r. **Swap** : action d'échanger un label sur un paquet MPLS;
s. **TE** : *Traffic Engineering* : ingénierie de trafic;
t. **Upstream** : en amont.

2.1.2 Présentation

La croissance exponentielle du réseau Internet a pris place grâce à la technologie IP. Si celle-ci s'adaptait parfaitement aux réseaux dans les années 90, la hausse inattendue du nombre d'équipements et du trafic a rapidement soulevé certains problèmes : notamment des problèmes de congestion et des tables de routage trop importantes, ralentissant ainsi les performances des routeurs. De plus, la gestion des réseaux s'est complexifiée, faisant apparaître le besoin de nouvelles technologies résolvant les limites posées par IP. Par exemple, le routage des paquets par ce protocole s'effectue uniquement en examinant l'adresse de destination, cette opération étant d'ailleurs répétée à chaque routeur. Même si le recours à des politiques de routage permet d'assouplir le routage dans IP, aucune solution ne répond vraiment à grande échelle à cette limite pour ce protocole [10].

Ainsi, un groupe de travail de l'*Internet Engineering Task Force* (IETF) regroupant des professionnels des réseaux (Cisco, Juniper) [11] a proposé en 2001 une architecture du Multi Protocol Label Switching [8].

L'idée de base de MPLS consiste à acheminer un paquet à l'aide d'un label et non plus de l'adresse de destination. MPLS effectue ainsi de la commutation de label tout en bénéficiant du routage de la couche réseau. En reprenant la classification par niveaux du modèle *Open Systems Interconnection* (OSI) [12], MPLS combine la puissance de commutation de la couche liaison aux avantages du routage de la couche réseau [10]. On parle de *Multi Protocol* car, en s'insérant entre les couches 2 et 3 (certains évoquent quelquefois la couche 2,5), MPLS est indépendant des différentes technologies, comme le montre la figure 1.

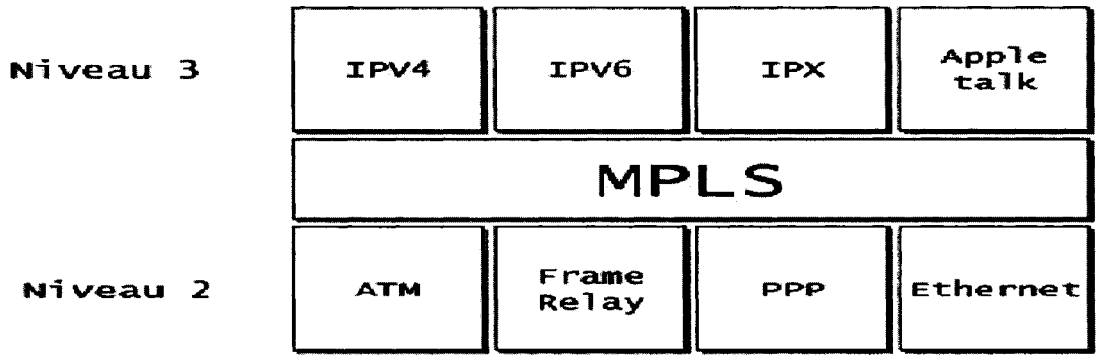

Figure 1 Caractère Multi Protocoles de MPLS

MPLS peut ainsi transporter des protocoles comme IPv4, IPv6, *Internetwork Packet Exchange* (IPX), AppleTalk sur des technologies comme Ethernet, *Asynchronous Transfer Mode* (ATM), Frame Relay ou *Point-to-Point Protocol* (PPP). Ainsi, quelle que soit la technologie utilisée au niveau 3, le niveau 2 ne voit que MPLS, et réciproquement le niveau 3 ne voit comme couche inférieure que MPLS. La gestion du cœur de réseau s'en trouve donc simplifiée, facilitant notamment l'ajout de fonctionnalités.

Plus techniquement, MPLS ajoute au datagramme un en-tête contenant un label (étiquette), lequel sera utilisé pour la commutation du paquet dans le réseau MPLS. Ainsi, contrairement à IP où l'adresse de destination est utilisée, MPLS se base strictement sur ce label. Celui-ci peut avoir plusieurs significations : il s'agit le plus souvent d'une destination, mais cela peut être une source, une application particulière, une classe de service, ou même un sous-ensemble d'une table de routage. En fait, un label sert à regrouper des paquets qui seront traités d'une manière unique. Cette notion de FEC confère à MPLS une flexibilité de routage sans précédent.

L'en-tête MPLS est constitué de 32 bits répartis de la manière illustrée à la figure 2 [13]. Il s'organise de la manière suivante :

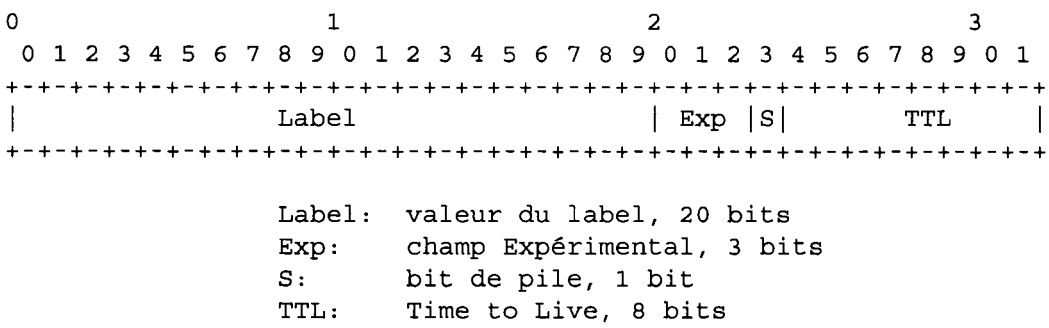

Figure 2 En-tête MPLS

- Le label en lui-même occupe ainsi 20 bits, permettant un large choix de valeurs. Seules les valeurs de 0 à 15 sont réservées.
- Le champ expérimental de 3 bits est utilisé pour transporter des informations relatives à la QoS. Huit classes de trafic peuvent ainsi être définies.
- Le bit de pile de labels sert à savoir, en cas d'empilement de labels, si l'en-tête actuellement examiné est le dernier de la pile. Le cas échéant, il est positionné à 1. Dans le cas contraire, il est mis à 0. Comme nous le verrons ultérieurement, il peut être utile d'avoir plusieurs encapsulations MPLS.
- Les bits *Time To Live* (TTL) : à l'instar d'IP, ces 8 bits servent à déterminer une durée de vie du paquet.

De par son caractère multi protocoles, l'en-tête MPLS s'adapte aux différentes technologies sous-jacentes, comme le montre la figure 3 [14].

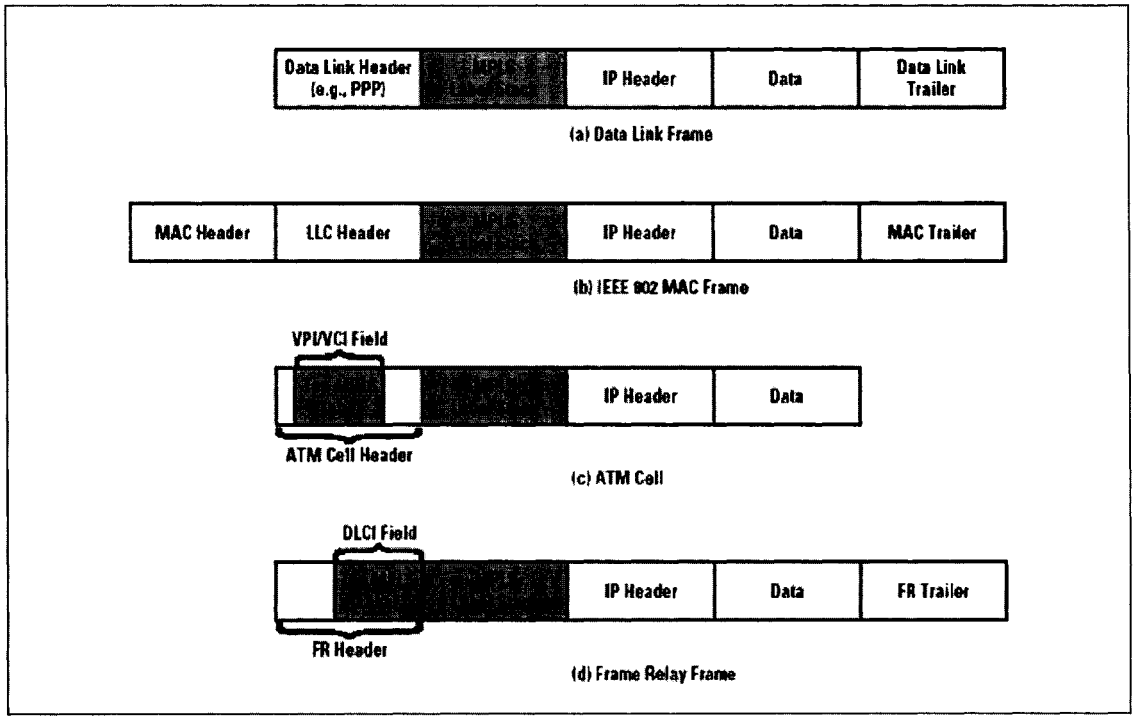

Figure 3 Implémentation de l'en-tête MPLS

Les valeurs de l'en-tête MPLS sont placées dans un nouveau format, appelé *shim* [8], lequel s'insère entre l'en-tête de la couche liaison et celui de la couche réseau. Cependant, certaines technologies comme ATM ou Frame Relay implémentent déjà le concept de commutation par étiquette. Ainsi leurs en-têtes respectifs peuvent être utilisés pour contenir des labels [8]. Pour ATM, MPLS utilise le champ *Virtual Path Identifier* (VPI) / *Virtual Channel Identifier* (VCI). Pour Frame Relay, c'est le champ *Data Link Connection Identifier* (DLCI) qui est utilisé.

La figure 4 présente un exemple de fonctionnement de MPLS.

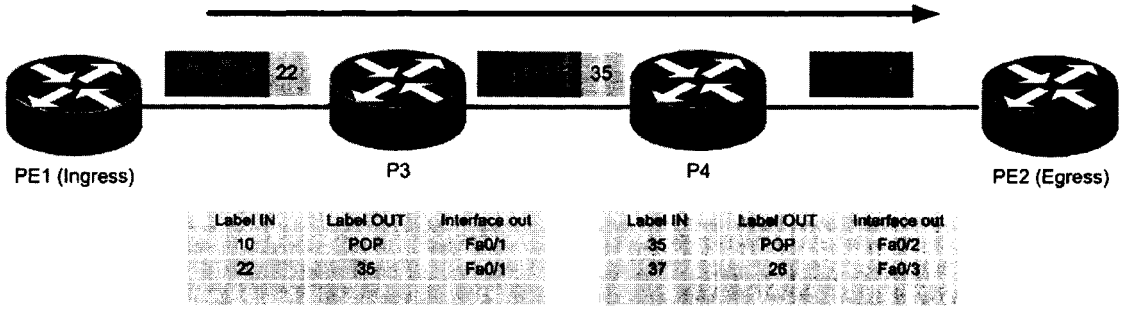

Figure 4 Exemple de fonctionnement de MPLS

- Un paquet IP arrive au routeur d'entrée PE1 (*Ingress*) du réseau MPLS. Celui-ci après avoir examiné le paquet lui associe une FEC et appose un label sur ce paquet, 22 dans notre exemple (*Push*). Il est alors envoyé vers le prochain routeur (P3).
- Le routeur P3 examine le label du paquet entrant, consulte sa table, et fait suivre le paquet vers l'interface prévue, en changeant le label, ici de 22 à 35 (*Swap*).
- Le routeur suivant examine le label du paquet entrant, et en consultant sa table, réalisant qu'il s'agit de l'avant-dernier nœud, le fait suivre mais en enlevant le label cette fois (*Pop*).
- Le dernier routeur (*Egress*) reçoit le paquet comme étant un paquet IP et le fait suivre selon le routage de ce protocole.

Plusieurs remarques s'imposent : on constate qu'un label n'a de valeur que pour un routeur donné, sa portée est donc locale. En théorie, un même label pourrait avoir différentes significations selon l'interface d'entrée [8]. Dans un tel cas, la table de commutation MPLS se baserait à la fois sur le label et sur l'interface d'entrée pour déterminer l'interface de sortie et le label à imposer. Dans la pratique des réseaux actuels, un label a souvent la même signification quelle que soit son interface d'entrée. Cela peut s'avérer utile dans certains cas, comme pour le *Fast Reroute* [15] où, en cas de panne, une nouvelle route doit être trouvée.

Pour calculer les chemins des LSP, MPLS a recours aux mêmes protocoles de routage que IP. L'établissement des labels se fait grâce à un protocole de signalisation. Il en existe plusieurs, mais le plus utilisé reste le *Label Distribution Protocol* (LDP) [16].

2.1.3 Applications

MPLS devait initialement résoudre un problème de performance [17]. En effet, en utilisant un label pour la commutation, il devait permettre une commutation plus rapide que pour IP, où il faut examiner l'adresse de destination. Cependant, les progrès en électronique ont rendu cette différence négligeable. Néanmoins, MPLS offre de nouvelles applications, principalement deux qui expliquent son succès actuellement. Il s'agit des *Virtual Private Networks* (VPN) et du *Traffic Engineering* (TE). Nous les présentons brièvement ici, sans rentrer dans les détails du fait de la complexité de ces technologies.

2.1.3.1 Virtual Private Networks

Le principe d'un VPN est d'établir la connectivité entre différents sites d'un client en utilisant une infrastructure publique, tout en fournissant un contrôle d'accès et des politiques de sécurité semblables à un réseau privé [10]. Ce concept n'est donc en lui-même aucunement lié à MPLS et était déjà mis en place avant son apparition. Ainsi, afin de mieux cerner l'apport de MPLS dans le cadre de VPN, considérons d'abord son déploiement sans MPLS. Deux implémentations de VPN se sont très répandues : le modèle *Overlay* et le modèle *Peer-to-Peer* [10].

Le modèle *Overlay* est illustré à la figure 5 [18]. Ici, le fournisseur établit des lignes émulées entre les différents sites du VPN [10]. Le client peut alors établir des communications entre ses routeurs via ces circuits. L'infrastructure du fournisseur d'accès est ainsi transparente au client, alors que le réseau interne de ce dernier reste

caché au fournisseur. Typiquement, des technologies de niveau 2 sont mises en place pour le modèle *Overlay*, comme ATM avec ses *Permanent Virtual Connection* (PVC) ou *Switched Virtual Connection* (SVC) [10]. Des technologies comme *Generic Routing Encapsulation* (GRE) [19] et *IPSec* [20] permettent aussi le déploiement du modèle *Overlay* au niveau 3.

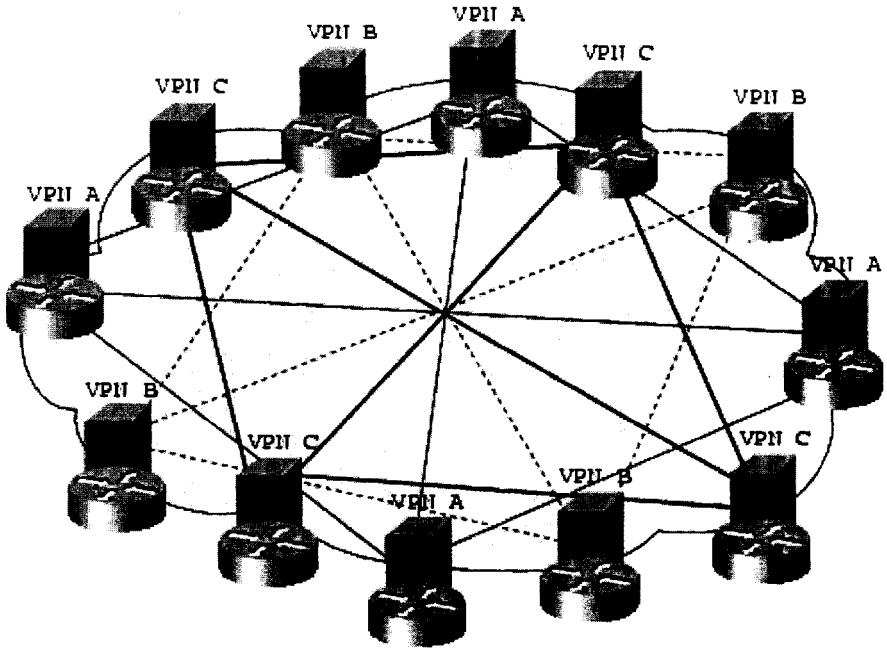

Figure 5 VPN Overlay

Ce modèle, bien que très utilisé, présente plusieurs inconvénients majeurs. Simple dans son concept, il devient très complexe à déployer à grande échelle quand le client possède de nombreux sites. En effet, s'il y a N sites qui veulent communiquer entre eux, il faut configurer au moins $N(N-1)$ circuits. L'ajout d'un $N+1^e$ site nécessite donc le provisionnement de N nouveaux circuits. De plus, il faut savoir dimensionner chaque circuit avant que le trafic n'y circule.

Le modèle *Peer-to-Peer* [10] a été mis au point pour pallier aux faiblesses du modèle *Overlay*. Il est présenté à la figure 6 [18].

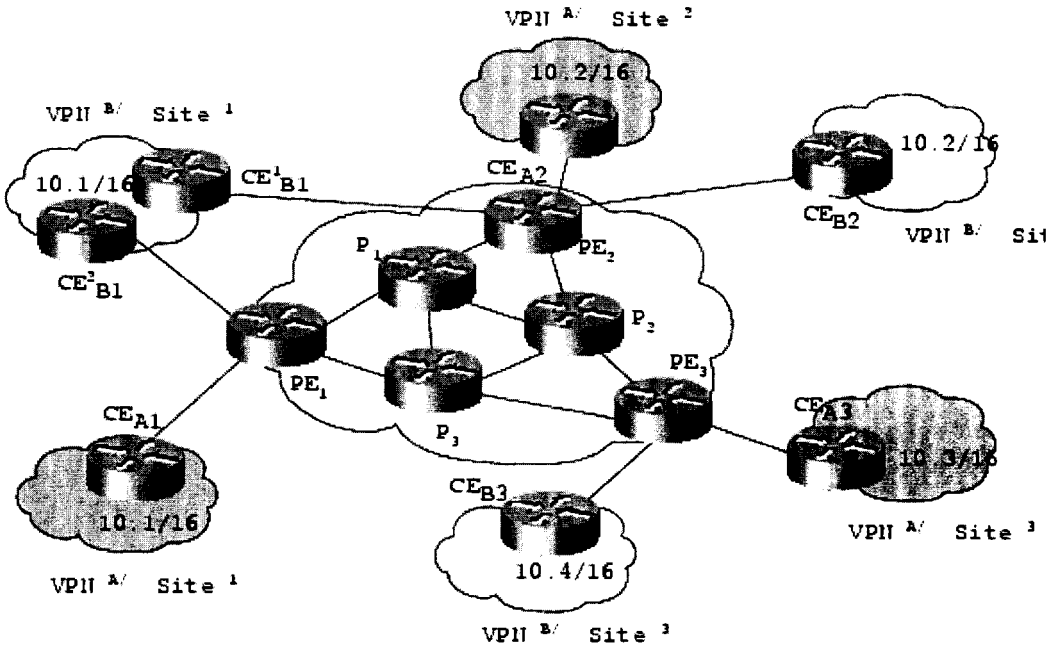

Figure 6 VPN Peer-to-Peer

Ce modèle simplifie le travail du client, mais complexifie celui du fournisseur d'accès. Chaque routeur d'un client (*CE*), communique directement avec un seul routeur de bordure du réseau du fournisseur (*PE*). Les CE communiquent leurs routes aux PE lesquels se les échangent entre eux. La majeure partie de la configuration du VPN repose sur les PE. Il n'est donc plus nécessaire de fournir pour chacun des *N* sites des circuits vers les *N-1* autres sites. L'ajout d'un site est simplifié. Le déploiement à grande échelle est ainsi facilité comparativement au modèle *Overlay*.

En dépit de la souplesse et de la simplicité apportées par le modèle *Peer-to-Peer*, celui-ci souffre d'un manque d'isolation entre les clients. En effet, les différents CE rattachés au même PE partagent le même espace d'adressage IP, ce qui pose le problème du chevauchement d'adresses.

De même que MPLS permet d'associer les avantages de la commutation niveau 2 à la flexibilité du routage niveau 3, il est possible de l'utiliser pour développer une nouvelle

approche de VPN, qui combine les avantages du modèle *Overlay* (isolement entre les clients) à la flexibilité de routage du modèle *Peer-to-Peer*. On parle de la technologie MPLS/VPN [21]. Le principe de ce modèle est illustré à la figure 7 [18].

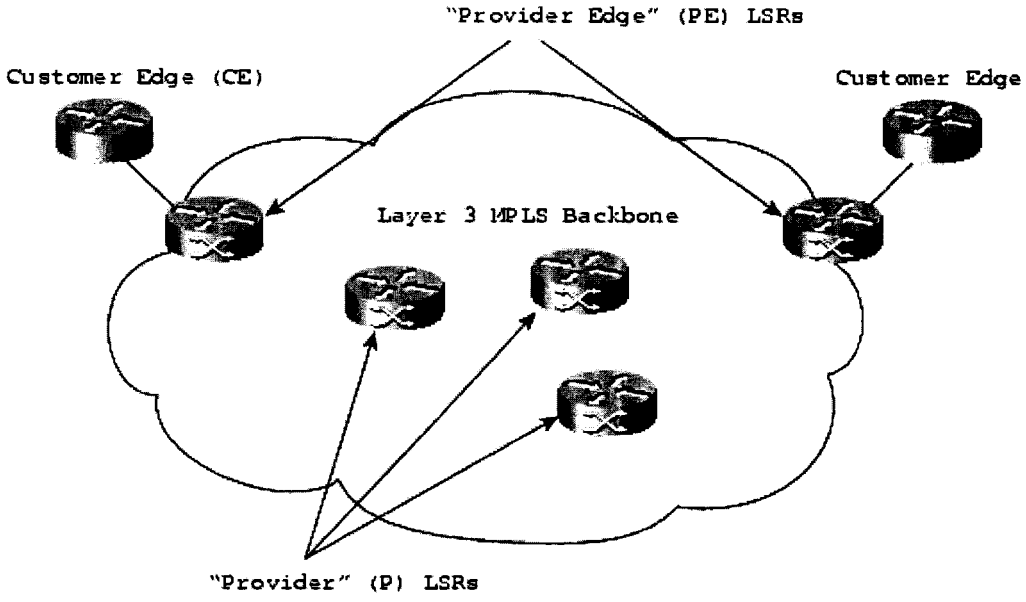

Figure 7 MPLS VPN

Ce modèle s'apparente au modèle *Peer-to-Peer* en ce que les CE ne communiquent qu'avec un PE. En revanche, l'isolation entre les différents clients est réalisée en définissant dans la table de routage du PE des sous-ensembles par client : on parle de *VPN routing and forwarding instance* (VRF). Pour distinguer les différents clients, des labels différents sont utilisés. Ceci permet d'utiliser les mêmes adresses pour plusieurs VPN. Ainsi, les paquets dans les VPN reçoivent deux labels MPLS : un pour identifier leur VPN, l'autre pour le LSP qu'ils empruntent dans le cœur de réseau.

2.1.3.2 Traffic Engineering

Une autre grande application de MPLS est le *traffic engineering*, ou ingénierie de trafic. Pour bien comprendre son principe, on a coutume de présenter le fameux « problème du poisson », illustré par la topologie de la figure 8.

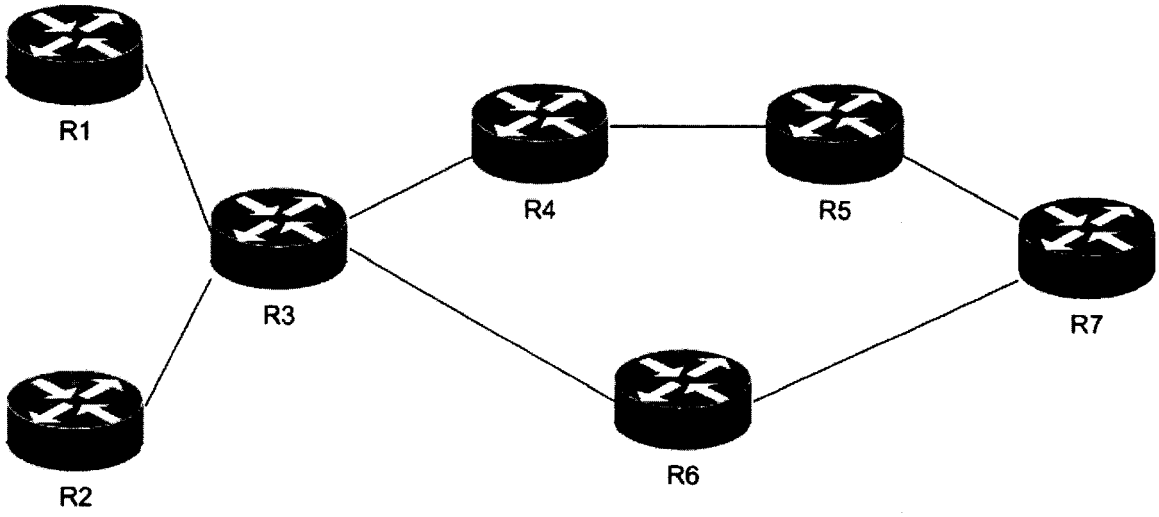

Figure 8 Problème du poisson

Supposons que R1 et R2 veuillent tous deux envoyer du trafic à R7. L'*Interior Gateway Protocol* (IGP) choisissant le chemin le moins coûteux, tout le trafic sera dirigé vers le routeur R6.

Ainsi, si les liens sont de 100Mbps et que R1 et R2 envoient respectivement 60Mbps et 80Mbps, une perte de 40Mbps aura lieu entre R3 et R6, alors que R4 et R5 demeurent inutilisés.

Pour remédier à ce problème, plusieurs options sont envisageables [15] :

a. Modifier les métriques pour faire passer le trafic par R4 et R5 plutôt que R6 : mais on ne fait là que déplacer le problème;

b. Réajuster les métriques pour effectuer du *load-sharing* entre les routes R3-R6-R7 et R3-R4-R5-R6-R7 : si cette solution apparaît satisfaisante pour ce cas-ci, elle n'en reste pas moins difficile à mettre en place pour des topologies de réseau plus larges, et souffre donc d'un sérieux problème de mise à l'échelle;
c. Utiliser plusieurs chemins définis selon plusieurs critères, incluant la métrique, mais aussi la bande passante nécessaire, afin d'optimiser l'exploitation des ressources réseaux : c'est là que MPLS TE entre en jeu.

La notion de TE n'est nullement restreinte à la technologie MPLS. Néanmoins, dans ce contexte, elle peut être puissamment déployée, en créant des LSP particuliers : on parle de tunnels. Il est alors possible de créer des tunnels garantissant une certaine bande passante, dont le chemin est calculé avec l'algorithme du *Constraint Shortest Path First* (CSPF) [15]. De cette façon, si un lien ne présente plus assez de bande passante disponible pour un trafic particulier, le *Label Switched Path* (LSP) s'établira en prenant une autre voie, peut-être moins directe, mais satisfaisant les conditions requises : des liens sous-utilisés seront ainsi rentabilisés.

En plus de seulement garantir de la bande passante, les tunnels MPLS TE offrent d'autres fonctionnalités offrant plus de flexibilité dans leurs déploiements. Par exemple, à un tunnel est affectée la notion de priorité, permettant à des tunnels jugés plus prioritaires de s'établir, même si cela doit se faire au détriment d'autres tunnels moins importants. En fait, deux priorités sont définies : la priorité d'établissement, prise en compte lorsque le tunnel cherche un chemin, et la priorité de maintien, utilisée une fois que le tunnel est mis en place. Une autre propriété de MPLS TE est l'affinité, laquelle consiste à marquer les liens et les tunnels, pour privilégier ou restreindre un chemin. Par exemple, si l'on souhaite établir un tunnel entre Montréal et Vancouver sans vouloir passer par les États-Unis, on marque alors les liens transfrontaliers d'une certaine valeur et le tunnel d'une autre. Ainsi, le tunnel restera au Canada.

MPLS TE permet également la mise en place de mécanismes de tunnels de secours pour minimiser la perte de paquets en cas de bris de LSP : on parle de *Fast Reroute* (FRR).

2.1.4 Le contexte de MPLS : le paradigme des réseaux convergents

MPLS en tant que tel ne se limite pas à l'apport d'une technologie supplémentaire parmi tant d'autres, mais s'affiche comme un élément clé d'une mutation majeure des réseaux. En effet, l'observation de l'évolution du monde des télécommunications montre depuis quelques années une tendance très forte vers un nouveau paradigme de réseau. Ce profond changement s'est opéré du fait de plusieurs facteurs [22] :

a. La dérégulation des marchés, qui a permis une concurrence plus féroce entre les différents acteurs, poussant chacun à innover tout en réalisant des économies d'échelle pour rester compétitif;
b. L'apparition de nouveaux services proposés au consommateur : l'échange de voix seule ne suffit plus, de nombreuses fonctionnalités multimédia sont déployées, notamment l'accès à Internet à haut débit, quel que soit l'endroit où se trouve l'usager;
c. Les évolutions technologiques des réseaux de données de ces dernières décennies, notamment la prépondérance, pour ne pas dire hégémonie, de la technologie IP dans le monde, le développement de nouvelles technologies comme MPLS, lesquelles permettent de nouvelles fonctionnalités et l'évolution des réseaux de transport vers du très haut débit (commutation optique, multiplexage de longueur d'ondes, en anglais *Wavelength Dense Multiplexing* (WDM)).

Ces différents éléments ont clairement fait ressortir à la fois le besoin et la possibilité technique d'une évolution des réseaux de télécommunications traditionnels, vers un réseau plus rapide, plus riche en fonctionnalités tout en étant plus économique. Les nouvelles technologies ont notamment permis d'acheminer sur IP des informations avec

des contraintes de QoS fortes, chose impossible avec IP seulement. Ainsi, les réseaux de voix et de données, jusqu'alors clairement distincts, ont convergé pour ne former qu'un seul et même réseau : on parle de réseau convergent. Toute l'information, voix ou données, transite sous forme de paquets sur des réseaux de données, majoritairement IP. La figure 9 [22] illustre la structure actuelle des réseaux métropolitains (*Metropolitan Area Network* (MAN)), régionaux (*Regional Area Network* (RAN)) et étendus (*Wide Area Network* (WAN)).

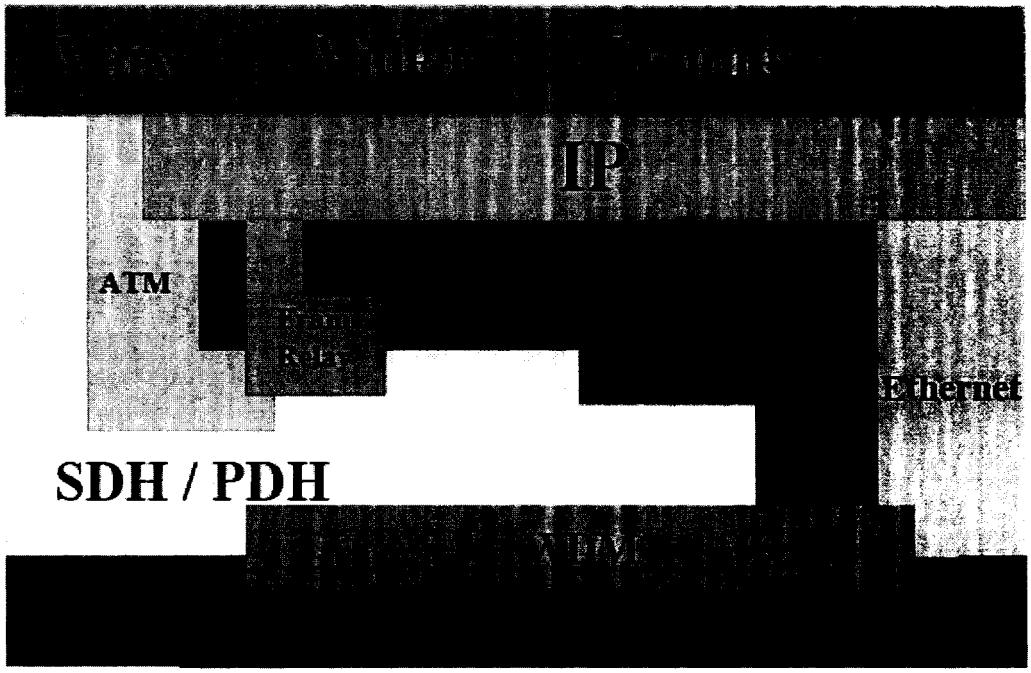

Figure 9 Structure en couches actuelle des MAN, RAN et WAN

Plusieurs acronymes doivent être explicités : SDH/PDH, respectivement *Synchronous Digital Hierarchy/Plesiochronous Digital Hierarchy*, désignent des technologies de la couche physique utilisant du multiplexage temporel (*Time Division Multiplexing* ou TDM). PoS (*Packet over SDH*) évoque la commutation IP sur TDM. Enfin G-MPLS signifie *Generalized MPLS* [23], l'extension de MPLS qui permet non seulement la

commutation par paquets, mais aussi la commutation temporelle, spectrale (par longueur d'onde) et spatiale.

Malgré la prédominance d'IP, les réseaux actuels demeurent hétérogènes, les technologies plus anciennes coexistant avec les nouvelles technologies. Or dans un souci de simplification de gestion de réseau, les réseaux de nouvelle génération (NGN pour *Next Generation Network*) présentent une structure plus uniforme, comme illustré sur la figure 10 [22].

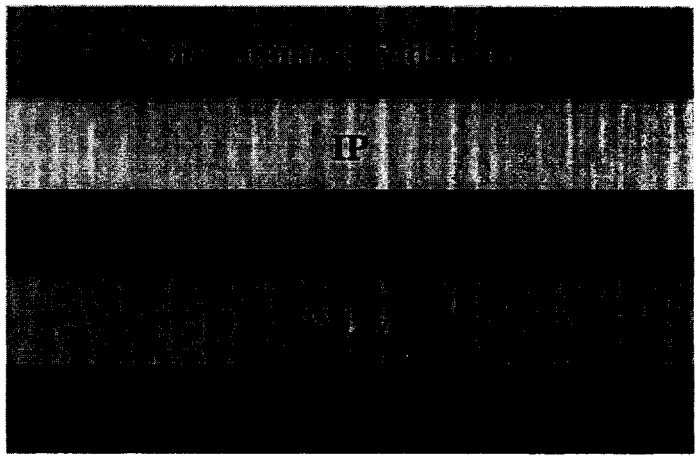

Figure 10 Structure d'un NGN fondé sur GMPLS/WDM

Le propos ici n'est pas d'expliciter chacune des technologies évoquées (leur complexité mériterait une présentation qui dépasserait le cadre de notre mémoire) mais bien de montrer que MPLS (et à terme G-MPLS) est une technologie incontournable dans la mutation des réseaux qui est en train de s'opérer. Pour les opérateurs ne l'ayant pas encore déployée, la question n'est pas de savoir s'il faut ou non le faire, mais quand. Le groupe d'experts GARTNER prévoit une hausse de NGN de plus de 60% ces 5 prochaines années [24].

MPLS est donc une technologie riche et puissante, devenue essentielle ces dernières années. Néanmoins, comme nous l'avons expliqué dans le chapitre précédent, la

croissance en complexité des réseaux requiert une gestion toujours plus rapide, plus économique et plus efficace, et MPLS ne fait pas exception. Il est donc crucial de fournir aux administrateurs un moyen de superviser le réseau MPLS, visualiser les LSP, dimensionner les tunnels TE et détecter les pertes de paquets ou même les congestions. L'utilisation d'un terminal où passer des commandes pour obtenir des informations ne suffit plus ou devient trop coûteuse en temps quand le nombre de LSP devient important. Ainsi, après avoir présenté cette technologie et ses applications principales, examinons dans le cadre de notre problématique les approches et outils existants.

2.2 Approches et outils de gestion MPLS

Quelle que soit la technologie réseau employée, la problématique de gestion reste la même. Les acteurs du développement de MPLS, bien conscients de cette réalité, ont ainsi établi un certain nombre de standards et recommandations quant à une gestion efficace des réseaux MPLS, sous forme de *Request For Comments* (RFC). De plus, diverses équipes de recherche ont mené et mènent toujours différentes approches expérimentales, implémentant des méthodes théoriques d'optimisation de gestion des réseaux MPLS tout en innovant. Enfin, le monde industriel n'est pas en reste avec le développement de solutions professionnelles de gestion des réseaux MPLS. Ce sont ces trois volets que nous analysons ici.

2.2.1 RFC

La démocratisation de MPLS s'est accompagnée de la publication d'un certain nombre de RFC pour gérer efficacement cette technologie.

Ainsi, un *framework* pour la gestion de MPLS a été établi, *A Framework for Multi-Protocol Label Switching (MPLS) Operations and Management (OAM)* [25]. Cette RFC (4378) reprend les cinq points de la gestion réseau (FCAPS) tout en explicitant

chacun dans le cadre de MPLS. Il permet ainsi une vue globale de la problématique abordée ici. Néanmoins ce document reste général : il tire les grandes lignes des défis à relever pour la gestion des réseaux MPLS, sans vraiment proposer de solutions.

La RFC 4377 [9], *OAM Requirements for MPLS Networks* présente également les exigences en OAM de réseaux MPLS, collectées auprès de professionnels des réseaux ayant déployé et testé MPLS, notamment les fournisseurs de service. Cette RFC permet donc d'être sensibilisé aux besoins de gestion des réseaux MPLS, tout en tirant profit de l'expérience du monde industriel. Cependant, là encore, aucun élément de résolution n'est apporté, la RFC soulevant seulement les difficultés de l'OAM.

Outre ces recommandations d'ordre général, plusieurs RFC abordent la problématique de la gestion de réseaux MPLS d'une manière plus spécifique. Ainsi, la RFC 4379 *Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures* [26], aborde la problématique de détection et de localisation de défaillances dans un LSP. L'idée générale est de reprendre le paradigme *ping/traceroute* cher au monde IP, en l'adaptant à MPLS. Ce document précise ainsi le format des paquets *ping* et *traceroute* MPLS tout en détaillant leur mode de fonctionnement.

Par ailleurs, plusieurs recommandations abordent l'utilisation de SNMP pour MPLS, SNMP demeurant le protocole de gestion de réseaux le plus utilisé. Ainsi, la RFC 4221 *Multiprotocol Label Switching (MPLS) Management Overview* [27] présente les différentes *Management Information Bases* (MIB) proposées pour MPLS. Plutôt que d'avoir une seule MIB pour MPLS, une division a été opérée, permettant de regrouper les objets importants selon les différents aspects de MPLS, les divers éléments de son architecture ou ses applications. Par exemple, une MIB est dédiée pour les informations relatives à LDP, une autre pour le TE, une troisième pour le VPN. Les MIB MPLS sont détaillées dans des RFC particulières [28-31].

Ces différentes recommandations montrent bien l'importance accordée à la problématique de gestion des réseaux MPLS. En outre, leurs dates de parution témoignent de la jeunesse de ce domaine. En effet, bien qu'existant depuis quelque temps au stade de *drafts*, les RFC n'ont été publiées en tant que telles que très récemment : 2004 et 2005 pour les RFC relatives aux MIB, 2006 pour les recommandations précédentes.

2.2.2 Approches expérimentales

Différentes équipes de recherches universitaires ou industrielles se sont penchées sur la gestion des réseaux MPLS. Ces travaux peuvent se diviser en deux catégories : d'une part, ceux qui, basés sur l'expérience des auteurs et la synthèse de diverses références, émettent des besoins en matière de gestions des réseaux tout en proposant la ou les meilleure(s) solution(s) à adopter. D'autre part, les travaux qui démontrent une implémentation pour résoudre un problème donné, résultats à l'appui.

Dans la première catégorie, les auteurs dans [32] présentent une vue globale de la gestion de réseaux MPLS, à l'instar de la RFC4377 évoquée précédemment. Cependant, en plus des RFC de l'IETF, cet article se base sur des recommandations de l'ITU-T.

AT&T partage ses expérimentations de MPLS dans [7]. Le principal thème abordé ici est l'objectif d'une gestion totalement automatisée en préconisant l'utilisation des MIB, mis en relief par l'expérience de l'entreprise.

D'autres travaux se consacrent à un problème précis, comme celui de la détection de bris de LSP, qui rentre dans la gestion de fautes. C'est le cas de [33] qui traite des dysfonctionnements au niveau de la signalisation de LSP par LDP. Différentes solutions et standards pour y remédier sont analysés.

Ces différents recherches, bien qu'intéressantes dans leur contenu, ne présentent aucun résultat, fruit d'une implémentation de méthodes ou techniques évoquées, sauf celui de [7] mais qui n'offre que des statistiques, telles le trafic transitant et l'utilisation des processeurs des routeurs PE. Examinons à présent les travaux réalisés en matière d'outils dédiés à MPLS.

Une équipe de recherche universitaire, conjointement avec la *National Aeronautics and Space Administration* (NASA), a mis au point un outil de gestion MPLS baptisé TEAM, comme acronyme de *Traffic Engineering Automated Manager* [34, 35]. L'objectif de TEAM est d'automatiser la gestion d'un réseau DiffServ-MPLS en configurant dynamiquement des LSP pour fournir au client la QoS exigée. Cet outil complexe permet donc d'adapter la configuration automatiquement selon l'évolution du trafic dans le réseau. Il s'attaque ainsi principalement à l'aspect gestion de configuration parmi les cinq points de FCAPS. Des algorithmes complexes de dimensionnement basés sur de la prédiction de trafic sont implémentés [36]. TEAM fournit donc un outil automatisé puissant pour gérer la QoS dans un réseau MPLS, grâce au TE.

D'autres outils de gestion MPLS ont également été développés, mais ils couvrent le plus souvent un aspect précis, souvent lié à la QoS. Ainsi, *Routing and Traffic Engineering Server* (RATES) [37] est un outil mis au point pour MPLS TE. Il permet la mise en place dynamique de LSP, mais seulement quand la bande passante est garantie. Enfin, *MPLS Adaptive Traffic Engineering* (MATE) [38] est une autre initiative d'outil pour MPLS TE, permettant de répartir la charge de trafic entre plusieurs LSP à même destination.

Les différents outils proposés dans le monde de la recherche abordent donc la gestion de configuration, en automatisant au maximum le dimensionnement des LSP pour satisfaire la QoS et éviter la congestion. TEAM a été l'outil le plus riche en fonctionnalités.

2.2.3 Outils industriels

Le monde industriel, bien conscient des réalités de la gestion des réseaux, propose lui aussi plusieurs outils pour assister l'administrateur réseau.

Un des outils les plus connus est *What's Up* d'Ipswitch [39], lequel fournit une solution de gestion d'un réseau IP qui se veut la plus complète possible. Il permet notamment de découvrir les différents équipements dans un réseau donné, de visualiser rapidement l'état du réseau, et d'être alerté en temps réel de pannes, par notification graphique, courriel, voire *Short Message Service* (SMS). Une capture d'écran de *What's Up* est présentée à la figure 11 [39].

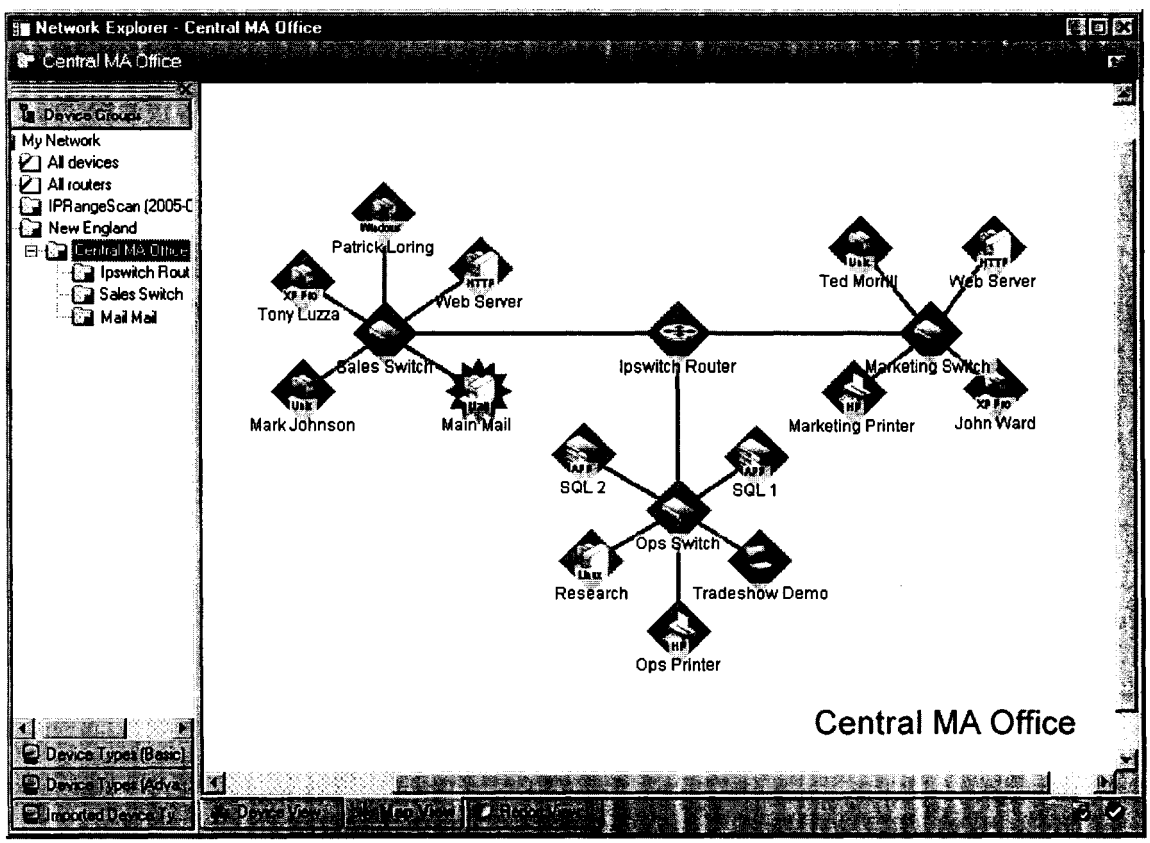

Figure 11 Aperçu de What's up

Il est cependant dédié au réseau IP et ne prend pas en charge la technologie MPLS. Nous avons néanmoins jugé souhaitable de mentionner cet outil, car même s'il ne s'applique pas dans notre problématique, sa richesse en fonctionnalités en fait une référence dont on peut s'inspirer.

Un autre outil, ou plutôt une autre famille d'outils reconnue dans la gestion de réseaux est la série *HP OpenView* [40]. Il s'agit en fait d'une gamme de logiciels dédiée à la gestion des services informatiques. L'architecture d'*HP OpenView* est modulaire. En particulier, le module *Infrastructure management* prend en charge la gestion des infrastructures. Au sein de ce dernier, il existe un composant dédié à MPLS, *HP OpenView Network Services Management Solution for MPLS networks* [41] . MPLS est ici principalement perçu pour son utilisation en tant que VPN. Ainsi, cette solution permet notamment de mesurer le trafic circulant dans les VPN MPLS, s'assurant que la QoS définie est bien respectée. La figure 12 [40] en présente un aperçu. Néanmoins, aucune mention n'est faite d'autres aspects de MPLS, comme le TE. De plus, de par ses nombreuses caractéristiques autres que MPLS, cette famille d'outils reste onéreuse et lourde à déployer, comme tout outil très complexe.

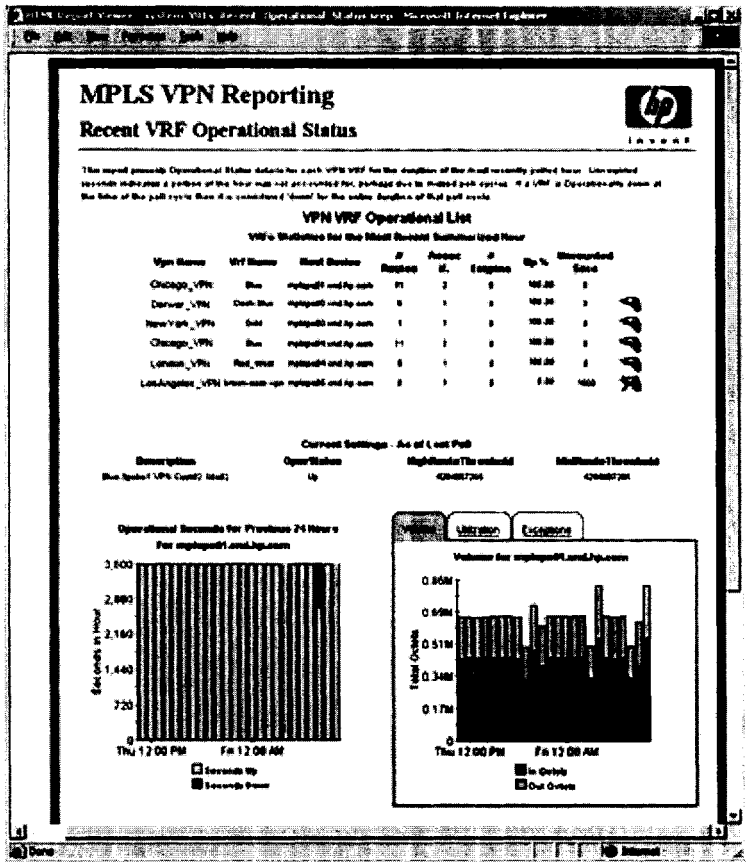

Figure 12 Aperçu du module MPLS de HP Openview

Une troisième famille d'applications de gestion réseau trouvée est *Cisco IP Solution Center* (ISC) [42]. Il s'agit d'un outil complet s'attaquant à chacun des aspects de MPLS, VPN comme TE. Son architecture est illustrée figure 13.

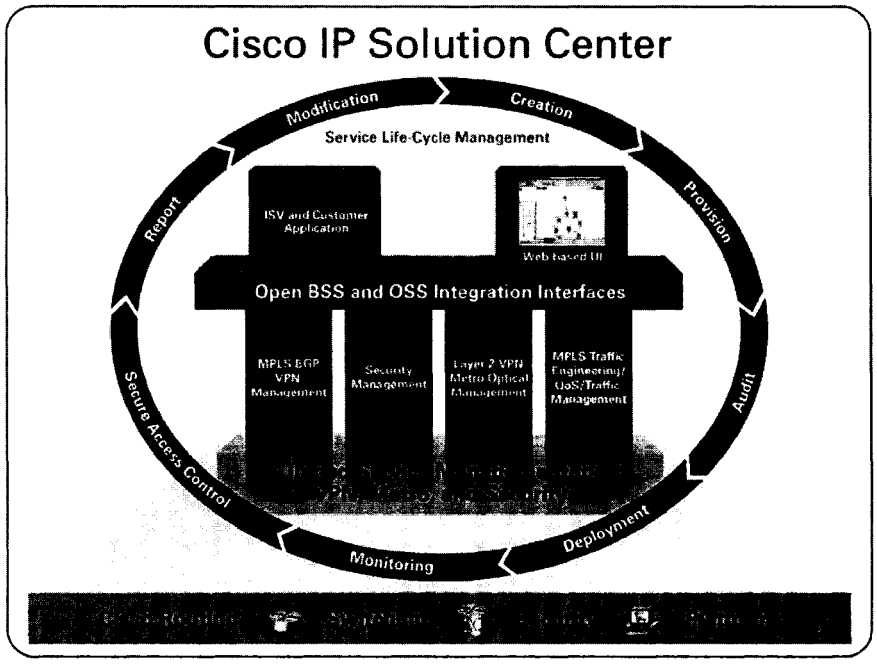

Figure 13 Architecture de Cisco IP Solution Center

On imagine aisément qu'il s'intègre bien avec l'*Integrated Operating System* (IOS) Cisco [43], du fait de la même compagnie. Cependant c'est par là même une lacune majeure : ISC ne fonctionne en effet qu'avec des équipements pourvus de l'IOS Cisco.

Établissons à présent un résumé de ce qui a été analysé.

2.2.4 Forces et faiblesses de ces outils

L'étude menée sur l'état de l'art de la gestion des réseaux MPLS fait ressortir plusieurs points. Comme nous l'avons expliqué précédemment, la problématique de gestion des réseaux est vaste et complexe, et MPLS ne fait pas exception. Ainsi plusieurs recommandations ont été établies ne serait-ce que pour lister tous les besoins en matière de réseaux MPLS. Les RFC et articles « généraux » permettent d'être sensibilisés aux

divers défis, grâce à l'expérience industrielle. Ils n'offrent aucune solution tangible, mais servent plutôt de guide, d'orientation, de référence.

Les outils et implémentations proposés par le monde universitaire répondent à des besoins précis dans la gestion des réseaux MPLS. Ils s'adressent principalement à la problématique de gestion de la configuration, avec pour souci une utilisation optimale des ressources du réseau. Ainsi, des algorithmes sont élaborés puis implémentés pour automatiser autant que possible le dimensionnement des LSP, en utilisant la flexibilité du TE. Cependant, des besoins comme la gestion des pannes ou même la visualisation des LPS ne sont pas abordés par ces outils.

Les solutions industrielles bénéficient de beaucoup plus de ressources pour leur réalisation, ce qui se perçoit assez aisément, par la richesse des fonctionnalités proposées. Eux aussi peuvent ainsi servir de références, par exemple pour la réalisation d'interfaces ergonomiques. Les différents outils professionnels rencontrés ont chacun leurs forces et faiblesses. *What's Up* permet de surveiller un réseau, d'en obtenir ses informations principales, en découvrant automatiquement les équipements, indépendamment du constructeur. Mais il reste limité au monde IP.

HP OpenView bénéficie du savoir-faire, de l'expérience d'Hewlett-Packard mais son approche de MPLS se limite à une utilisation en VPN. L'ISC de Cisco apparaît alors comme la solution la plus complète, mais elle est enfermée dans l'univers Cisco. Or bien qu'un des plus gros fournisseurs d'équipements réseaux, il n'est pas le seul, et pour diverses raisons, un réseau peut être hétérogène en termes de vendeurs. Enfin, plus une solution est riche, plus elle recueille des informations du réseau et plus grand est alors son impact sur le réseau. Or, on ne souhaite pas que le déploiement d'outils de gestion dans un réseau MPLS se fasse au détriment du trafic qui y circule.

2.3 Des besoins non satisfaits

L'étude de l'état de l'art a montré la richesse et la complexité de la gestion des réseaux MPLS. Alors que cette technologie devient incontournable, son besoin de gestion l'est également. Plusieurs problématiques de la gestion ont été abordées, notamment la gestion de configuration à travers l'automatisation dimensionnement des tunnels MPLS TE pour maximiser les ressources du réseau.

Néanmoins, aucun outil n'a été trouvé, permettant simplement de visualiser un réseau MPLS avec toutes ses caractéristiques (LSP, VPN, TE...), dans un réseau aux équipements provenant de différents constructeurs. De plus, la problématique de gestion des pannes dans MPLS n'est pas souvent abordée, ou alors seulement en théorie. La jeunesse des différents travaux et des références abordant ce sujet témoigne de la primeur de ce champ d'études.

Comme nous l'avons mentionné, la gestion des réseaux MPLS est un domaine large, alors sans avoir la prétention de fournir un outil couvrant toute cette thématique, notre outil se propose de répondre à des points non abordés jusque là, ou d'améliorer les aspects déjà étudiés.

CHAPITRE 3

QMA : UN NOUVEL OUTIL

« Les idées ne sont pas faites pour être pensées mais vécues. »
André Malraux

Après avoir présenté l'état de l'art de la gestion des réseaux MPLS, nous proposons un nouvel outil *QoS MPLS Assistant* (QMA) pour répondre à certains besoins identifiés. Comme son nom l'indique, QMA comprend deux volets : un pour la QoS, l'autre pour MPLS. Ce mémoire se concentre exclusivement sur l'aspect MPLS, la partie QoS faisant l'objet d'une autre maîtrise [44]. Ce chapitre s'organise ainsi : tout d'abord les différentes spécifications de QMA sont énumérées et expliquées. Ensuite, pour chacune des spécifications, les différentes options sont présentées et la meilleure alternative est retenue en justifiant.

3.1 Spécifications proposées

L'étape d'identification des besoins, effectuée au chapitre précédent, conduit naturellement à celle de rédaction de spécifications dans le processus de développement logiciel. Insistons là encore sur le fait que la problématique de la gestion de réseau étant vaste, nous ne pouvons, dans le cadre d'une maîtrise, proposer un outil couvrant tous ses aspects. Aussi présentons-nous ici les exigences jugées les plus critiques, auxquelles QMA doit répondre. Celles-ci peuvent être divisées en deux catégories : les exigences fonctionnelles et celles non fonctionnelles. Elles sont ici numérotées pour y faire référence plus facilement ultérieurement.

3.1.1 Exigences fonctionnelles

QMA doit offrir les fonctionnalités suivantes :

1. **Afficher la topologie du réseau** : un utilisateur doit pouvoir visualiser le réseau, avec notamment les différents équipements, la nature des liens entre eux (ATM, FastEthernet) ainsi que les adresses IP utilisées, aussi bien de *loopback* que des interfaces.

2. **Afficher les LSP** : QMA doit offrir une visualisation des LSP mis en place dans le réseau. L'utilisateur doit pouvoir lister les LSP par origine-destination, ou alors énumérer les LSP passant par un équipement précis. Pour chaque LSP, la liste des segments doit être affichée. Également, QMA doit afficher pour chaque LSP sa FEC (souvent une adresse ou une plage d'adresses IP et/ou des numéros de ports).

3. **Afficher les Tunnels** : les tunnels MPLS TE étant des LSP particuliers, une partie de cette exigence a déjà été exprimée précédemment. Cependant, QMA doit afficher les propriétés propres des tunnels, comme la description, la bande passante utilisée, les priorités d'établissement et de maintien, l'affinité et les statuts administratif et opérationnel. Il est souhaitable d'avoir des statistiques sur les tunnels, comme la quantité de trafic y transitant.

4. **Afficher les VPN** : QMA doit permettre la visualisation des VPN : pour chaque routeur les informations des VRF.

5. **Surveiller le réseau** : QMA doit fournir un moyen de détecter toute panne du réseau : perte de connexion vers un routeur, ou même bris de LSP.

6. **Prévoir le réseau** : il serait souhaitable que QMA puisse prévoir le trafic d'un lien, LSP ou tunnel donné, afin d'en connaître la valeur sans avoir à interroger trop souvent le réseau, ce qui encombrerait inutilement la bande passante. En elle-même cette fonctionnalité garde un intérêt limité, mais constitue un premier pas dans des perspectives d'évolution où l'on voudrait ajouter de la configuration dynamique de LSP ou tunnels, aspect largement abordé dans les articles donc non repris ici.

7. **Authentification** : QMA doit proposer un mécanisme d'authentification pour se protéger de toute utilisation frauduleuse

3.1.2 Exigences non fonctionnelles

En plus des exigences fonctionnelles, voici d'autres spécifications que doit respecter QMA :

8. **Plusieurs utilisateurs** : QMA doit pouvoir être utilisé par plusieurs personnes en même temps, au moins cinq, nombre estimé d'administrateurs travaillant simultanément sur le réseau.

9. **Disponibilité** : QMA doit pouvoir fonctionner 24h/24, 7 jours/7 puisqu'il s'agit du besoin de disponibilité d'un réseau.

10. **Confidentialité** : QMA doit rendre confidentielles les données échangées.

11. **Évolutivité** : QMA doit permettre des évolutions, notamment des ajouts de fonctionnalités, éventuellement son intégration dans une application plus large. Un soin particulier sera donc apporté pour fournir un produit modulaire, documenté, facilitant au maximum la réutilisation des composants. Il va sans dire que QMA doit

être conçu et développé avec un maximum de simplicité. Une telle spécification est difficilement quantifiable, mais permet de donner une direction au travail.

12. **Simplicité** : QMA se veut être simple d'utilisation. On estime donc qu'il faudra donc au plus une heure à un nouvel utilisateur pour maîtriser l'outil : il s'agit du temps maximum estimé pour qu'un néophyte puisse découvrir et maîtriser toutes les fonctionnalités.

13. **Ergonomie** : QMA est conçu pour faire gagner du temps à ses utilisateurs. Ainsi, il est conçu dans une perspective ergonomique : il ne faudra pas plus de quatre clics de souris pour atteindre une fonctionnalité du système.

14. **Indépendance du constructeur** : QMA doit pouvoir opérer quelle que soit la marque de l'équipement MPLS, pourvu que celui-ci respecte les standards établis dans les RFC. Il s'agit de la fonctionnalité innovante principale par rapport à ISC.

15. **Impact minimum sur le réseau** : QMA doit avoir un impact minimum sur le réseau, en termes de trafic généré. Là encore, il s'agit d'une spécification non quantifiée, mais qui confère un cap à suivre.

3.2 Réponses aux exigences

QMA doit proposer un certain nombre de solutions à des problèmes précis. Nous présentons ici l'approche de QMA pour répondre à ses exigences, d'un point de vue architectural tout d'abord. Rappelons, une fois de plus, que les détails de l'implémentation de QMA seront fournis dans le chapitre suivant. Pour l'heure, il convient premièrement de comprendre la structure de QMA à haut niveau. Reprenons donc les spécifications exposées pour y répondre.

3.2.1 Interaction avec le réseau (spécifications 1 à 4, 14, 15)

3.2.1.1 Position du problème

Un aspect crucial de QMA est son interaction avec le réseau. Les premières spécifications (1-4) imposent que QMA obtienne de celui-ci toutes les informations jugées essentielles, tout en minimisant l'encombrement de la bande passante (14). Notons que dans le cadre des spécifications proposées, il s'agit davantage de collecter des informations du réseau (lecture) que de les modifier (écriture). QMA n'est en effet pas prévu pour la configuration automatique de LSP, mais plutôt pour leur visualisation.

3.2.1.2 Différentes alternatives

Examinons les différentes possibilités pour interagir avec un routeur :

- *Telnet* [45] : probablement le moyen le plus utilisé. Simple mais puissant, il souffre d'une grande lacune en matière de sécurité. En effet, toutes les commandes envoyées au routeur via ce protocole passent en clair sur le réseau.

- *Secure Shell* (SSH) [46] : la "version sécurisée" de Telnet. Il permet donc de passer des commandes facilement et en toute sécurité. Néanmoins, tous les routeurs ne supportent pas ce protocole. De plus, du fait de la sécurité qu'il apporte, beaucoup de messages doivent être échangés pour établir la communication.

- SNMP [4-6] : Le protocole reconnu de gestion de réseau. La version 3 permet d'échanger des informations avec le réseau en toute sécurité, et le recours au *User Datagram Protocol* (UDP) [47] plutôt qu'au *Transmission Control Protocol* (TCP) [48] minimise l'encombrement sur le réseau. Cependant, son efficacité

reste tributaire de l'implémentation de l'agent au sein du noeud géré : on ne peut obtenir que les informations définies dans les MIB installées. De même, les *traps* ne peuvent être configurées à souhait. Ainsi, s'il fournit un moyen sécurisé de gérer le réseau, SNMP n'a néanmoins pas la même flexibilité que *Telnet*.

- *HyperText Transfer Protocol* (HTTP) [49], *HyperText Transfer Protocol Secured* (HTTPS) [50] : les routeurs ont souvent une interface Web, mais même si la sécurité est présente avec HTTPS, l'interface web propose souvent des commandes de configuration limitées, et ne permet pas d'envoyer des commandes précises sur des aspects avancés de l'IOS, comme les commandes liées à l'administration MPLS.

- *Trivial File Transfer Protocol* (TFTP) [51] : davantage utilisé pour des configurations que pour des commandes ponctuelles, même régulières.

Les outils présentés au chapitre précédent comme *What's Up* ou *TEAM* montre une préférence très nette pour SNMP, du moins quand il s'agit de récupérer des informations du réseau. Pour la configuration, TEAM utilise un client *Telnet* ou alors TFTP pour uploader un fichier de configuration. L'implémentation de serveurs SSH au sein des routeurs est relativement récente (elle n'est présente dans aucune *General Deployment* des IOS Cisco), ce qui expliquerait encore que ce protocole ne soit pas encore utilisé, malgré ses qualités. IP Solution Center de Cisco, en revanche, l'utilise déjà [42].

La première idée est donc de recourir à un protocole sécurisé, si l'on souhaite accéder au réseau de n'importe où, comme le montre la figure 14.

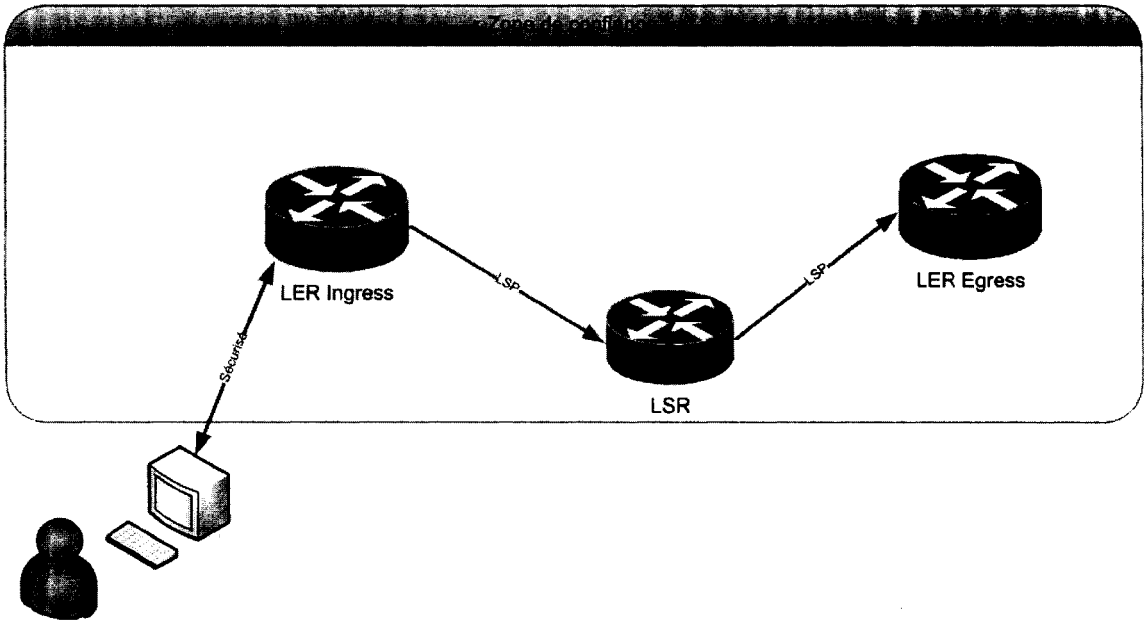

Figure 14 Architecture avec protocole sécurisé uniquement

Néanmoins, le recours à Telnet peut ne pas être gênant, si le serveur qui établit la communication avec le(s) routeur(s) se trouve dans une "zone de confiance". La figure 15 illustre un tel cas. Il faut alors utiliser un protocole sécurisé entre la machine client, et le serveur situé dans la zone de confiance.

Dans tous les cas, il est important de souligner que les différents moyens d'interagir avec un routeur ne s'excluent pas mutuellement. Certes, avec SSH, *Telnet* devient inutile, mais SSH ne remplacera jamais SNMP, ou même l'utilisation de TFTP. Au contraire, ils peuvent être complémentaires.

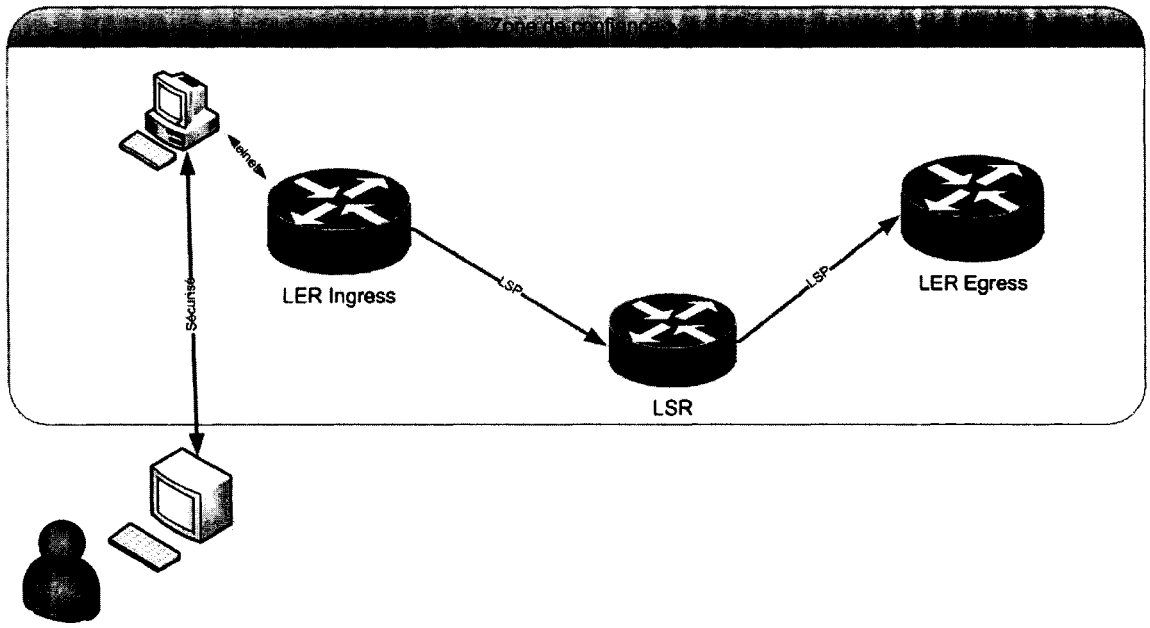

Figure 15 Exemple de recours à Telnet

3.2.1.3 Choix de SNMP

Dans le cadre de QMA, SNMP reste la meilleure option pour les raisons suivantes :

- C'est un protocole conçu pour un échange automatisé d'informations, à l'inverse de *Telnet* ou SSH. En effet, ces deux protocoles sont plutôt utilisés par un administrateur directement pour interagir avec un routeur. Au contraire, les informations telles qu'elles sont obtenues par SNMP sont certes riches, mais difficilement compréhensibles sans traitement préalable.
- Grâce à l'utilisation d'UDP, il encombre moins la bande passante que des protocoles basés sur TCP, satisfaisant ainsi la spécification 14.
- Il permet l'indépendance des constructeurs (spécification 13) dans le sens où l'implémentation des MIB est standardisée par des RFC. Si les informations étaient obtenues par Telnet par exemple, il faudrait s'adapter à chacun des logiciels sur les routeurs des différents constructeurs.

Il reste en outre le protocole préféré de l'industrie ou de la recherche pour la collecte d'informations, comme l'a montré l'étude des différents outils de gestion.

Le seul inconvénient de ce choix est qu'il nous rend dépendant de l'implémentation effective des objets dans les MIB et plus généralement des MIB elles-mêmes. En d'autres termes, étudier les MIB dans les RFC est une chose, les voir implémentées et remplies dans les routeurs en est une autre.

3.2.2 Méthode de détection des pannes (spécification 5)

Une problématique majeure que devrait traiter QMA est la détection de pannes. Celle-ci existait bien évidemment déjà avant MPLS, et quelle que soit la technologie réseau des mécanismes de détection y ont toujours été implémentés.

3.2.2.1 Position du problème : insuffisance des mécanismes existants

Comme MPLS s'insère entre la couche liaison et la couche réseau, on peut être porté à croire que les dispositifs pour ces deux niveaux sont suffisants. Or il n'en est rien.

En effet, considérons par exemple qu'un problème dans une session LDP, ou une intervention humaine maladroite, corrompe la table des labels MPLS dans un routeur. Alors le routeur contiendrait dans sa table des valeurs de labels qui ne correspondent plus à celles de la table de son voisin. Il se pourrait que le label erroné puisse être interprété par le routeur suivant comme ayant une autre signification (dans le cas d'un renouvellement de la table par exemple), auquel cas le paquet sera envoyé vers une mauvaise destination. Mais plus simplement, une corruption de labels pourrait faire qu'un routeur enverrait un label inconnu à son voisin, lequel ignorera alors purement et simplement le paquet comme spécifié dans [8]. Un tel cas est illustré figure 16.

Une panne au niveau MPLS peut ainsi conduire à un "trou noir". Or, s'il s'agit *juste* d'une corruption de labels : au niveau de la couche liaison, tout sera fonctionnel, aucune anomalie ne sera détectée. Les auteurs de [33] résument les anomalies au niveau de LDP qui peuvent provoquer des erreurs dans les labels : erreur de session, erreur de configuration, labels obsolètes, erreurs dans l'implémentation du protocole, corruption de la table des labels.

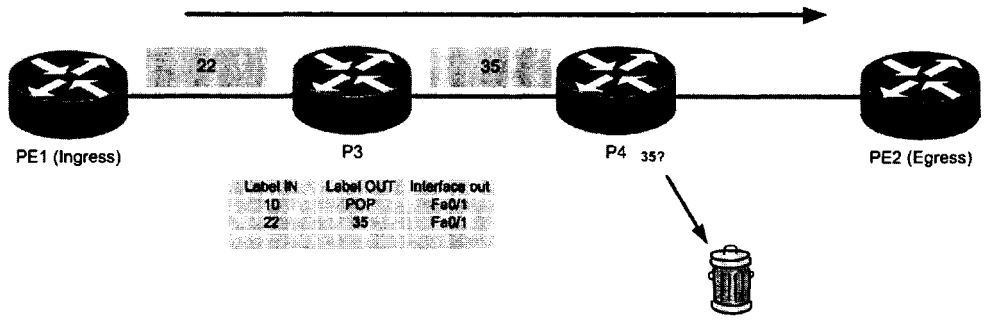

Figure 16 « Trou noir » suite à une corruption dans la table des labels

3.2.2.2 Différentes alternatives

Bien conscients des limites, dans le monde MPLS, des mécanismes de détection traditionnels, les chercheurs et ingénieurs se sont appliqués à imaginer et concevoir des mécanismes pouvant détecter et/ou diagnostiquer des pannes dans un réseau MPLS. Les deux groupes principaux ayant étudié cette question sont *l'International Telecommunication Union* (ITU) et l'IETF.

ITU-T

La recommandation Y.1711 de l'*ITU Telecommunication Standardization Sector* (ITU-T), appelée *Connectivity Verification (CV)*, consiste à envoyer périodiquement, à chaque

seconde, des paquets "tests" sur le LSP, contenant l'information du LSR *Ingress* et du LSR *Egress* [32]. Le format d'un paquet CV est présenté figure 17 [32].

Il y a un empilement de labels, le second label servant à désigner un paquet de contrôle avec un label 14 (désigné comme le label *Operation Administration and Maintenance* (OAM) [52]), le champ EXP devant désigner la priorité maximum. Le champ clé est le champ *Trail Termination Source Identifier* (TTSI) identifiant le LSR *Ingress* (LSR ID) et le LSP (LSP ID).

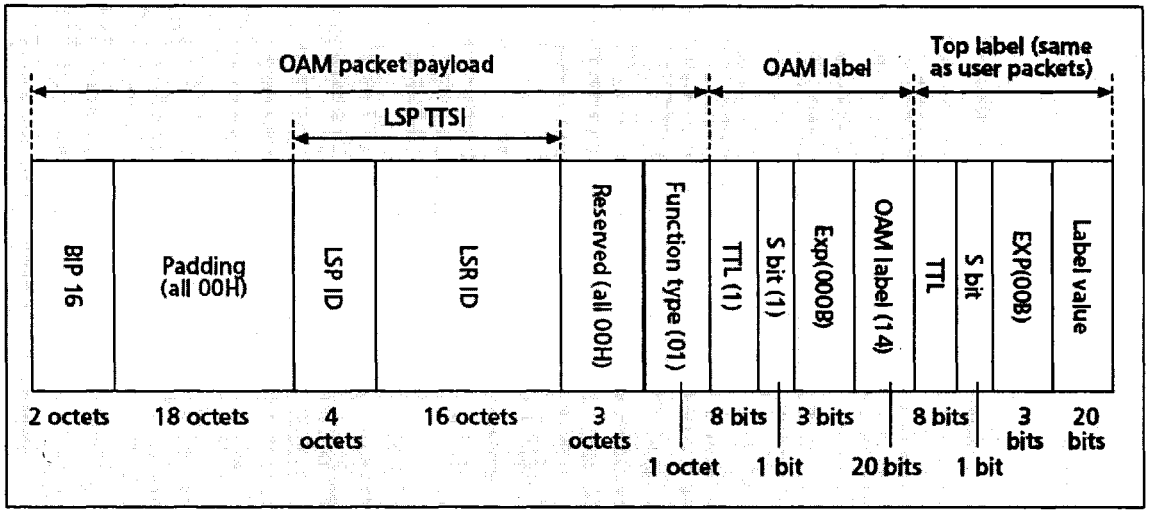

Figure 17 Format d'un paquet Connectivity Verification

Chaque seconde un paquet CV est envoyé. Le LSR *Egress* effectue la détection :

a. s'il ne reçoit aucun paquet pendant plus de 3 secondes, il signale une perte de connectivité (*dLOCV*) ;
b. s'il reçoit un paquet avec un TTSI inattendu, il signale une erreur de disparité (*dTTSI-mismatch*) ;
c. s'il reçoit à la fois des paquets avec des TTSI bons et d'autres inattendus, il signale une erreur de fusion (*dTTSI-miss-merge*) ;
d. s'il reçoit plus de cinq paquets en 3 secondes avec de bonnes valeurs de TTSI, il signale une réception excessive (*dExcess*) : un paquet doit être répété.

Pour permettre une détection plus rapide, la recommandation révisée 1711 a en outre développé le concept de *Fast Failure Detection* (FFD), permettant d'envoyer des paquets CV à intervalle plus court.

IETF

L'IETF, de son côté, a proposé une gamme plus large d'initiatives pour la détection et le diagnostic de bris de LSPs.

Ping/Traceroute MPLS

L'idée la plus célèbre est sans conteste la reprise du paradigme *ping/traceroute* du monde IP, *ping* servant surtout à détecter un problème et *traceroute* à le localiser. *Ping* MPLS [26], aussi appelé LSP *Ping*, utilise un paquet UDP encapsulé dans IPv4 ou IPv6. Le format d'un paquet Ping MPLS est illustré figure 18 [32].

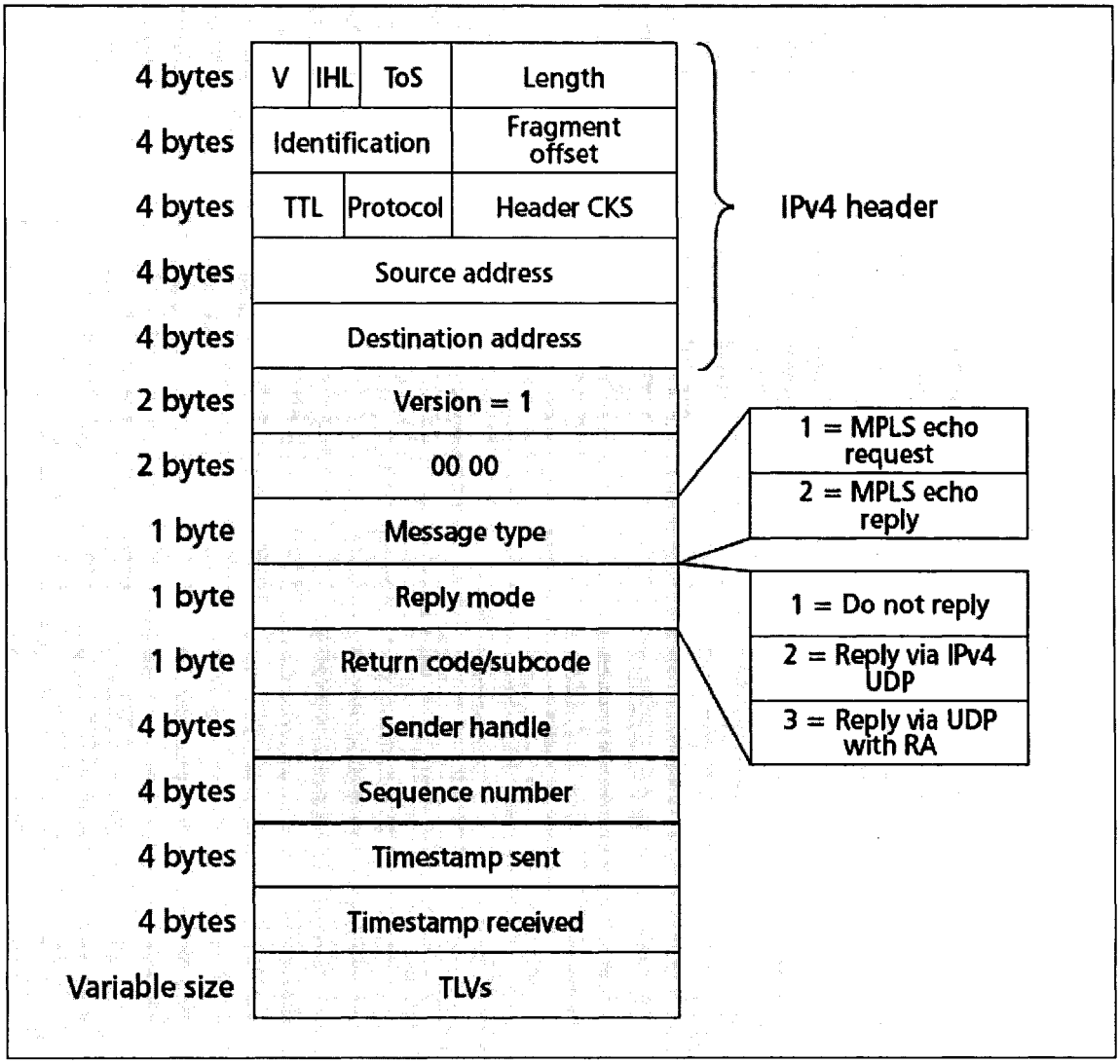

Figure 18 Format d'un paquet Ping MPLS

Ping MPLS présente typiquement deux types de messages, *echo request* et *echo reply*, mais il est plus souple que son prédécesseur IP : on peut spécifier au destinataire de ne pas répondre (avec *reply-mode* mis à 1) : l'intérêt est alors au niveau du récepteur de compter les *ping* reçus et d'enregistrer des statistiques de délai, gigue, par exemple dans une MIB à l'instar des *ping* IP, ce qui en outre traite la question du chemin de retour (si un *ping* ne reçoit pas de réponse, il est difficile de déterminer lequel des paquets *request*

ou *reply* s'est perdu). Il est également possible de demander une réponse avec l'option *Router Alert* dans l'en-tête IP, obligeant les routeurs à examiner plus attentivement le paquet [53].

Un *ping* MPLS sert à vérifier qu'un paquet envoyé sur un certain LSP arrive bien au LER *Egress* de ce LSP. Pour cela, il contient un ou plusieurs champs *Type Length Value* (TLV) servant à identifier la FEC. Par exemple, un LER associe l'adresse *172.168.18.64* au label 37, et il veut s'assurer que ce label permet bien d'acheminer le paquet vers l'adresse donnée. Alors il envoie un *ping* MPLS avec le label 37, contenant un champ TLV où le type sera "*LDP IPv4 Prefix*", avec comme valeur *172.168.18.64*. Le champ TTL de la *shim* est mis à 255, pour s'assurer que le paquet peut aller jusqu'au LER *Egress*, et le champ TTL de l'en tête IP à 1 pour ne pas sortir du nuage MPLS. Ainsi, le routeur *Egress* pourra analyser le champ TLV et vérifier qu'il est bien le routeur *Egress* pour cette adresse.

Traceroute MPLS est semblable à *traceroute* IP : il consiste à envoyer des *ping* avec des TTL croissants, afin notamment de localiser une faute. Un champ TLV spécial est conçu dans le cas de chemin multiple *Equal Cost MultiPath* (ECMP) : il s'agit du *Downstream Mapping*. Il permet à un routeur Downstream d'informer le routeur Ingress de plusieurs chemins ; dans ce cas, le routeur Ingress envoie différentes requêtes pour tester les chemins possibles. Une utilisation habituelle de *ping/traceroute* MPLS est donc d'effectuer des *ping* réguliers et de recourir au *traceroute* en cas de problème, c'est-à-dire paquets perdus ou contenant un code d'erreur indiquant un dysfonctionnement.

Cependant, à cause de ses mécanismes de contrôles, *ping* MPLS ne permet pas des détections de pannes inférieures à la seconde.

Bidirectionnal Forwarding Detection (BFD)

BFD est un principe qui n'est pas limité au monde MPLS, puisqu'il est multicouches [54]. Cependant, des travaux ont été menés pour l'adapter à MPLS [55]. Le principe est d'établir des sessions pour chaque LSP et d'envoyer régulièrement des paquets pour vérifier la connectivité. Les sessions sont démarrées par des *ping* MPLS, permettant ainsi la vérification du champ de contrôle (le bon label est utilisé pour la bonne FEC), puis maintenues par des paquets de contrôle BFD.

Ceux-ci contiennent des "discriminants" qui permettent, couplés avec l'adresse d'origine, d'identifier la session. La différence avec l'utilisation de *ping* MPLS réside dans le format des paquets BFD, qui contiennent moins d'information que les *ping* MPLS, permettant ainsi une détection plus rapide, inférieure à la seconde.
En contrepartie, seule la connectivité des paquets donnés est surveillée, il n'y a plus de vérification des champs de contrôle.

Il existe deux modes d'utilisation de BFD :

a. Mode asynchrone : lors de la session, les deux LSR s'envoient périodiquement des paquets BFD, et lorsqu'un certain nombre d'entre eux est perdu, la session est déclarée terminée;
b. Mode à la demande : une fois la session établie, les deux LSR ne s'envoient rien sauf si l'un d'entre eux souhaite vérifier la connectivité, auquel cas il envoie une séquence de paquets, acquittés par l'autre LSR.

Un mode adjoint est le mode écho, où les paquets reçus sont renvoyés à l'expéditeur.
Chacun des modes a ses avantages et inconvénients respectifs, suivant les protocoles pour lesquels BFD est utilisé. Néanmoins dans le cadre de MPLS, le mode asynchrone paraît le plus adapté : on souhaite un temps de détection court, ce qui disqualifie le mode

à la demande. Le mode écho est inapproprié à cause de la nature unidirectionnelle des LSP.

LSR Self Test

Le principe de LSR Self Test est d'effectuer une vérification locale [56]. Ainsi, LSP Self Test ne vérifie pas un LSP, seulement la connectivité entre les routeurs Test, *Upstream* et *Downstream*. Le fonctionnement de Self-Test est illustré figure 19.

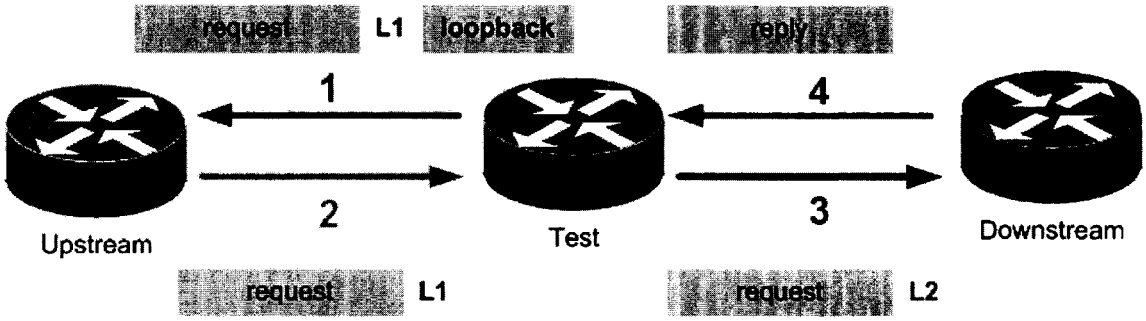

Figure 19 LSR Self-Test

Le routeur Test envoie une requête à son routeur *Upstream* (**1**) avec un TTL qui doit expirer au routeur *Downstream*. Celui-ci renvoie le paquet au routeur Test (**2**), lequel le transmet au routeur *Downstream* (**3**). Le routeur *Downstream* émet alors une réponse vers le routeur Test pour le notifier du succès du test (**4**).

Les paquets échangés sont des *ping* MPLS, mais pour minimiser le travail du routeur Downstream, deux nouveaux types de messages *ping* MPLS sont créés : *MPLS Data Plane Verification Request* et *MPLS Data Plane Verification Reply* : les *timestamps* sont notamment supprimés.

Pour minimiser la charge du routeur Upstream, LSR Self-Test introduit la notion de FEC de type Loopback. En cas de réception d'un label de FEC Loopback, le routeur ôte ce label (POP), puis renvoie le paquet via son interface d'arrivée. La FEC Loopback permet en outre de tester des labels entrants du routeur Test qui ne sont pas encore assignés par le routeur Upstream.

IETF versus ITU-T

L'étude des mécanismes proposés par l'ITU-T et l'IETF fait apparaître un contraste assez marquant : les propositions IETF sont plus nombreuses, plus accessibles, et proviennent de professionnels d'équipements de réseaux, notamment Cisco et Juniper. C'est donc sans surprise que les mécanismes implantés actuellement dans les routeurs sont principalement issus des travaux de l'IETF.

Par ailleurs, les propositions IETF sont préférables pour des raisons plus techniques :

a. la recommandation Y.1711 ITU-T nécessite de pouvoir gérer le champ TTSI, alors que *ping* MPLS utilise des TLV déjà connues (par exemple une adresse IPv4);
b. la fréquence d'envoi des paquets CV de Y.1711 est fixe, alors que celle de BFD est modifiable ainsi que l'envoi des *pings* MPLS. Mais il est vrai que FFD permet aussi une fréquence modifiable;
c. la recommandation Y.1711 permet de détecter une erreur (perte de paquet, mélange de LSP...), alors que *ping/traceroute* MPLS permet aussi de localiser l'erreur.

Ainsi, la gamme d'outils proposée par l'IETF est plus riche, et plus souple que les recommandations ITU-T en matière de détection et diagnostic de LSP brisés. De plus, les propositions sont actualisées régulièrement : il est significatif de voir que les références de *ping* MPLS, BFD, LSR Self-Test datent toutes de moins d'un an.

L'analyse des outils proposés par l'IETF suggère ces quelques points :

a. *ping/traceroute* MPLS permet de détecter et localiser les pannes mais les contrôles effectués prennent du temps;
b. BFD est rapide pour détecter une perte de connectivité, mais effectue moins de contrôle que *ping* MPLS;
c. LSR Self-Test est efficace pour la vérification d'un routeur, mais il ne permet pas de vérifier un LSP dans son ensemble. Dépendant le type de protection à apporter, ce mécanisme sera choisi ou délaissé.

Ainsi les différents outils proposés ont chacun leurs spécificités, mais une sage combinaison permettra d'en tirer le meilleur profit : on peut par exemple utiliser BFD pour une détection rapide, en cas de panne recourir à *ping* MPLS afin de vérifier le contrôle de données, puis localiser le problème avec *traceroute* MPLS.

3.2.2.3 Méthode de détection proposée

Sécurisation des communications

Il est important avant tout de proposer une architecture disponible pour le maximum de réseaux, nous supposons que le serveur chargé d'automatiser la surveillance du réseau ne se trouve pas dans une zone de confiance, et par conséquent requiert une communication sécurisée avec le routeur. SSH et SNMPv3 doivent donc être utilisés pour les échanges équipements/serveur.

Gestion à l'échelle du réseau

On suppose ici que la topologie du réseau est parfaitement connue, notamment tous les LSP.

Mise en place des traps SNMP

Une des premières choses à faire est de se "mettre à l'écoute" du réseau, en configurant le serveur pour recevoir des *traps* précises. Les *traps* sont définissables par l'agent, en particulier pour des MIB expérimentales, c'est pourquoi nous ne pouvons pas fournir ici de liste de *traps* à écouter et quelles actions prendre pour chacune, il s'agit plutôt d'une recommandation générale.

Idéalement, il faudrait même configurer des *informs* plutôt que des *traps*, les *informs* étant acquittées. Néanmoins, les *traps* restent encore plus fréquentes.

Hiérarchie des LSP

Certains LSP peuvent être plus importants que d'autres. En fait, de nombreux LSP existants ne sont pas utilisés pour transiter les données. C'est le cas des LSP entre routeurs de cœur de réseau. Ainsi, il faut définir quels sont les LSP à vérifier. Ceci peut être accompli de plusieurs manières :

a. Lister explicitement les LSP à vérifier (ou à ne pas vérifier) : pratique, mais seulement adapté pour des petits réseaux;
b. Considérer les LSP avec une ou plusieurs propriété(s) précise(s) : par exemple, une origine avec un LER défini, un nombre minimum de sauts, etc. ...

Vérification des LSP

Une fois la liste des LSP à vérifier établie, il faut les contrôler un à un. Ceci permet de contrôler chacun des LSP, mais à grande échelle, une telle pratique s'avère coûteuse, en termes de trafic notamment. Cependant une telle procédure est nécessaire pour garantir un temps de détection de pannes maximum. En revanche, certains LSP peuvent ne pas nécessiter de détection rapide, auquel cas ils peuvent être contrôlés ponctuellement, sur

la base d'une fréquence déterminée par l'administrateur. En cas de très grand nombre de LSP, on peut imaginer une vérification statistique où un certain nombre de routeurs, là encore paramétrable, sont aléatoirement sélectionnés et contrôlés, le tout à une fréquence donnée. Néanmoins, une telle approche ne doit pas être utilisée pour des LSP dont on veut garantir une détection rapide. Elle peut s'avérer utile sur des LSP de secours en revanche.

Détection et diagnostic d'un LSP

Pour minimiser le temps de détection et l'impact de la vérification, BFD, couplé à *ping* MPLS, offre le meilleur compromis ; BFD pour sa légèreté et *ping* MPLS pour sa vérification du champ de contrôle, comme discuté précédemment. Si une erreur est détectée au niveau de BFD, un *traceroute* MPLS doit être lancé par le routeur pour localiser l'erreur. Une fois le routeur en cause trouvé (il peut aussi s'agir d'un de ses liens), une vérification doit être menée, grâce à SNMP : états des interfaces, des sessions LDP. Cette vérification devra permettre d'établir la cause du problème. SNMP permet également de savoir si FRR est activé (avec l'objet *mplsTunnelLocalProtectInUse* de la table *mplsTunnelTable* de la MIB MPLS TE). Les opérations BFD, *ping* et *traceroute* doivent être lancées par le routeur à la demande de QMA.

3.2.3 Méthode de prévision de trafic (spécification 6)

Afin d'optimiser la gestion des ressources du réseau, il est important de connaître et de prévoir le trafic, afin d'anticiper les congestions grâce à une allocation dynamique de ressources (réservation de bande passante ou utilisation de chemins secondaires). La théorie de prédiction, qui s'applique à de nombreux domaines variés, trouve donc une utilité particulière dans la gestion de réseaux informatiques.

3.2.3.1 Position du problème

On souhaite connaître, pour un lien donné, la bande passante d'un lien, le plus souvent possible, et avec le minimum d'erreur. Le mot « souvent » étant subjectif, il conviendra à l'utilisateur de définir l'intervalle de temps entre chaque valeur calculée. Les résultats doivent être calculés automatiquement, sans intervention humaine.

La difficulté consiste à trouver le compromis pour le bon nombre de mesures à effectuer sur le réseau : trop de mesures génèrent beaucoup de trafic et peuvent donner de l'information redondante, alors qu'un nombre insuffisant de mesures nuit à la précision. Il s'agit donc de trouver une méthode de prévision qui permet des précisions avec un horizon plus ou moins large.

Le trafic est supposé inconnu, rendant toute modélisation aléatoire et par conséquent risquée. Néanmoins, on peut conjecturer qu'une saisonnalité sera présente (le trafic étant généralement plus faible la nuit et les jours de repos).

3.2.3.2 Différentes alternatives

Le besoin de prévision étant très largement répandu, dans divers domaines, plusieurs méthodes de prévision ont été mises au point. On peut diviser les méthodes en 3 catégories : les méthodes par expertise, les méthodes explicatives, et les méthodes par extrapolation.

Méthodes par expertise

Il s'agit de méthodes subjectives où l'on détermine la prévision d'après l'avis d'un certain nombre d'experts. L'exemple le plus connu est la méthode Delphi. On interroge plusieurs experts sur une prédiction à effectuer, compile les résultats obtenus, et

réinterroge les experts en leur faisant tenir compte des résultats globaux. Ce processus est réitéré autant de fois que jugé nécessaire.

Bien qu'intéressante dans des cas très précis, cette approche souffre de deux principaux inconvénients : son efficacité repose sur la qualité des personnes interrogées, mais surtout il est très difficile, voire impossible, d'automatiser ce type de méthode. Les méthodes par expertises sont donc inappropriées pour notre problématique.

Méthodes explicatives

On compare ici la série à prévoir avec d'autres séries, qui servent alors de références. En démontrant le lien entre la série et ses références, on peut la prévoir à partir des estimations de celles-là. Or, si là encore dans plusieurs situations précises une telle approche peut s'avérer payante, il convient de trouver des séries « références » que l'on connaît davantage, afin d'obtenir des estimés fiables rapidement. Ne disposant que de peu d'informations sur le trafic que l'on cherche à prévoir, il paraît difficile d'expliquer notre série à l'aide d'autres séries plus connues.

Méthodes par extrapolation

Les méthodes par extrapolation consistent à étudier le passé d'une série afin d'en apprendre le plus possible, pour ensuite deviner les valeurs futures en extrapolant le passé. Examinons différentes méthodes par extrapolation :

Analyse des composantes

Une série peut être décomposée en une combinaison des éléments suivants [57] :

$$X_t=T_t+S_t+C_t+\varepsilon_t \tag{3.1}$$

a. T_t Tendance : effet à long terme de la série;

b. S_t Saison : composante périodique;
c. C_t Cycle : traduit une période (exemple en économie, période de récession, période de prospérité);
d. ε_t Composante aléatoire : partie inexpliquée de la série.

Par exemple, pour une suite à saisonnalité additive (la saison s'ajoute à la tendance), la suite se décomposera ainsi :

$$X_t=T_t+S_t+C_t+\varepsilon_t \tag{3.2}$$

Dans le cadre d'une suite à saisonnalité multiplicative, on aura plutôt :

$$X_t=T_tS_t+C_t+\varepsilon_t \tag{3.3}$$

En séparant ces différentes composantes, par filtrage notamment, on peut modéliser chacune d'entre elles et ainsi prévoir la série comme somme des prévisions de ses composantes. Cette technique est particulièrement intéressante en ce qu'elle permet de comprendre la série, ce qui la compose. En revanche, elle est compliquée, et donc coûteuse en termes de calculs : plusieurs opérations sont à effectuer : filtrage/séparation des composantes, modélisation de chacune, prévision. De plus, elle est risquée car les erreurs de modélisation de chacune des composantes s'additionnent lors de la prédiction de la série.

Lissage exponentiel

Le lissage exponentiel [57] suppose que la série va suivre le même modèle que dans le passé. Il consiste, dans sa forme la plus simple, à prédire la série comme somme des valeurs passées pondérées par des coefficients calculés de façon à privilégier les valeurs récentes.

Le cas le plus basique est le lissage exponentiel simple, où, pour une suite y_t, la valeur estimée S_t est

$$S_t=\alpha y_{t-1}+(1-\alpha)S_{t-1}. \tag{3.4}$$

Cependant ce premier modèle s'est vite avéré insuffisant pour des séries sophistiquées (du moins avec présence d'une tendance), d'où l'apparition de lissage exponentiel double voire triple (dans le cas où la série présente une tendance et une saisonnalité). Plus de calculs sont alors nécessaires.

Filtre de Kalman

Le filtre de Kalman [57] est un filtre récursif puissant permettant de déterminer l'état du système à partir de mesures incomplètes et bruitées, en minimisant la covariance de l'erreur d'estimation. Les applications de cette théorie sont diverses, et en particulier dans le domaine des réseaux de télécommunications.

Le filtre de Kalman suppose que le trafic est modélisé sous la forme suivante :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) + w(k) \\ y(k) = Cx(k) + v(k) \end{cases} \tag{3.5}$$

Où x(k) désigne le vecteur d'état, y(k) le vecteur de sortie, u(k) le vecteur de contrôle, w(k) et v(k) les bruits de modélisation et observation, supposés de loi normale avec moyenne nulle et covariances respectives Q(k) et R(k).

Le calcul se fait alors en deux étapes :

Prédiction :

$$\begin{cases} \hat{x}(m/m-1) = A\hat{x}(m-1/m-1) + Bu(m) \\ P(m/m-1) = AP(m-1/m-1)A^T + Q(m) \end{cases} \tag{3.6}$$

Estimation :

$$\begin{cases} K(m) = P(m/m-1)C^T[CP(m/m-1)C^T + R(m)]^{-1} \\ \hat{x}(m/m) = \hat{x}(m/m-1) + K(m)[y(m) - C\hat{x}(m/m-1)] \\ P(m/m) = \{I - K(m)C\}P(m/m-1) \end{cases} \quad (3.7)$$

Le filtre de Kalman est une technique puissante dont la précision n'est plus à démontrer. Cependant il suppose que le modèle sur lequel sont bâties les équations est correct. En effet, des chercheurs l'ayant implémenté [36, 58] l'ont toujours fait dans le cadre d'un réseau particulier avec des hypothèses précises. Ainsi, pour un trafic dont on ne sait finalement que peu de choses, il est hasardeux d'établir un modèle sur lequel reposera le filtre de Kalman. Des méthodes implémentent un filtrage de Kalman adaptatif, où le modèle est dynamiquement adapté, mais elles demeurent très sophistiquées et donc par là même dispendieuses en termes de calculs.

Par ailleurs, son horizon de prédiction est de 1 : il peut prévoir la valeur à *t+1*, mais pas *t+2*, *t+3*... Nul doute donc de la richesse du filtre de Kalman, mais il s'adapte finalement mal au cadre de notre projet.

3.2.3.3 Choix d'une méthode : Holt-Winters

Il n'existe pas de méthode miracle pour toutes les situations, et le nombre de méthodes de prédiction existantes reflète bien la diversité des situations rencontrées. Il convient donc de choisir d'après le problème la méthode la plus adéquate.

Après examen des différentes méthodes, la meilleure approche qui nous permet de faire des prévisions relativement fiables pour un trafic qu'on connaît peu semble être le lissage exponentiel triple, aussi appelé méthode Holt-Winters. En effet, il est adapté pour une série avec tendance et saisonnalité, on peut choisir l'horizon de prédiction et surtout

cette méthode peut être automatisée. Ses diverses qualités en font donc une technique appréciée [57, 59, 60].

Les équations sont les suivantes [60] :

$$St = \alpha \frac{yt}{It - L} + (1 - \alpha)(St - 1 + bt - 1) \qquad \textit{lissage général}$$

$$bt = \gamma(St - St - 1) + (1 - \gamma)bt - 1 \qquad \textit{lissage de la tendance}$$

$$It = \beta \frac{yt}{St} + (1 - \beta)It - L \qquad \textit{lissage de la saison} \qquad (3.8)$$

$$Ft + m = (St + mbt)It - L + m \qquad \textit{prévision}$$

Où :

a. y est la série observée (mesurée);
b. S est l'observation lissée;
c. b est le paramètre de tendance;
d. I est la composante saisonnière;
e. $Ft+m$ est la prédiction à t+m;
f. t est l'indice de temps

α, β et γ désignent les coefficients déterminés de façon à minimiser l'erreur carrée moyenne.

Simulation

Pour éprouver cette théorie et ainsi constater par nous-mêmes ses capacités, nous l'avons implémentée à l'aide de l'outil Matlab sur une série chronologique. Nous avons composé notre série chronologique d'une tendance linéaire (le trafic augmente petit à petit), d'une saisonnalité multiplicative, censée représenter les variations journalières, et d'une composante aléatoire non négligeable.

Nous sommes partis de 500 valeurs de départ, et souhaitons connaître les 100 prochaines valeurs, à raison d'une mesure tous les cinq. Entre chaque mesure, il faut donc prévoir quatre valeurs. Les coefficients α, β et γ optimaux sont calculés par notre programme.

La figure 20 présente la suite originale.

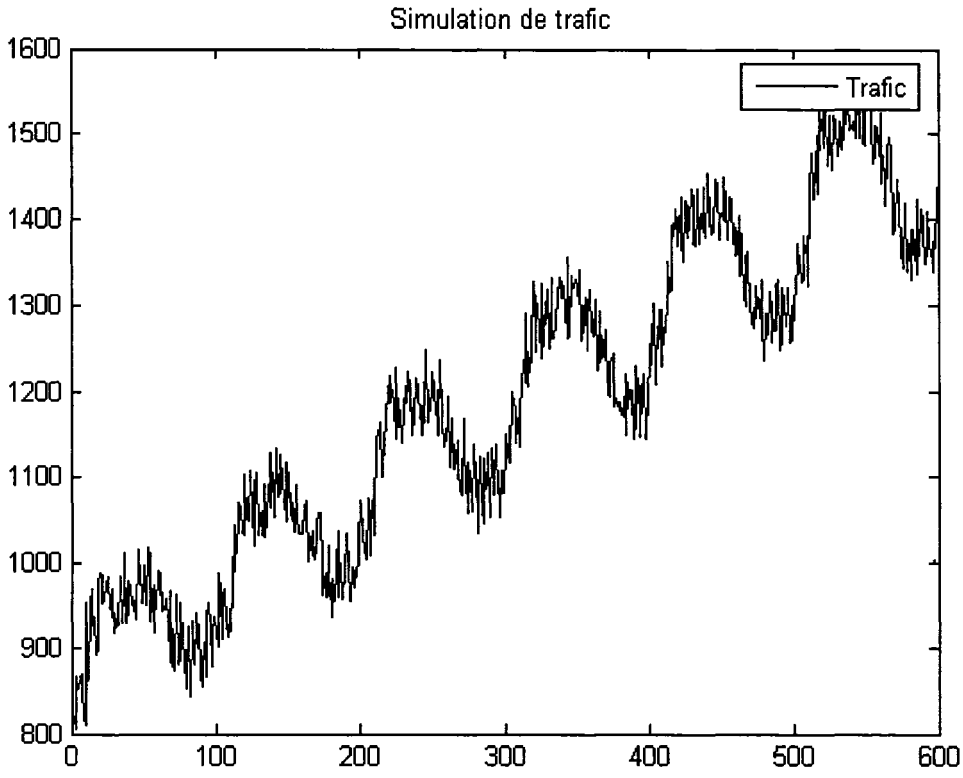

Figure 20 Simulation d'une suite chronologique sous Matlab

La figure 21 présente le résultat de la prédiction par rapport à la suite originale. La prédiction suit fidèlement la suite, malgré quelques écarts. Mais la mesure, toutes les cinq unités de temps, réajuste notre prédiction.

Notons toutefois que du fait d'une composante aléatoire, chaque simulation ne donnera pas exactement les mêmes résultats, mais après avoir effectué plusieurs simulations, nous constatons que les résultats sont similaires.

Nous avons ainsi effectué 1000 simulations et obtenu une erreur carrée moyenne de 400 pour des valeurs autour de 1500.

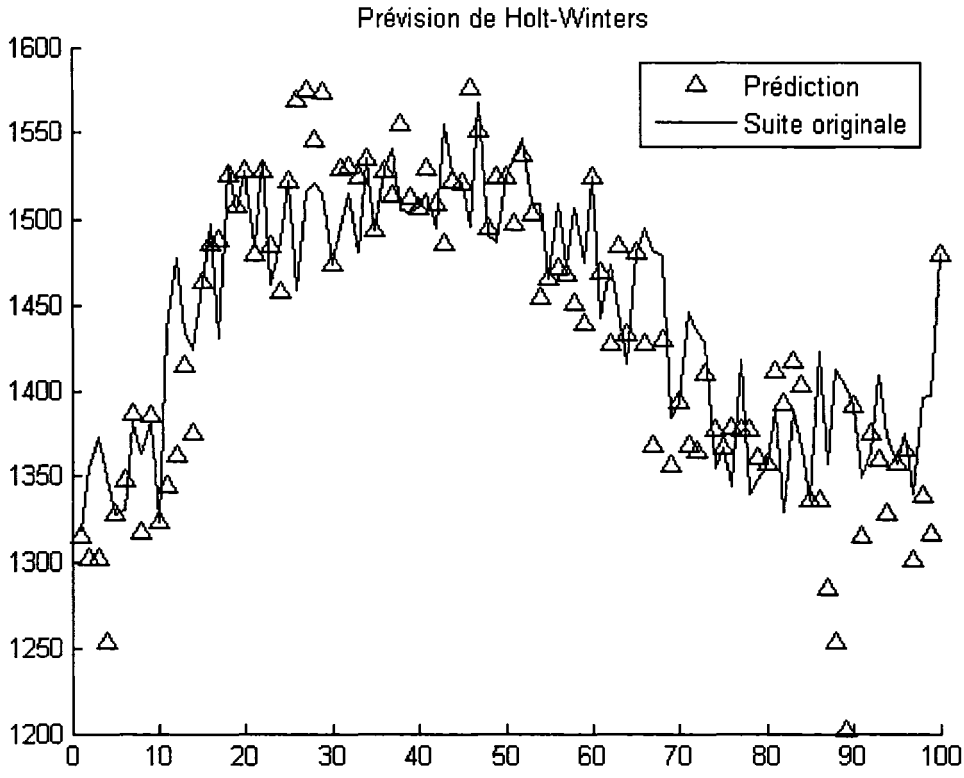

Figure 21 Implémentation de la méthode de Holt-Winters sous Matlab

Le code MATLAB des simulations est disponible dans l'annexe 1.

3.2.4 Mode d'authentification : mot de passe (spécification 7)

QMA doit imposer une authentification pour s'assurer que seules les personnes autorisées peuvent l'utiliser et accéder aux informations. Plusieurs modes d'authentification existent, du mot de passe à la biométrie. Néanmoins dans le cadre de cet outil, il ne s'agit pas de prendre un mode démesuré par rapport à l'information à protéger. On suppose que l'accès à l'outil en lui-même sera déjà protégé. Le traditionnel usage du nom d'utilisateur couplé au mot de passe est donc repris ici pour sa simplicité et son acceptation au sein du personnel.

3.2.5 Choix d'une architecture (spécifications 8 à 10)

3.2.5.1 Position du problème

QMA doit pouvoir être utilisé simultanément par plusieurs personnes (spécification 10). QMA ne peut donc être un seul programme, une architecture de type client-serveur doit être déployée : différents clients effectuent des requêtes auprès d'un serveur en charge de leur répondre. Seul le serveur communique avec le réseau, supprimant ainsi toute redondance d'informations. De plus, cette architecture est préférable pour des raisons de sécurité : l'accès au réseau MPLS ne passe que par le serveur, ce qui facilite son contrôle.

3.2.5.2 Différentes alternatives

Plusieurs architectures client-serveur existent, notamment les architectures à 2 niveaux (*2-tier*) et celles à 3 niveaux (*3-tier*). L'architecture 2-tier est illustré à la figure 22 [61]. Le serveur est alors responsable du traitement des requêtes du client et de l'interaction avec la base de données. Ce modèle est simple, mais offre moins de souplesse que le

modèle 3-tier. En effet, le modèle 3-tier permet une plus grande flexibilité en séparant les différentes tâches.

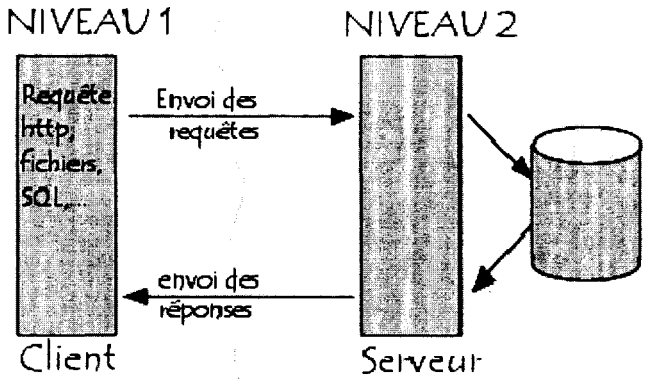

Figure 22 Architecture 2-tier

3.2.5.3 Choix de l'architecture : 3-tier avec serveur Web

Nous choisissons ici l'architecture 3-tier, légèrement plus complexe que l'architecture 2-tier mais qui apporte la flexibilité recherchée. Sa mise en place dans notre projet est présentée à la figure 23.

Dans notre architecture 3-tier, chacun des composants a un rôle bien précis :

- Le client cherche l'information auprès du serveur, et la présente de manière ergonomique à l'utilisateur.
- Le serveur répond aux requêtes du client, en interrogeant les modules adéquats. Il sert principalement d'interface.
- Les modules sont véritablement le cœur de l'outil. Ils gèrent la base de données, communiquent avec le réseau MPLS et exploitent les informations obtenues.

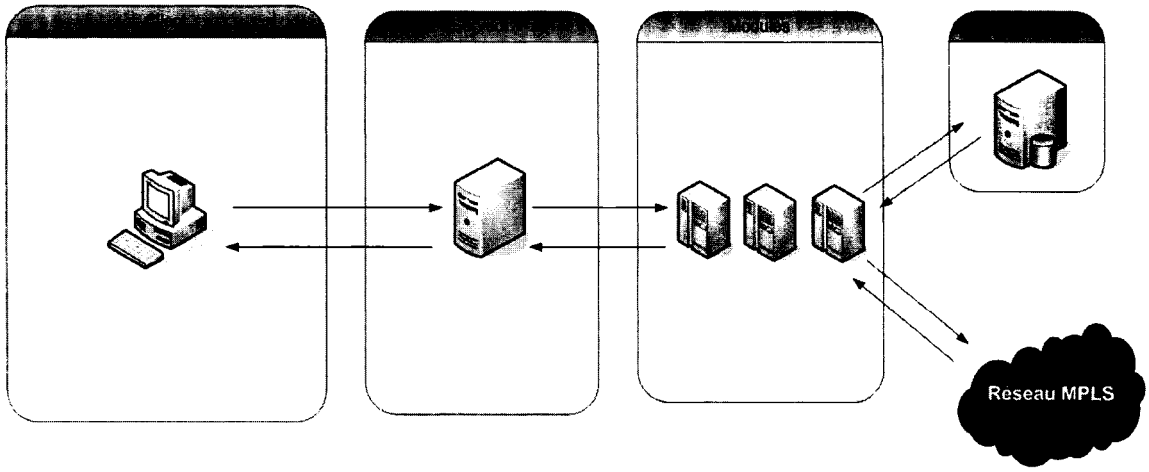

Figure 23 Architecture 3-tier de QMA

Ce modèle permet donc une approche modulaire, où à chaque partie est affecté un rôle bien précis. Il permet une plus grande flexibilité : par exemple le changement de base de données affectera les modules mais restera transparent au serveur web. Il est en outre possible d'ajouter des mécanismes de sécurité à chaque niveau, rendant le tout plus robuste. Par ailleurs, les différents modules sont cachés à l'utilisateur qui ne connaît que le serveur. Pour garantir la confidentialité la communication entre client et serveur se fait par le protocole HTTPS utilisant la technologie *Secure Socket Layer* (SSL).

De plus, pour faciliter la disponibilité de cet outil, la partie client sera téléchargée depuis un serveur web à l'instar d'une applet. Cela dispense les différents utilisateurs de devoir installer un programme client spécifique. De plus, en cas de contrôles d'accès accrus, notamment par pare-feu, le recours au web évite de devoir modifier les configurations relatives aux ports autorisés, le port 80 étant un port commun.

3.2.6 Des composants modulaires pour plus d'évolutivité (spécification 11)

QMA est un projet qui se fait conjointement : comme nous l'avons mentionné précédemment, il comprend un aspect QoS et un aspect MPLS. De plus il doit pouvoir

être repris par d'autres étudiants ou professionnels afin de permettre l'ajout ou le perfectionnement de fonctionnalités.

Pour ce besoin majeur d'évolutivité, la solution retenue consiste donc à offrir une structure la plus modulaire possible, afin que certains composants puissent être aussi bien utilisés pour l'aspect QoS de QMA que pour MPLS, mais aussi qu'ils puissent être facilement repris. La modularité de QMA s'exprime dans son implémentation, présentée au chapitre suivant.

3.2.7 Une interface simple et ergonomique (spécification 12 et 13)

Comme pour la spécification précédente, la solution apportée aux exigences de simplicité et d'ergonomie se manifeste dans l'implémentation de QMA où l'utilisation d'onglets limite le nombre de clics nécessaires à l'accès de chacune des fonctionnalités.

3.3 Choix technologiques

Comme nous l'avons expliqué précédemment, QMA repose sur une architecture 3-tier pour une meilleure souplesse et plus de sécurité. Se pose alors le choix des technologies à utiliser pour implémenter l'outil.

3.3.1 Plate-forme de développement

Pour implémenter une solution web, deux technologies phares sont reconnues :

a. *Java Platform, Enterprise Edition* (Java EE) anciennement J2EE, par Sun [62];
b. *.Net* proposée par Microsoft [63].

Chacune des technologies avec ses forces et faiblesses respectives a été explicité par un collaborateur du projet [64]. De nombreuses ressources comparant ces deux technologies

abondent sur Internet. Le souci d'objectivité n'est cependant pas toujours respecté, certains choisissant parfois l'une ou l'autre technologie pour des raisons affectives et/ou idéologiques.

La figure 22 résume les services proposés par ces deux technologies. Rappelons que les plateformes .Net et Java EE sont concurrentes et qu'à ce titre, aucune ne se démarque vraiment de l'autre dans les fonctionnalités offertes, du fait de leur compétition perpétuelle. Java EE est arrivé en 1998 sur le marché, soit trois ans avant .Net. Le premier bénéficie donc de plus d'expérience et en outre du soutien de la communauté *OpenSource*. De plus, il est interopérable, pouvant se déployer sur différents systèmes d'exploitation. .Net quant à lui comble son manque de maturité par une simplicité de développement, permettant notamment d'utiliser plusieurs langages de développement, dont C# spécialement conçu pour cette technologie. Ainsi, il tendrait à gagner en popularité. Néanmoins, il reste à ce jour utilisable seulement sur Microsoft *Internet Information Services* (IIS) lequel nécessite un système d'exploitation de la même compagnie. De plus, des fonctionnalités avancées comme le recours à une applet .Net ne sont actuellement possibles qu'avec l'utilisation du navigateur *Internet Explorer*. Les comparaisons en termes de performances restent contentieuses, nous les supposerons donc sensiblement équivalentes.

L'équipe du projet Bell s'est donc retrouvée face un choix : d'un côté Java EE, une technologie interopérable, bénéficiant d'une expérience certaine, mais qui reste complexe à développer et à maintenir, et de l'autre .Net, technologie jeune mais prometteuse à l'environnement de développement puissant *Visual Studio .Net*. La décision a alors été prise de retenir .Net pour ce projet, pour son côté novateur. Dans le cadre de ce mémoire, le choix de .Net a ainsi été imposé.

Couche de présentation: génération HTML		
J2EE	**JSP**	Les Java Server Pages permettent de créer des pages HTML dynamiques, créées à la volée à partir de contenus et de sources diverses.
	JSF	Les JavaServer Faces étendent les capacités des JSP pour faciliter la création et la mise à jour d'objets au sein de l'interface (barre de navigation, etc.).
	Servlets	Ils définissent la logique de navigation d'un site Web en conjonction avec les JSP (état des cessions, etc.).
.Net	**WinForms**	La classe Windows Forms gère l'interface utilisateur de .Net en environnement Windows.
	ASP.Net	Conçu pour le serveur IIS, Active Server Pages .Net gère la génération de pages HTML, mais également l'état des cessions ainsi que l'authentification des utilisateurs.
Logique applicative		
Transaction		
J2EE	**EJB**	Les transactions J2EE, c'est-à-dire les tâches prises en charge par l'application en tant que telle, peuvent être codés par le développeur ou bien exécutées par des composants appelés Enterprise Java Bean.
.Net	**Serviced Components (objets COM, COM+)**	Au sein de la plate-forme .Net, les transactions sont directement gérées par le CLR (pour Common Language Runtime) via les objets COM et COM+.
Appel d'objets distribués		
J2EE	**JNDI**	Java Naming and Directory Interface permet d'invoquer des composants Java (EJB et JMS) au sein d'un environnement de serveurs en grappe.
.Net	**.Net Remoting**	Ce module gère l'appel d'objets distribués aussi bien entre applications, que processus ou encore machines.
Accès aux données		
J2EE	**EJB**	Les EJB offrent des services de gestion des transactions, mais aussi une infrastructure conçue pour exécuter la mise à jour des bases de données utilisées (par le biais de JDBC).
.Net	**ADO .Net Classes**	Issus de ActiveX Data Objects, ADO .Net Classes s'appuie sur diverses interfaces (ODBC, Tec.) pour gérer les accès à des sources de données tierces (XML, etc.), et de publier ces dernières par le biais de Web Services notamment.
Langages supportés		
Transaction		
J2EE	**Java**	La plate-forme J2EE est adaptée au Java. D'autres langages peuvent néanmoins être utilisés en s'appuyant sur une interface, comme Java Native Interface (JNI) ou encore les Web Services.
.Net	**"Agnostique"**	L'environnement .Net est indépendant des langages de développement, pour peu qu'il intègre un dispositif de cartographie dessiné au cas par cas. Pour l'heure, quelques éditeurs, comme Fujitsu (avec NetCOBOL) ou encore ActiveState (avec Visual Perl et Visual Python) ont produit des compilateurs CLR. Une démarche qui nécessite la signature d'un partenariat avec Microsoft.

Figure 24 Services offerts par .Net et J2EE

3.3.2 Serveur de base de données

Le second choix technologique à faire concerne le serveur de la base de données. De nombreuses possibilités existent, que ce soit Oracle [65], MySql [66] ou PostGreSql [67] pour n'en citer que quelques-uns. Dans le cadre de ce projet, nous avons choisi Microsoft SQL Server [68], du moins la version Express, pour son intégration avec .Net et sa facilité de gestion.

Notons cependant que ce choix reste d'importance secondaire : en effet, de par la nature modulaire de QMA, toute modification du type de serveur n'impactera que le module en charge de la communication avec la base de données. Outre la migration des données, il sera donc facile si besoin de changer le serveur.

CHAPITRE 4

CONCEPTION ET IMPLÉMENTATION

« Le savoir n'est pas difficile, seule sa mise en pratique l'est. »
Proverbe Chinois

Le chapitre précédent a expliqué l'objectif de QMA, les différents problèmes qu'il aborde, et pour chacun la solution proposée. À présent, la conception et l'implémentation sont présentées, pour montrer en pratique comment QMA est fait. Ce chapitre débute par une vue d'ensemble des différents composants, puis chacun est détaillé, en précisant notamment les classes utilisées, les méthodes et algorithmes mis en place. Une fois que chacun des composants aura été ainsi examiné, des diagrammes de séquence commentés illustreront le fonctionnement global de QMA. Enfin, le chapitre conclura sur les éléments originellement prévus dans QMA qui n'ont pas été implémentés en expliquant pourquoi. La présentation effectuée ici est technique, du point de vue du développeur. Pour découvrir QMA du côté de l'usager, des guides d'installation et d'utilisation sont rédigés en annexes 2 et 3.

Le choix de .Net ayant été retenu pour ce projet, QMA a été implémenté avec l'environnement de développement Microsoft Visual Studio [69], d'abord la version Visual Studio .Net, puis Visual Studio 2005, une fois celle-ci disponible.

Les diagrammes de classes proposés ici ne sont pas exhaustifs pour des raisons de clarté et de concision. Les listes complètes des composants de chacune des classes alourdiraient considérablement le mémoire, n'en rendant sa lecture que plus laborieuse. Ne sont donc présentés ici que les membres et méthodes jugés importants pour la compréhension de QMA. Ces diagrammes ont été générés automatiquement par Visual Studio.

4.1 Vue d'ensemble de QMA

Comme mentionné précédemment, QMA adopte une architecture 3-tier. La figure 25 illustre plus en détail comment celle-ci est réalisée.

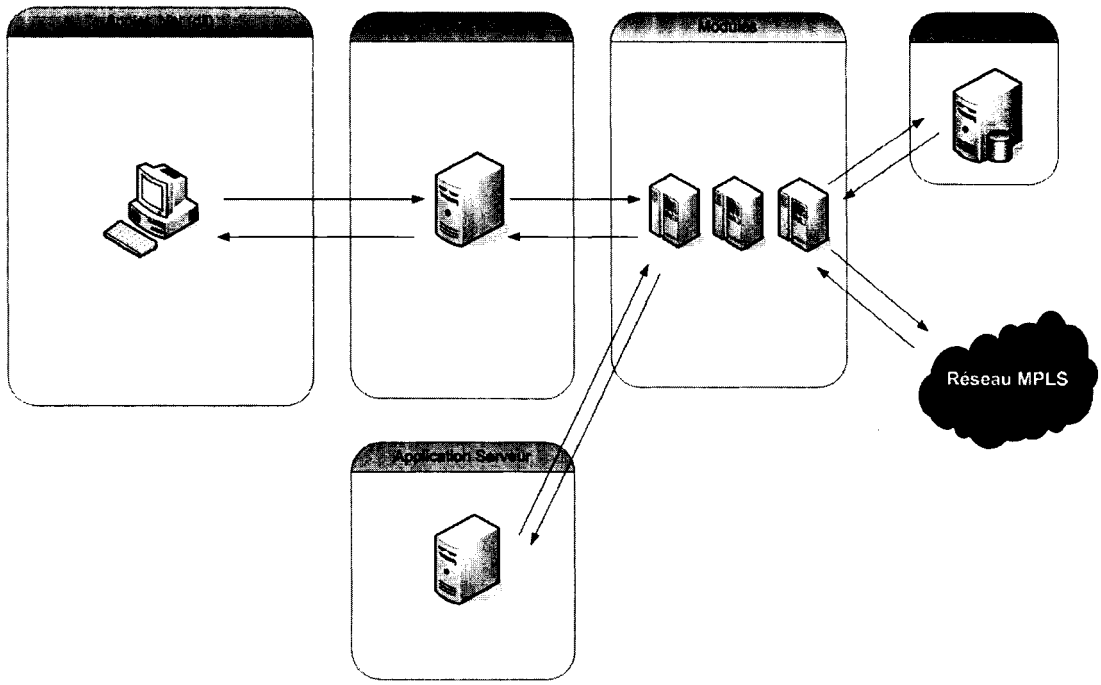

Figure 25 Architecture de l'implémentation de QMA

Les composants sont les suivants :

a. L'application client : comme expliqué précédemment, celle-ci a exclusivement pour rôle l'affichage des informations et sert d'interface entre l'usager et l'outil. Diverses options ont été explorées par un autre membre du projet [64], et l'utilisation d'une applet .Net a été retenue. En effet, de simples pages statiques ou même des images générées dynamiquement ne suffisent pas à répondre aux fonctionnalités souhaitées : l'interactivité souhaitée requiert qu'un programme à

part entière soit exécuté côté client. Celui-ci doit en outre être exécutable depuis une page Web, d'où l'applet .Net;

b. Le serveur Web : il s'agit en fait d'un Webservice. En effet, on souhaite ici rendre accessible à plusieurs utilisateurs un service dont la logique interne reste cachée aux utilisateurs, pour des raisons de sécurité. Il est donc l'interlocuteur exclusif de l'applet;

c. L'application serveur : ce programme prend en charge plusieurs tâches précises. En effet, certaines opérations ne doivent être effectuées qu'une seule fois, indépendamment du nombre de clients connectés, par exemple la collecte de statistiques ou la surveillance de réseau. Il est donc pratique d'avoir un composant en charge d'exécuter les tâches singulières. En réalité il est tout à fait possible de s'affranchir de ce composant, en agissant au niveau du Webservice, mais cette application serveur est nécessaire à la partie QoS de QMA, donc étant implémentée de toute façon, nous en avons profité pour y ajouter certaines fonctions;

d. Les modules : il s'agit d'une bibliothèque de classes implémentant les fonctionnalités de QMA, appelée par le Webservice et l'application serveur;

e. La base de données : il s'agit de Microsoft SQL Server. La base est donc de type relationnelle et le langage utilisé SQL.

Sous Visual Studio, chaque composant est un projet, le tout formant une solution comme le montre la figure 26.

Deux types d'utilisateurs sont définis dans QMA :

a. Les administrateurs, ayant accès à toutes les fonctionnalités de QMA;

b. Les autres utilisateurs, ne pouvant disposer de certaines fonctionnalités comme la modification de certains paramètres serveurs, ou la visualisation du fichier de logs. Cette distinction est précisée dans les sections suivantes.

À présent, expliquons chacun des composants plus en détail. Il n'a pas été jugé judicieux de présenter dans ce chapitre le modèle de la base de données. Celui-ci est néanmoins disponible à l'annexe 4.

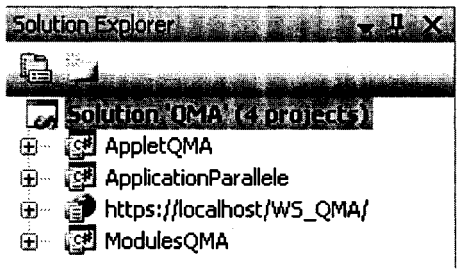

Figure 26 Explorateur de la solution QMA sous Visual Studio

4.2 Côté client : Applet

L'applet est un programme devant s'exécuter dans une page Web. Son organisation est présentée à la figure 27. Ses différents éléments sont les suivants :

a. Les contrôles utilisateurs (*User control*) peuvent être perçus comme différentes briques, chacune ayant un rôle précis, formant un tout cohérent et fonctionnel. Ils sont regroupés en un contrôle plus grand, appelé d'ailleurs QMA qui structure l'applet, comme le montre la figure 28. Chaque contrôle est explicité dans une section subséquente, sauf *QoSConfig.cs* et *QosStatsUC.cs* exclusivement dédiés à la QoS et donc non traités ici;

b. Les classes du répertoire *GraphContainer* servent à la représentation graphique des informations. Elles sont utilisées dans le contrôle utilisateur

DisplayControl.cs et seront donc présentées dans la section qui leur est consacrée;

c. Les classes du répertoire *InfoContainer* servent à la modélisation des différentes informations visualisées. Elles sont également utilisées dans le contrôle utilisateur *DisplayControl.cs* et seront donc présentées dans la section qui leur est consacrée;

d. Les autres éléments sont soit des images, soit des fichiers générés automatiquement par Visual Studio pour son bon fonctionnement. Une référence au WebService est définie explicitement.

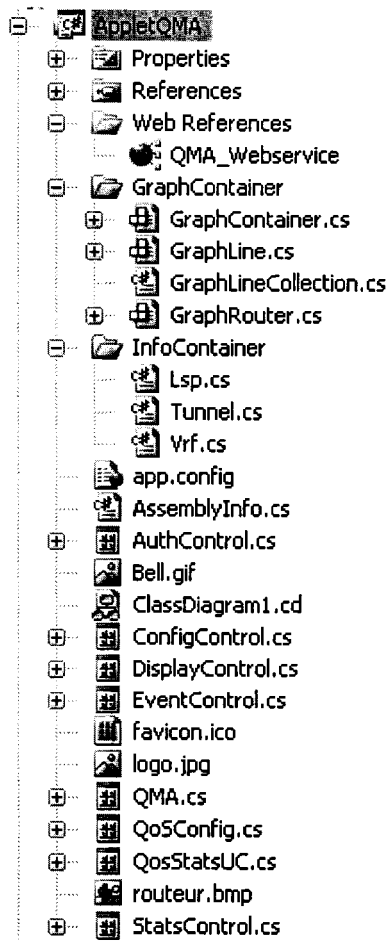

Figure 27 Structure de l'applet sous Visual Studio

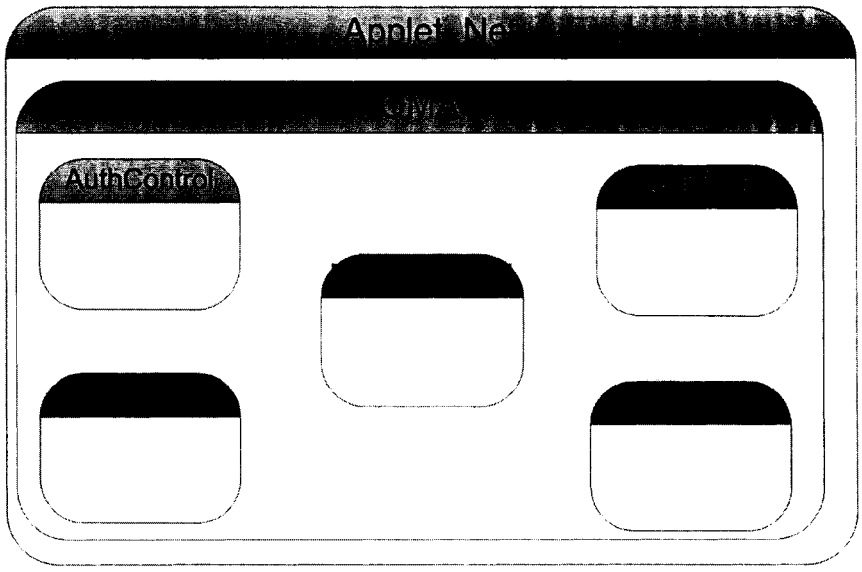

Figure 28 Structure des contrôles utilisateurs dans l'applet

4.2.1 Contrôle utilisateur AuthControl

Ce contrôle utilisateur a pour rôle de fournir l'interface utilisateur permettant à celui-ci de s'authentifier à l'aide d'un nom d'utilisateur et d'un mot de passe. Ces deux informations sont transmises au contrôle utilisateur *QMA*. La figure 29 montre la structure du contrôle utilisateur *Authentication.*

Cette classe contient donc des *TextBox* ainsi que des accesseurs *Get* pour lire le nom d'utilisateur et le mot de passe rentrés par l'utilisateur. L'utilisation du protocole HTTPS avec SSL entre l'applet et le Webservice garantit la confidentialité de la vérification, aucune information ne transitant sur le réseau en clair.

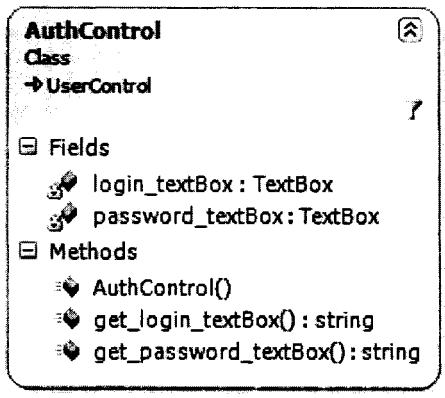

Figure 29 Structure de AuthControl

4.2.2 Contrôle utilisateur DisplayControl

4.2.2.1 Classe DisplayControl

Ce contrôle utilisateur, le plus complexe de l'applet, affiche les informations du réseau. La structure de cette classe est présentée figure 30. Elle a tout d'abord été implémentée par un autre membre du projet [64] puis reprise et améliorée.

Ses membres sont les suivants :

a. *_QMA_WS* : permet de faire la connexion avec le Webservice pour obtenir les informations;
b. Les routeurs du réseau sont représentés par des objets *GraphRouter*, qui comme les autres classes de représentation graphique, sont explicités en [64];
c. Les champs *ListBox* et *TextBox* servent à l'affichage des informations;
d. *statsControl*, explicité ultérieurement permet l'affichage des statistiques de trafic dans un tunnel;
e. *timer_stats* : minuteur qui permet de collecter les statistiques à une fréquence donnée (par exemple 20 sec);

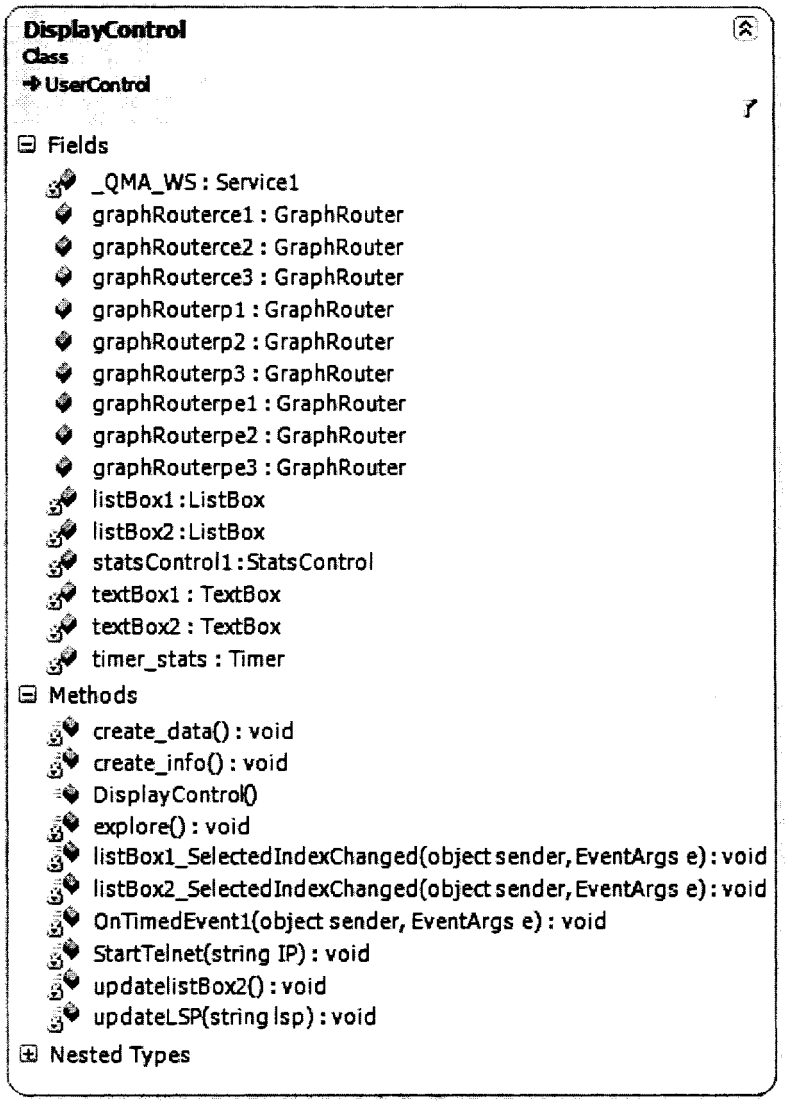

Figure 30 Structure de DisplayControl

Les méthodes sont les suivantes :

a. *create_data* : selon le type de données demandé (choisi par un menu), cette méthode demande au WebService des informations précises et instancie des objets *LSP*, *Tunnel* ou *Vrf* le cas échéant. Ceux-ci sont expliqués un peu plus tard;

b. *create_info* : appelée juste après *create_data*, cette méthode remplit la *ListBox1* selon le type d'informations : par exemple la liste des VPN;

c. *explore* : méthode qui demande un rafraîchissement complet de la base de données;
d. *listBox1_SelectedIndexChanged* : cette méthode, appelée lorsqu'un item dans ListBox1 est sélectionné, nettoie le panel graphique (par exemple des LSP) et appelle *updatelistBox2*;
e. *listBox2_SelectedIndexChanged* : cette méthode affiche les informations dans les *TexBox* propres à l'item sélectionné. En reprenant l'exemple du VPN, en choisissant un des VRF, la méthode va afficher dans *TextBox1* la configuration du VRF et dans *TextBox2* ses routes associées;
f. *OnTimedEvent1* : cette méthode est appelée chaque fois que le minuteur des statistiques atteint son intervalle : elle demande alors au Webservice les statistiques rafraîchies;
g. *StartTelnet (string IP)* : démarre une connexion Telnet vers le routeur dont l'IP est spécifiée en argument;
h. *updatelistBox2* : cette méthode, appelée par *listBox1_SelectedIndexChanged*, affiche les informations relatives à cet item dans la *ListBox2*, dépendamment du type d'information. Par exemple, si la *ListBox1* contient la liste des VPN, en choisissant un des items, la méthode va faire afficher dans ListBox2 la liste des VRF formant le VPN sélectionné;
i. *updateLSP(string LSP)* : dessine le LSP dont le nom est fourni en argument.

4.2.2.2 Classe LSP

Les données sont représentées dans trois classes, contenues dans le répertoire *InfoContainer*. La première est la classe *LSP*, dont la structure est illustrée figure 31.

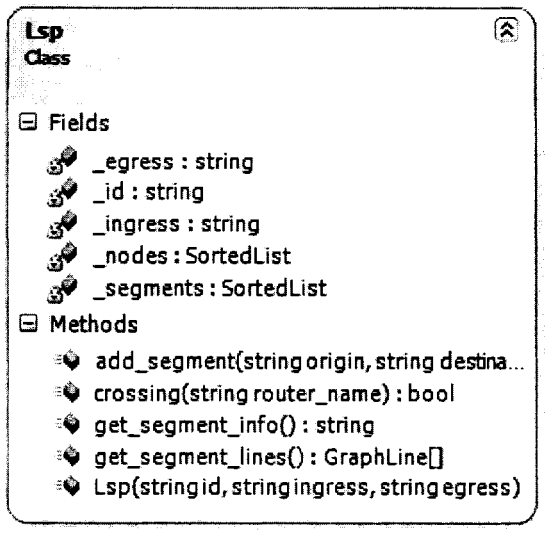

Figure 31 Structure de LSP

Les membres représentent les caractéristiques des LSP :

a. *_egress* : nom du routeur de sortie du LSP;
b. *_id* : numéro du LSP dans la base de données;
c. *_ingress* : nom du routeur d'entrée du LSP;
d. *_nodes* : liste contenant les noms des routeurs traversés par ce LSP;
e. *_segments* : liste des segments composant le LSP.

Les méthodes sont les suivantes :

a. *add_segment* : ajoute un segment au LSP. Prend comme arguments l'IP et le nom des routeurs de départ et d'arrivée, le label utilisé, l'ordre du segment ainsi que la représentation dans un objet *GraphLine* du lien entre les 2 routeurs;
b. *crossing (string router_name)*: indique si le LSP passe ou non par le routeur dont le nom est indiqué en argument;
c. *get_segment_info* : retourne dans une chaîne de caractères les informations sur les segments du LSP;
d. *get_segment_lines* : retourne la liste des objets *GraphLine* représentant graphiquement les segments;

e. *LSP (string id, string ingress, string egress)* : constructeur qui prend en arguments l'id du LSP dans la base de données ainsi que le nom des routeurs d'origine et de destination.

4.2.2.3 Classe Tunnel

Un tunnel est un LSP avec des propriétés particulières. Ainsi, la classe *Tunnel* hérite de *LSP*. Elle est présentée figure 32.

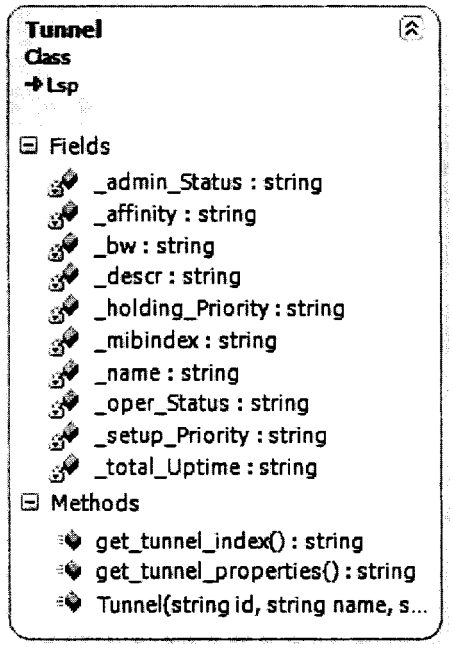

Figure 32 Structure de Tunnel

Ses paramètres sont les propriétés des tunnels :

a. *_admin_Status* : statut administratif du tunnel;
b. *_affinity* : affinité du tunnel;
c. *_bw* : bande passante réservée;
d. *_descr* : description du tunnel;
e. *_holding_Priority* : priorité de maintien du tunnel;

f. *_mibindex* : index du tunnel dans la MIB;
g. *_name* : nom du tunnel;
h. *_oper_Status* : statut opérationnel du tunnel;
i. *_setup_Priority* : priorité d'établissement du tunnel;
j. *_total_Uptime* : *Uptime* du tunnel.

Les méthodes sont les suivantes :

a. *get_tunnel_index* : retourne l'index du tunnel;
b. *get_tunnel_properties* : retourne les propriétés du tunnel;
c. Le constructeur du tunnel qui prend les caractéristiques (cf. membres de la classe) en argument;

4.2.2.4 Classe VRF

La classe *VRF* est représentée figure 33. Ses membres sont les suivants :

a. *_active_interfaces* : nombre d'interfaces actives;
b. *_confstorage_type* : type de stockage;
c. *_current_routes* : nombre de routes configurées;
d. *_description* : description du VRF;
e. *_If_list* : liste des interfaces reliées à ce VRF;
f. *_illegal_label_violations* : nombre de violation de labels dans ce VRF;
g. *_name* : nom du VRF;
h. *_node* : nom du routeur où se trouve le VRF;
i. *_oper_status* : statut opérationnel;
j. *_rd* : *Route Distinguisher* du VRF;
k. *_Route_list* : liste des routes du VRF;
l. *_vrf_index* : index du VRF.

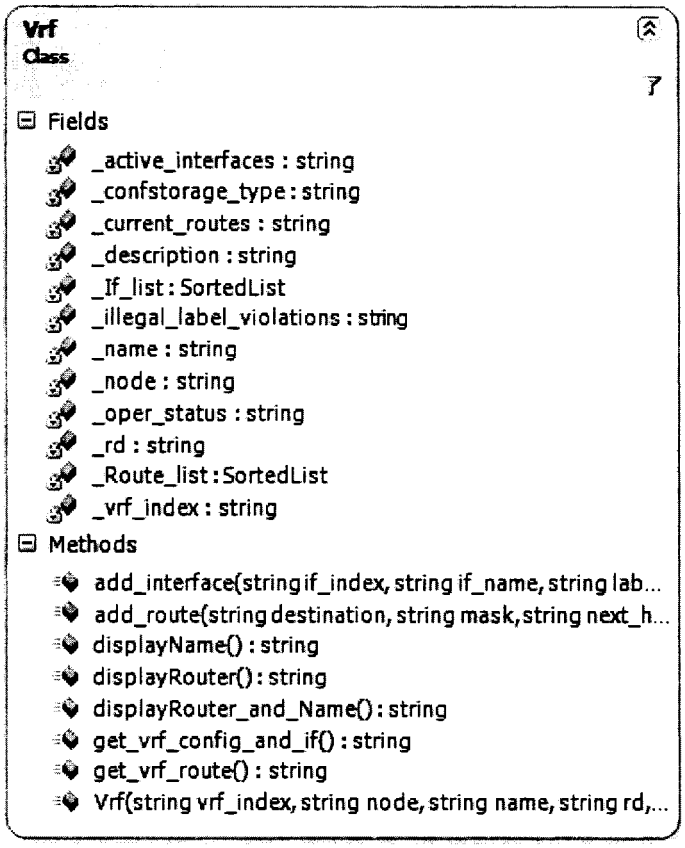

Figure 33 Structure de VRF

Ses méthodes sont les suivantes :

a. *add_interface* : ajoute une interface, avec comme arguments ses propriétés : index, nom, type de label, classification du VPN, type de stockage, index de l'interface;
b. *add_route* : ajoute une route, avec comme arguments ses propriétés : adresse IP de destination, masque, nom du prochain saut, champ TOS, nom de l'interface, type, protocole utilisé, âge, statut opérationnel, type de stockage;
c. *displayName* : retourne le nom du VRF;
d. *displayRouter* : retourne le nom du routeur où se trouve le VRF;
e. *displayRouter_and_Name* : retourne le nom du routeur où se trouve le VRF suivi du nom de celui-ci;

f. *get_vrf_config_and_if* : retourne la configuration du VRF et de ses interfaces;
g. *get_vrf_route* : retourne la configuration des routes du VRF;
h. Le constructeur de la classe qui prend en argument les caractéristiques du VRF décrites dans les paramètres (sauf les listes de routes et d'interfaces).

4.2.3 Contrôle utilisateur EventControl

Ce contrôle a pour tâche de lister les évènements enregistrés par QMA. Il est donc composé d'un *Texbox* pour l'affichage et d'un accesseur pour l'écriture, comme le montre la figure 34.

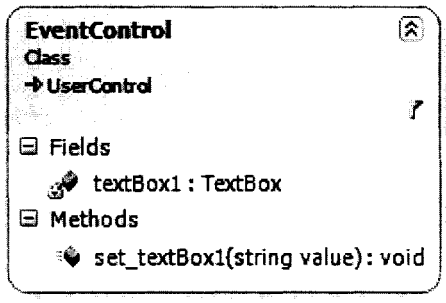

Figure 34 Structure de EventControl

4.2.4 Contrôle utilisateur ConfigControl

Ce contrôle sert à la configuration avancée de l'outil : il permet de visualiser les fichiers de logs et de *traps* de QMA, mais aussi de modifier certaines informations nécessaires à l'outil :

a. Pour la base de données : nom du serveur, le nom des bases utilisées, nom d'utilisateur et mot de passe;
b. Nom d'utilisateur et mot de passe pour SNMP v3 configuré dans les routeurs.

Il est composé de trois onglets : un pour visualiser les fichiers de logs, un pour afficher le fichier des *traps*, et un pour modifier les paramètres cités précédemment. Sa structure est illustrée figure 35.

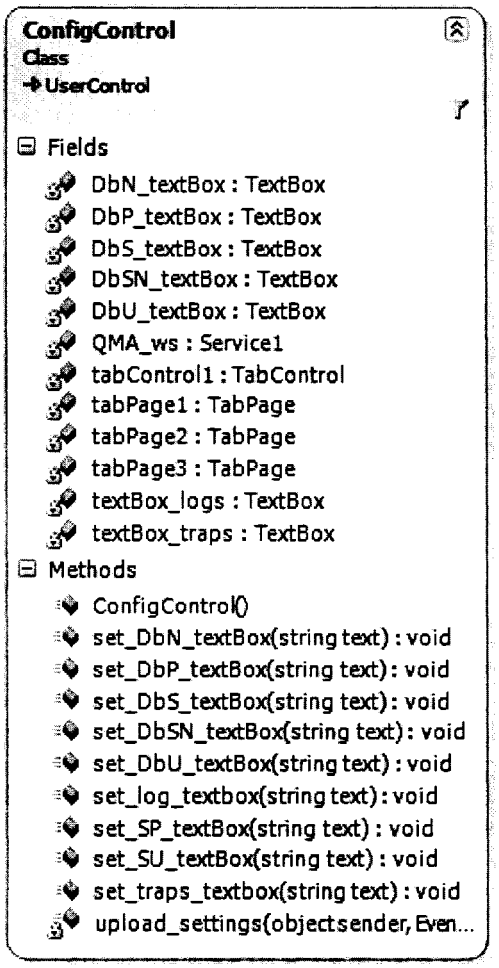

Figure 35 Structure de ConfigControl

Ses paramètres sont les suivants :

a. Un ensemble de *TextBox* pour modifier chacune des informations évoquées plus haut;
b. *QMA_ws* : permet de faire la connexion avec le Webservice pour mettre à jour les informations;
c. Un objet *TabControl* et trois objets *tabPage* pour l'organisation en onglets.

Les méthodes sont composées de :

a. Une série d'accesseurs *Set* qui permet l'affichage d'informations;

b. *upload_settings* : appelée pour mettre à jour les informations modifiées sur le serveur.

4.2.5 Contrôle utilisateur StatsControl

Ce contrôle utilisateur, instancié dans *DisplayControl*, a pour objectif l'affichage des statistiques liées au trafic transitant par un tunnel. Il n'a donc de sens que pour un tunnel opérationnel. Il présente quatre types de données : le nombre d'octets, le nombre de paquets, la bande passante estimée en octets par seconde et en paquets par seconde. Il s'organise en cinq onglets : quatre représentant chacun un type de données sur un graphe, le cinquième affichant un résumé des valeurs les plus récentes. Les graphes ont été réalisés à l'aide de la bibliothèque ZedGraph [70]. La structure de StatsControl est présentée figure 36.

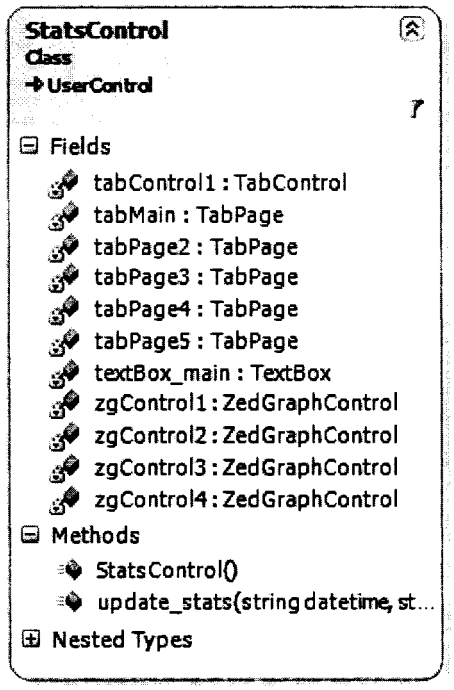

Figure 36 Structure de StatsControl

Ses membres sont les suivants :

a. Un *TabControl* et quatre *TabPage* pour la disposition en onglets;
b. Un champ *TextBox* pour l'affichage du résumé des informations dans le premier onglet;
c. Quatre objets *ZedGraphControl* pour les graphes.

Ses méthodes sont composées de :

a. Un constructeur;
b. *update_stats* : met à jour les statistiques en fournissant comme arguments :
    a. La date et l'heure de la dernière mesure;
    b. Les dernières mesures de chaque type de données (trafic en octets, en paquets, bande passante en octets par seconde, bande passante en paquets par seconde);
    c. Les séries de mesures de chaque type de données (en abscisse : la date et l'heure de la mesure ; en ordonnée : la valeur de la mesure).

4.2.6 Contrôle utilisateur QMA

Ce contrôle utilisateur est le composant principal de l'applet. C'est lui qui inclut tous les autres et qui les coordonne. Il s'organise en onglets, chacun contenant un des contrôles utilisateurs listés précédemment, à l'exception de *StatsControl* instancié dans *DisplayControl*. La structure du contrôle utilisateur QMA est présentée figure 37.

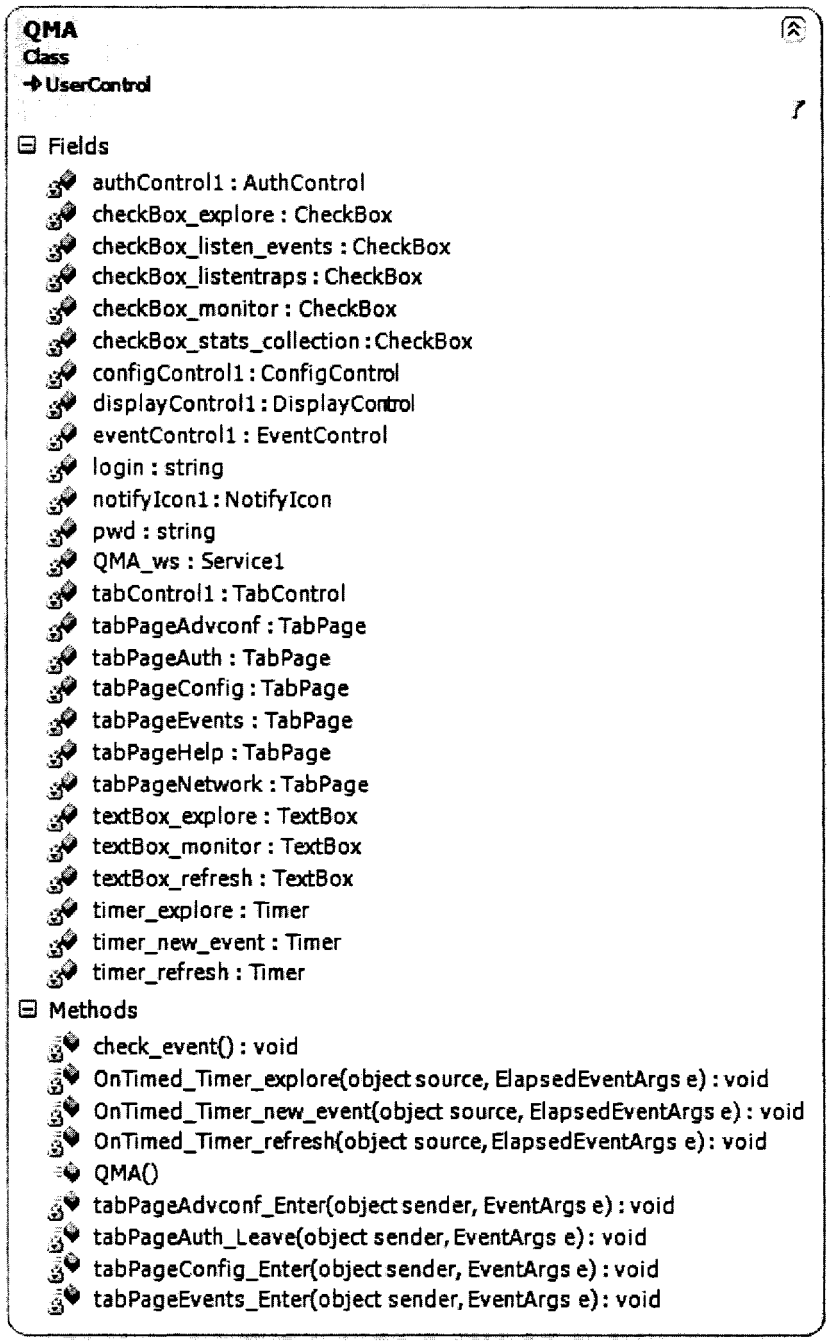

Figure 37 Structure de QMA

Ses membres sont les suivants :

a. Les contrôles utilisateurs expliqués précédemment, à savoir : *authControl1*, *configControl1*, *displayControl1*, et *eventControl1*;
b. Les *Checkbox* et *Textbox* nécessaires à la configuration. Deux types sont proposés :
    a. Configuration locale au niveau de l'applet :
        i. Possibilité de rafraîchir périodiquement la base de données (*check_Box_explore*) en définissant la fréquence (*textBox_explore*);
        ii. Possibilité de vérifier périodiquement si de nouveaux évènements ont eu lieu (*check_Box_listen_events*) en spécificiant la fréquence (*textBox_refresh*).
    b. Configuration de la surveillance du réseau au niveau du serveur :
        i. Possibilité de faire de la surveillance (*checkBox_monitor*);
        ii. Possibilité d'écouter les *traps* (*checkBox_listentraps*);
        iii. Possibilité de collecter des statistiques sur le trafic des tunnels (*checkBox_stats_collection*) en précisant la fréquence (*textBox_monitor*).
c. Les valeurs *login* et *pwd* obtenues du contrôle utilisateur *AuthControl* qui servent à l'authentification dans toute requête vers le Webservice. (Ces valeurs sont d'ailleurs redonnées aux contrôles utilisateur ayant besoin d'interaction avec le serveur, en l'occurrence *DisplayControl* et *ConfigControl*);
d. *notifyIcon1* : icône dans la barre de notifications utilisée pour signaler un nouvel évènement;
e. *QMA_ws* : permet de faire la connexion avec le Webservice pour mettre à jour les informations;
f. Un objet *TabControl* et six objets *TabPage* pour la répartition en onglets;
g. Trois minuteurs pour les opérations périodiques : *timer_explore* pour les rafraîchissements de base de données, *timer_new_event* pour la vérification de

nouveaux évènements et *timer_refresh* pour signaler au serveur la présence d'une applet connectée. Comme expliqué précédemment avec la configuration, les deux premiers minuteurs ont une fréquence paramétrable, le troisième a eu une fréquence de 120 secondes.

Les méthodes sont les suivantes :

a. *check_event* : vérifie la présence d'un nouvel évènement et le cas échéant le signale par l'icône de notification;
b. *OnTimer_Timer_explore* : évènement déclenché lorsque le minuteur *timer_explore* arrive à 0 ; il lance un rafraîchissement de la base de données;
c. *OnTimer_Timer_new_event* : évènement déclenché lorsque le minuteur *timer_new_event* arrive à 0 : démarre *check_event*;
d. *OnTimer_Timer_refresh* : évènement déclenché lorsque le minuteur *timer_refresh* arrive à 0 : signale sa présente au Webservice;
e. *QMA* : est le constructeur;
f. *tabPageAdvconf_Enter :* cette méthode est appelée quand l'utilisateur entre dans l'onglet de configuration avancée, où est instancié le contrôle utilisateur *ConfigControl* : les paramètres modifiables par ce dernier (accès la base de données et aux routeurs par SNMP) sont alors actualisés;
g. *tabPageAuth_Leave* : cette méthode est appelée quand l'utilisateur quitte l'onglet d'authentification : une vérification du nom d'utilisateur et du mot de passe rentré est alors effectuée. En cas d'échec, l'usager est prévenu et bloqué sur l'onglet d'authentification;
h. *tabPageConfig_Enter* : cette méthode est appelée quand l'utilisateur entre dans l'onglet de configuration : ses différents paramètres, présentés plus haut sont alors actualisés;
i. *tabPageEvents_Enter* : cette méthode est appelée quand l'utilisateur entre dans l'onglet des évènements, où est instancié le contrôle utilisateur *EventControl* : la liste d'évènements est alors actualisée.

4.2.7 Exécution de l'Applet

Pour que l'applet fonctionne dans une page web, il suffit de commander son exécution directement dans le code de la page, en utilisant la balise *<object>*, par exemple : *<object id="ctlDotNET" classid="AppletQMA.dll#Applet.QMA"> </object>* où *classid* permet de spécifier la dll à exécuter suivi de l'espace de nom (Applet) et de la classe à instancier (QMA). Un fichier de configuration Web est aussi nécessaire (*Web.Config*) ; celui-ci est automatiquement rempli lors de la compilation par Visual Studio.

4.3 WebService

Le Webservice fait le lien entre l'applet et les modules. Il permet de fournir des informations aux utilisateurs, tout en cachant la logique interne pour des raisons de simplicité et de sécurité. Suivant les informations souhaitées par l'applet, il appelle les classes des modules appropriés. Pour éviter toute autorisation frauduleuse, chaque service requiert une identification. Il n'y a pas à proprement parler de session entre l'applet et le Webservice. Chaque fois que celle-ci a besoin d'informations, elle envoie une requête en s'identifiant. Ainsi, il devient plus difficile d'effectuer un détournement de session. Rappelons en outre que la communication applet-Webservice se fait avec le protocole HTTPS, pour plus de confidentialité.

La structure de ce composant est présentée figure 38.

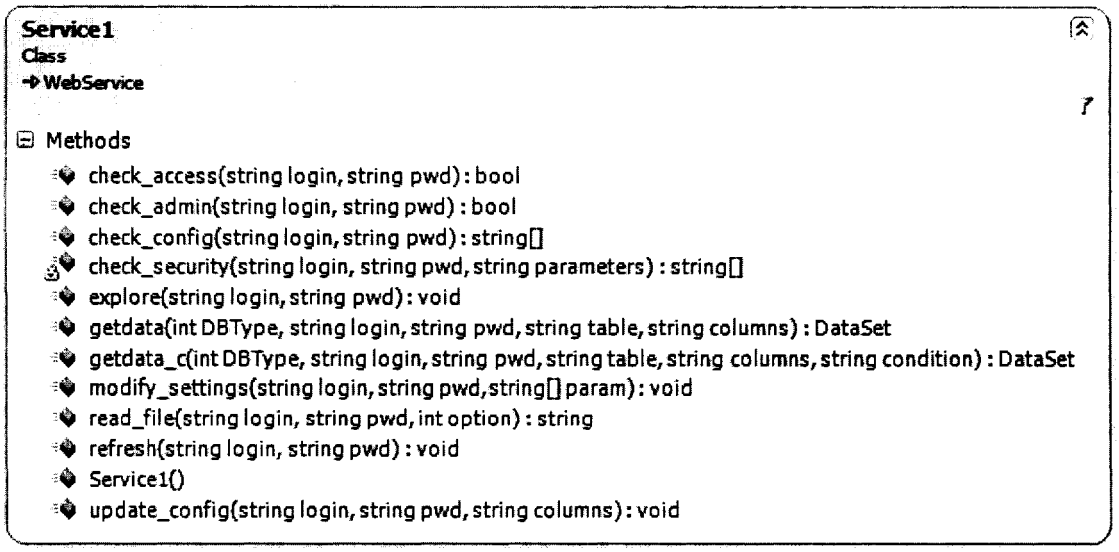

Figure 38 Structure du Webservice

Ses méthodes sont les suivantes :

a. *check_access (string login, string pwd)* : vérifie si les crédits entrés (nom d'utilisateur et mot de passe) sont corrects;

b. *check_admin (string login, string pwd)* : vérifie si l'usager identifié par son nom d'utilisateur et son mot de passe est un administrateur;

c. *check_config(string login, string pwd)* : retourne les informations nécessaires à la connexion à la base de données (nom du serveur, nom des bases, nom d'utilisateur et mot de passe) et à la communication SNMP avec les routeurs (nom d'utilisateur et mot de passe). Cette méthode est utilisée pour visualiser ces paramètres, mais ils ne servent pas en tant que tels à l'applet;

d. *explore(string login, string pwd)* : méthode qui démarre un rafraîchissement de la base de données par un *Thread*;

e. *getdata(int DBType, string login, string pwd, string table, string columns)* : méthode qui interroge la base de données. Outre les paramètres d'authentification, les arguments sont le type de base de données (*DBType*) pour préciser s'il s'agit d'une table de statistiques ou non (cf. annexe 3), ainsi que le nom de la table interrogée et les colonnes recherchées. Il est possible d'effectuer

des requêtes sur plusieurs tables à l'aide de jointures. La méthode retourne est un *DataSet*, objet s'apparentant à un tableau;

f. *getdata_c(int DBType, string login, string pwd, string table, string columns, string condition)*: même méthode que précédemment, mais en précisant une condition dans la requête;

g. *modify_settings(string login, string pwd, string [] param)* : permet de modifier les paramètres du serveur évoqués dans la méthode *check_config*;

h. *read_file(string login, string pwd, int option)* : retourne le contenu du fichier de *logs* de QMA si l'option est à 0, et du fichier de *traps* si l'option est à 1. Sinon un champ vide est renvoyé. Pour des raisons de sécurité évidentes, le nom du fichier reste inconnu à l'utilisateur, ce qui l'empêche en outre d'utiliser cette fonction pour lire n'importe quel fichier;

i. *refresh(string login, string pwd)* : permet à l'applet de signaler au serveur sa présence. Cette fonction est donc périodiquement appelée;

j. *Service1* : constructeur;

k. *update_config(string login, string pwd, string columns)* : cette fonction sert à mettre à jour les paramètres de configuration de QMA relatifs à la surveillance (écouter les *traps*, collecter les statistiques...) évoqués dans l'onglet QMA de l'applet.

4.4 Modules

Les modules QMA sont une bibliothèque de classes, réparties selon leur but :

a. *AuthManagement* : module en charge de l'authentification et plus généralement de la gestion des comptes usagers, et contient en outre les informations d'accès à la base de données et le nom d'utilisateur/mot de passe nécessaire à la communication avec les routeurs;

b. *DbManagement* : module en charge de la gestion de la base de données et de la recherche d'informations dans le réseau;

c. *MonitorManagement* : module en charge de la surveillance du réseau;
d. *LogManagement* : module en charge de consigner des évènements ou informations dans des fichiers donnés.

Notons que ces différents modules ne sont pas nécessairement indépendants. Par exemple, les fonctions d'authentification requièrent l'accès à la base de données donc des classes de *DbManagement*. Pour une meilleure compréhension de ce composant majeur de QMA, le lecteur est invité à se reporter à l'annexe 4 qui présente l'organisation de la base de données.

À présent, examinons chacun des modules.

4.4.1 AuthManagement

La protection de l'outil QMA, en plus de celle fournie par un serveur Web sécurisé, se réalise par l'utilisation de noms d'utilisateur/mots de passe stockés dans une base de données. Pour empêcher leur lecture directe, les mots de passe ne sont pas écrits directement, mais sous forme hachée avec un sel généré aléatoirement. L'algorithme de hachage utilisé est le *Secure Hash Algorithm* (SHA) 256. Plus précisément, la valeur stockée dans la base de données est le mot de passe haché avec le sel suivi du sel lui-même. Sans connaître ce sel, il serait impossible de vérifier la valeur hachée…

Ce module, en charge de l'authentification, et plus généralement de la gestion des usagers, se compose d'une classe : *Control_Auth,* dont la structure est présentée figure 39. Les méthodes relatives au hachage ont été inspirées par [71].

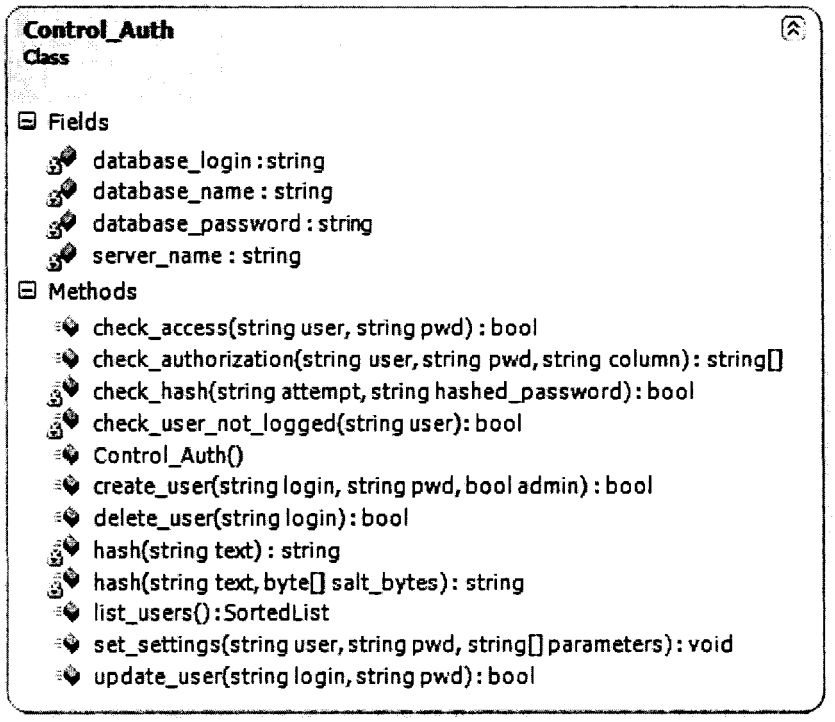

Figure 39 Structure de Control_Auth

Ses membres sont les paramètres nécessaires à la connexion à la base de données où se trouvent les tables d'authentification (cf. annexe 3) : nom du serveur, nom de la base de données et nom d'utilisateur/mot de passe.

Ses méthodes sont les suivantes :

a. *check_access(string user, string pwd)* : vérifie que le couple *user/password* fourni correspond bien à une entrée dans la base de données en appelant *check_hash*. Dans l'affirmative, cette méthode signale la date et l'heure à laquelle elle a été appelée pour signaler que l'utilisateur est connecté;
b. *check_authorization(string user, string pwd, string column)* : authentifie l'usager et retourne les paramètres de configurations demandés. Si l'authentification est un succès, cette méthode signale la date et l'heure à laquelle elle a été appelée pour signaler que l'utilisateur est connecté;

c. *check_hash(string attempt, string hashed_password)*: vérifie si le mot de passe entré en clair correspond à la valeur hachée. Pour cela, le sel est extrait de la valeur hachée stockée dans la table et un hachage est effectué sur la valeur *attempt* avec ce sel. Une comparaison est ensuite effectuée;

d. *check_user_not_logged(string user)* : examine si un utilisateur (dont le nom est fourni en argument) est actuellement connecté ou non. L'idée est de regarder la dernière fois que l'utilisateur a été authentifié (renseigné par *check_access* ou *check_authorization*), en sachant que l'applet signifie automatiquement sa présence toutes les 120 secondes. Pour plus de précaution, cette méthode calcule donc si la dernière authentification de l'utilisateur dépasse 180 secondes. Si oui, il est considéré déconnecté. Dans le cas contraire, il est considéré comme toujours connecté;

e. *Control_Auth* : constructeur;

f. *create_user(string login, string pwd, bool admin)* : crée un nouvel usager en ajoutant une entrée dans la base de données : nom d'utilisateur (en vérifiant qu'il n'existe pas déjà), mot de passe haché avec la méthode *hash* et valeur booléenne précisant si l'usager est administrateur ou non;

g. *delete_user(string login)* : supprime un compte, à condition que l'usager ne soit pas actuellement connecté, ce que la méthode *check_user_not_logged* permet de savoir;

h. *hash (string text)* : génère un sel aléatoire, puis hache *text* avec ce sel;

i. *hash(string text, byte[] salt_bytes)* : hache *text* avec le sel fourni sous forme d'octets;

j. *list_users* : retourne une liste contenant les noms d'utilisateur;

k. *set_settings(string user, string pwd, string [] parameters)* : modifie les informations nécessaires à la connexion à la base de données (nom du serveur, nom des bases, nom d'utilisateur et mot de passe) et à la communication SNMP avec les routeurs (nom d'utilisateur et mot de passe);

l. *update_user(string login, string pwd)* : met à jour le mot de passe d'un utilisateur, à condition que l'usager ne soit pas actuellement connecté, ce que la méthode *check_user_not_logged* permet de savoir;

4.4.2 DbManagement

Ce module a en charge l'interaction avec le réseau et la base de données. Il est composé de trois classes :

a. *Snmp* : classe en charge d'effectuer les requêtes SNMP pour collecter les informations. Dans QMA, le client SNMP utilisé est la librairie de fonctions Net-SNMP [72]. Les requêtes utilisées sont SNMPGET [4] pour obtenir un objet en particulier, sinon SNMPGETBULK [5] (appelée par l'exécutable *snmpbulkwalk* de la librairie). La seconde requête, nouveauté de SNMPv2, permet d'obtenir à l'aide d'une requête plusieurs objets consécutifs. Ainsi, pour obtenir la liste de 10 segments, il ne faut qu'une requête SNMPGETBULK pour 10 requêtes SNMP. Le trafic généré est largement réduit, et les performances significativement améliorées;
b. *Database* : classe en charge de la communication avec la base de données;
c. *Control_DB* : classe qui gère les deux classes précédentes pour collecter l'information du réseau et la stocker dans la base.

4.4.2.1 Classe SNMP

La structure de la classe *Snmp* est décrite figure 40. Elle implémente des méthodes adaptées aux différents types d'objets dans la MIB.

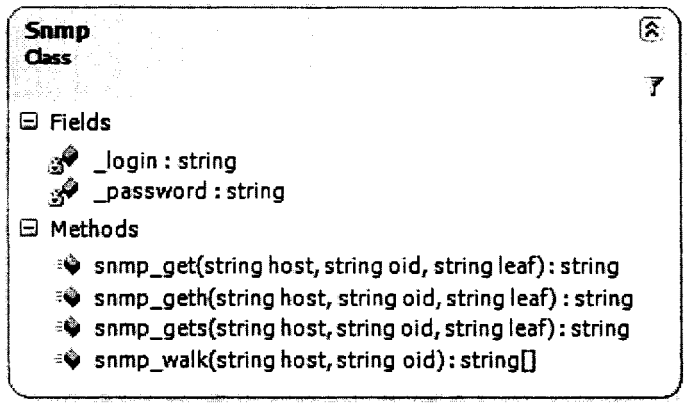

Figure 40 Structure de *Snmp*

Ses paramètres sont les informations nécessaires pour interroger un routeur avec SNMPv3 : un nom d'utilisateur et un mot de passe.

Les méthodes sont les suivantes :

a. *snmp_get(string host, string oid, string leaf)* : effectue une requête SNMPGET sur un routeur donné (*host*), en précisant l'*Object Identifier* (OID) dans la MIB et la feuille concernée (*leaf*). Seule la valeur est retournée, sans le type de données. Par exemple si le résultat de la requête est *INTEGER: 2*, cette méthode renvoie simplement l'information nécessaire, ici *2*;
b. *snmp_geth(string host, string oid, string leaf)*: à l'instar de *snmp_get*, effectue une requête SNMPGET mais sur un objet codé en hexadécimal. La valeur retournée est la conversion décimale. Par exemple, si l'objet dans la MIB est *AC 15 FD 0E*, la valeur retournée est *172.21.253.14*;
c. *snmp_geth(string host, string oid, string leaf)*: à l'instar de *snmp_get*, effectue une requête SNMPGET mais sur un objet de type *string*. Par exemple, si l'objet dans la MIB est *STRING: "Tunnel2 TE vers PE3"*, *Tunnel2 TE vers PE3* est renvoyé;
d. *snmp_walk (string host, string oid)* : appelle la fonction *snmpbulkwalk* de Net-SNMP, qui effectue une série de SNMPGETBULK, pour lister toutes les valeurs

d'une branche donnée (*oid*). Cette méthode retourne un tableau de *string* dimensionné au nombre de feuilles.

4.4.2.2 Classe Database

La classe *Database* est responsable du dialogue avec la base de données. On y retrouve les opérations classiques de sélection, insertion, mise à jour et suppression. Sa structure est illustrée figure 41.

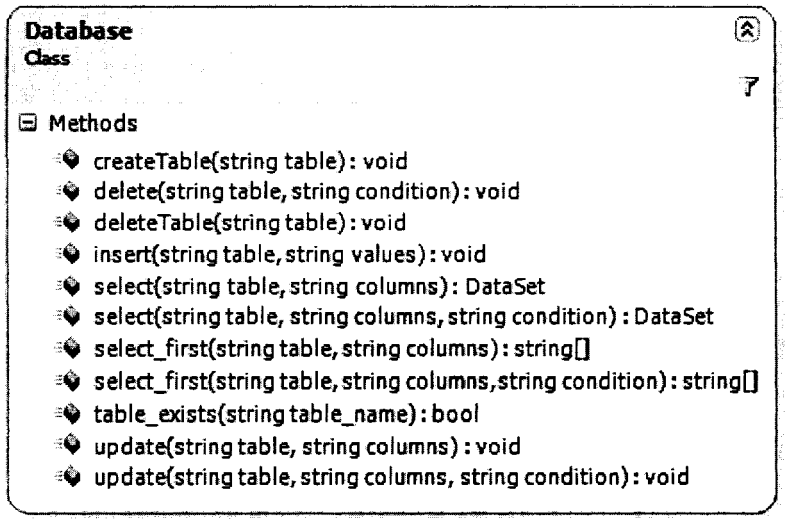

Figure 41 Structure de *Database*

Ses méthodes sont les suivantes :

a. *createTable(string table)* : crée une nouvelle table ; utile pour les statistiques où les tables dépendent de la configuration du réseau;
b. *delete(string table, string condition)* : supprime des lignes dans une table selon une condition donnée;
c. *deleteTable(string table)* : supprime une table;
d. *insert(string table, string value)* : ajoute une entrée dans une table;
e. *select(string table, string columns)* : retourne sous forme de *DataSet* les colonnes demandées d'une table donnée;

f. *select(string table, string columns, string condition)* : même opération que précédemment mais selon certaines conditions;
g. *select_first(string table, string columns)* : retourne la première ligne d'une table sous forme de tableau contenant les champs spécifiés en colonne. Cette méthode est utile si on sait que la ligne recherchée est unique. On n'a alors pas à s'encombrer d'un *DataSet*;
h. *select_first(string table, string columns, string condition)* : même fonction que précédemment, mais avec des conditions;
i. *table_exists(string table_name)* : renvoie un booléen indiquant si la table existe ou non;
j. *update(string table, string columns)* : met à jour les champs précisés dans une table;
k. *update(string table, string columns, string condition)* : même opération que précédemment en précisant une condition.

4.4.2.3 Classe Control_DB

Si les classes *Database* et *Snmp* sont principalement des interfaces vers la base de données et le réseau, la classe *Control_DB* implémente davantage la logique de l'outil. Elle est donc beaucoup plus complexe que les classes précédentes. Sa structure est présentée figure 42. Un point important à mentionner est l'utilisation de tables duales pour les informations susceptibles d'être modifiées fréquemment. Ceci est davantage expliqué dans l'annexe 3. L'idée est juste de permettre aux utilisateurs de consulter la base de données alors que celle-ci se met à jour : pour cela plusieurs tables sont disponibles en double (exemple : *LSP0* et *LSP1*), l'une servant à la consultation et l'autre à la mise à jour, leur rôle changeant tour à tour. Les MIB utilisées sont les MIB standards MPLS définies dans les RFC [28, 29, 73] explicitées dans [74].

Les membres de *Control_DB* sont les suivants :

a. *_database* : objet *Database* pour l'interface avec la base de données;

b. *_snmp* : objet *Snmp* pour l'interface avec le réseau;

c. *_table_number* : numéro des tables disponibles en consultation.

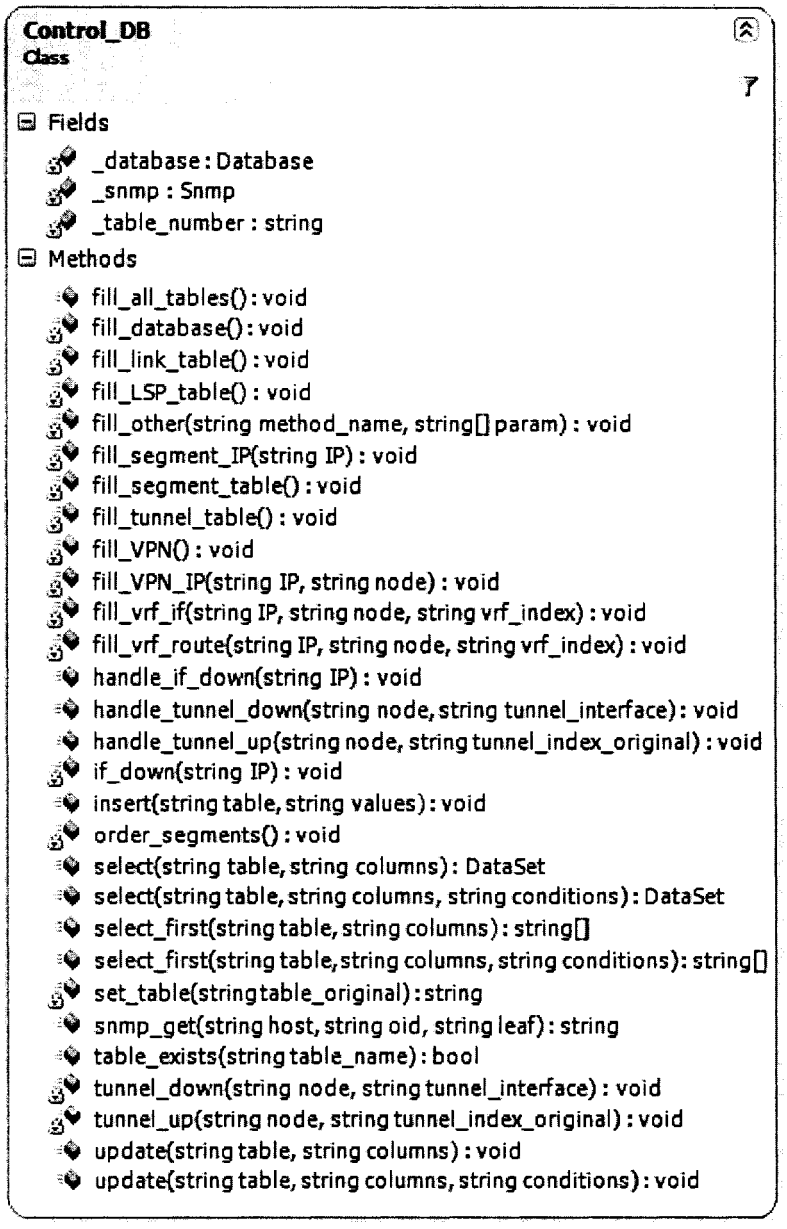

Figure 42 Structure de *Control_DB*

Ses méthodes sont composées de :

- *fill_all_tables* : cette méthode et *fill_other* constituent les deux méthodes de contrôle d'accès à l'écriture de la base de données. La problématique est la suivante : pour pouvoir permettre la consultation de la base de données pendant que celle-ci est mise à jour, plusieurs tables sont doublées ; mais que se passe-t-il quand plusieurs fonctions veulent mettre à jour la base simultanément? Par exemple, une *trap* signale un tunnel qui vient de tomber et un utilisateur clique sur *explore* de l'applet pour mettre à jour toute l'information sur le réseau. Deux fonctions écrivant dans la table simultanément conduiraient celle-ci dans un état instable, empêchant QMA de fonctionner normalement. Il faut donc un algorithme de gestion des méthodes qui veulent écrire dans la base. Pour cela, il convient de noter qu'il y a deux types de méthodes modifiant la base de données : celle qui rafraîchit toute l'information (*fill_database*) et les autres qui agissent sur un composant précis du réseau (*if_down*, *tunnel_down*, *tunnel_up*). Rafraîchir tout le réseau inclut de facto toutes les autres opérations, donc l'algorithme d'accès à l'écriture de la base de données doit être différent selon ce que la méthode veut mettre à jour tout ou partie de l'information du réseau. *fill_all_table* est la méthode gérant l'accès en écriture aux tables à *fill_database* selon l'algorithme illustré figure 43 sous forme *Specification and Design Language* (SDL).

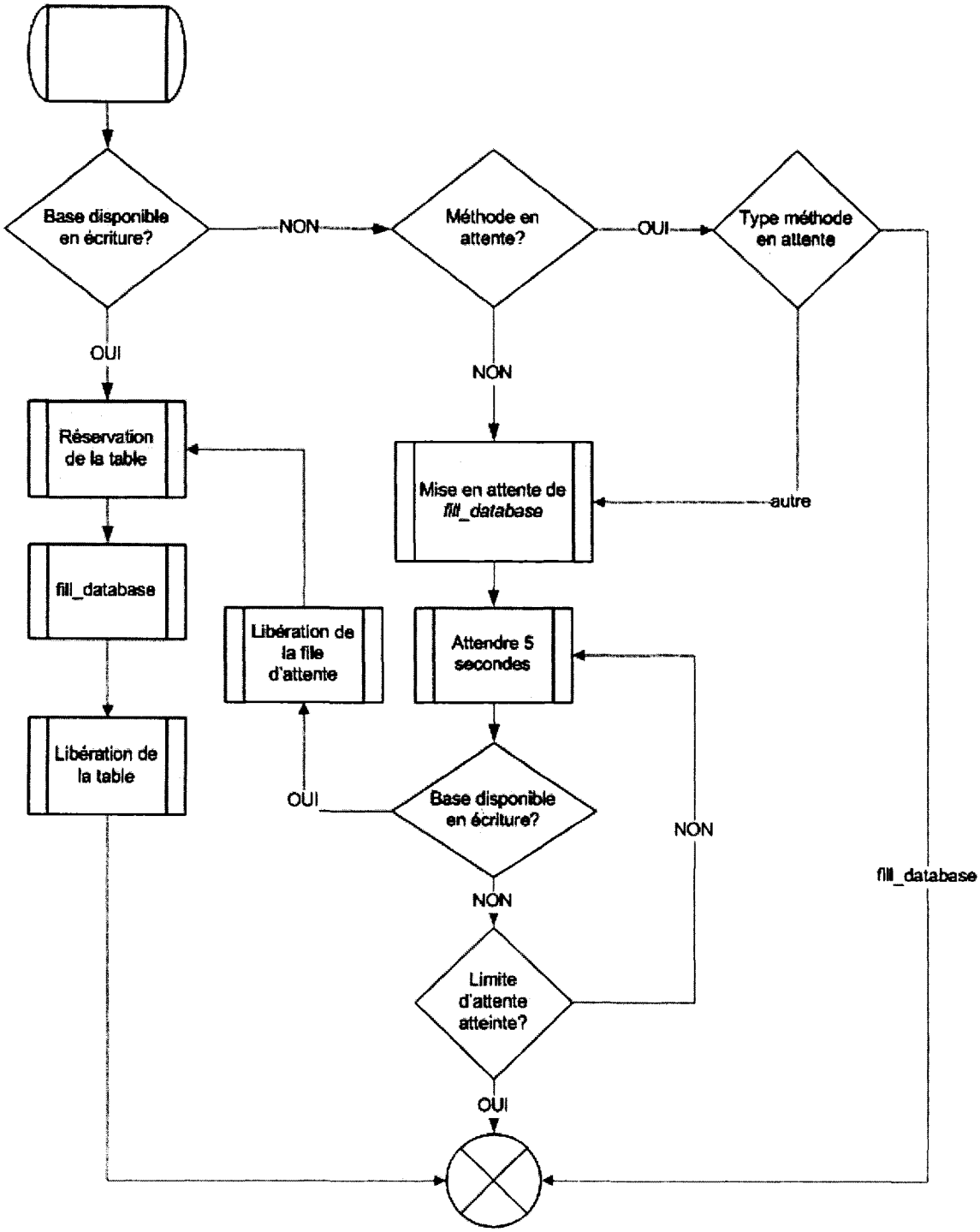

Figure 43 Algorithme SDL de *fill_all_tables*

Le principe est le suivant : on regarde si la table est libre en écriture :

- si oui l'action est exécutée.
- sinon, on regarde si une méthode est en attente :
  - s'il s'agit déjà de *fill_database*, on s'arrête : un processus est déjà en attente d'exécuter la fonction, il est inutile d'attendre.
  - sinon qu'il y ait une fonction en attente ou non, *fill_database* est réservée (elle inclut les méthodes de modification locale). Puis on essaie régulièrement (toutes les cinq secondes) jusqu'à ce qu'un nombre maximum d'essais soit atteint. Cette limite empêche le phénomène de famine.

- *fill_database* : rafraîchit l'ensemble des informations sur le réseau. C'est elle qui est ultimement appelée par la méthode *explore* du Webservice ou de l'applet. Elle vide tout d'abord les tables contenant les informations des LSP, Tunnels et VRF pour les remplir à l'aide de *fill_link_table*, *fill_segment_table*, *fill_LSP_table*, *fill_tunnel_table*, *order_table* et *fill_VPN* explicitées ci-dessous.
- *fill_link_table* : met à jour la bande passante des liens en utilisant l'objet *ifSpeed* de la MIB *Interfaces*.
- *fill_LSP_table* : remplit la table LSP : elle est appelée une fois que les segments sont tous fournis et regroupés en LSP. Alors, pour chaque LSP, cette méthode cherche l'*ingress* et l'*egress* et insère une entrée dans la table LSP.
- *fill_other(string method_name, string[] param)* : à l'instar de *fill_all_tables*, cette méthode gère l'accès en écriture à la base, mais pour les fonctions (*if_down, tunnel_down, tunnel_up*). Le nom de la fonction à exécuter, ainsi que ses arguments sont respectivement fournis dans *method_name* et *param*. L'algorithme de cette méthode, présenté figure 44, est légèrement différent de celui de *fill_all_tables*, puisque cette dernière inclut toutes les autres modifications.

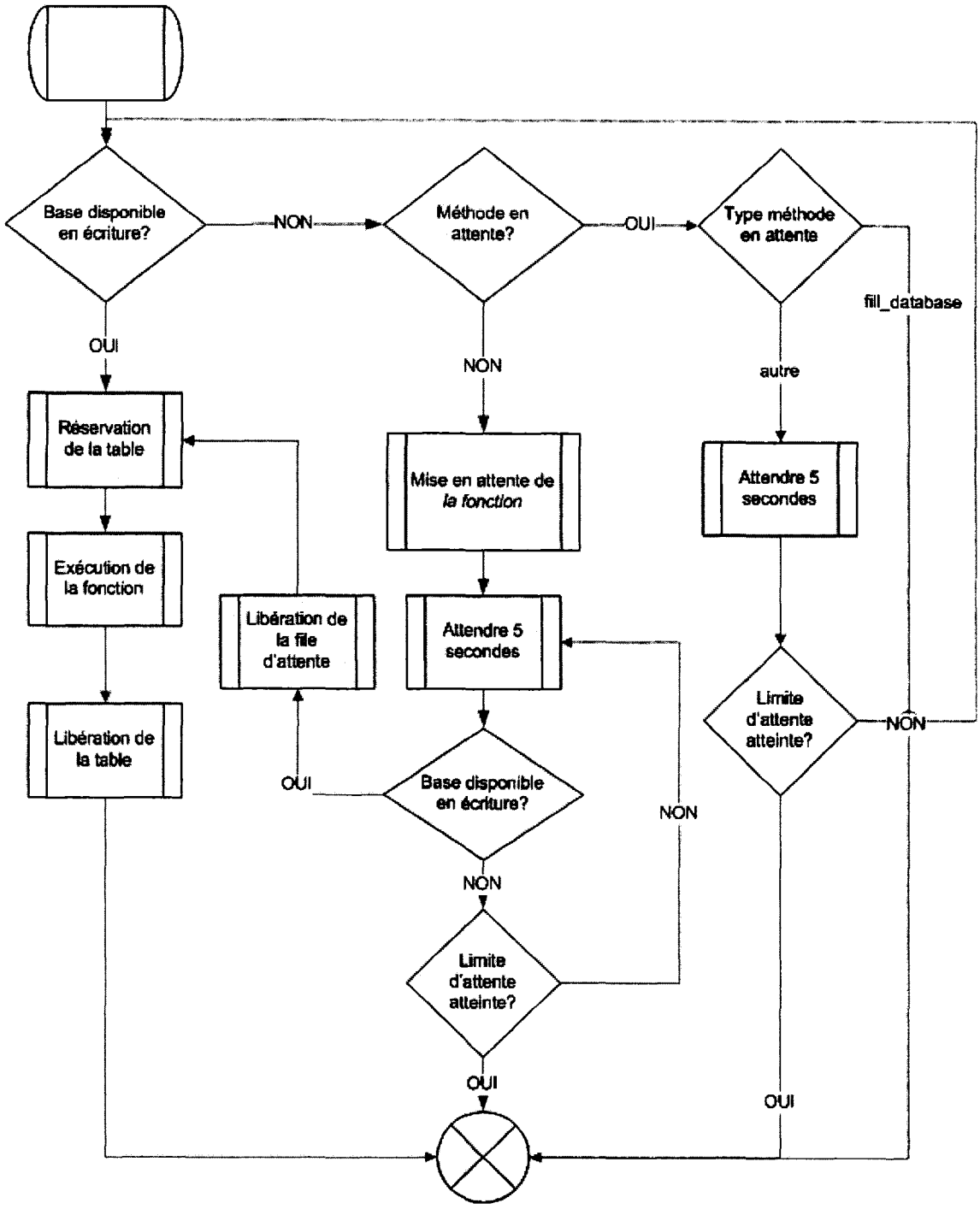

Figure 44 Algorithme SDL de *fill_other*

Le principe est le suivant : on regarde si la table est libre en écriture :

- si oui l'action est exécutée.
- sinon, on regarde si une méthode est en attente :
  - s'il s'agit déjà de *fill_database*, on s'arrête : cette méthode rafraîchissant tout le réseau, il est inutile d'attendre.
  - sinon, on attend cinq secondes et on essaie régulièrement depuis le début, jusqu'à ce qu'un nombre maximum d'essais soit atteint. Cette limite empêche le phénomène de famine.

- *fill_segment_IP(string IP)* : obtient d'un routeur la liste de ses segments. Pour bien comprendre cette fonction, rappelons-nous que du point de vue du routeur, le LSP est constitué d'un segment entrant et d'un segment sortant (sauf pour les routeurs *ingress* ou *egress*). En pratique, un segment sortant peut en fait être utilisé par plusieurs LSP comme le montre la figure 45 : les paquets entrant dans P3 avec un label 10 sortent avec un « label POP » sur l'interface 3. Ainsi il y a un segment sortant, mais deux LSP y transitent.

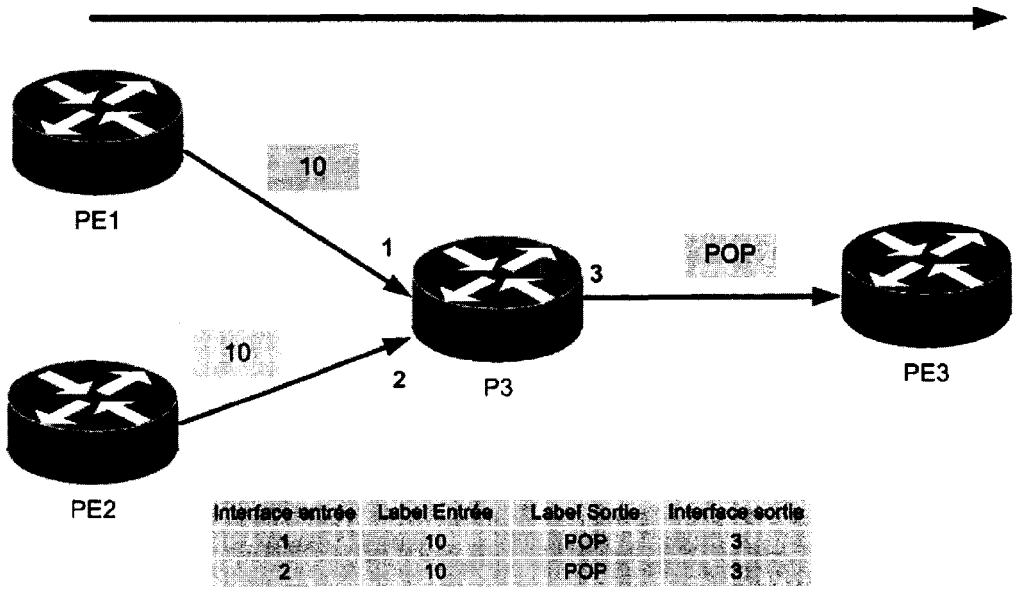

Figure 45 Segments entrants et sortants d'un routeur

Dans la MIB *MPLS-LSR* [29] les segments entrants et sortants sont listés dans des tables séparément. La correspondance se fait au travers d'un index partagé, *XCIndex*. Le principe de *fill_segment_IP* est d'une part de lister tous les segments entrants avec l'objet *mplsInSegmentXCIndex* de la MIB et d'autre part les segments sortants avec les objets *mplsOutSegmentNextHopIpv4Addr* (adresse du prochain saut) et *mplsOutSegmentTopLabel* (label du segment sortant). Pour chacun de ses segments, l'index de l'interface entrante ou sortante est examiné pour déterminer si celle-ci est considérée par QMA (selon les informations de la base de données), ce qui permet de filtrer les segments qui sortent du champ du réseau traité par l'outil.

Ensuite, pour chaque segment sortant, on cherche les segments entrants correspondant, et pour chaque segment entrant trouvé, une entrée est ajoutée dans la table des Segments. Dans le cas de la figure 45, deux segments sont ajoutés pour P3 : le premier entrant par l'interface 1 avec un label 10 et sortant par l'interface 3 avec un « label POP », le second entrant par l'interface 2 avec les autres propriétés similaires. Si pour un segment sortant, aucun segment entrant n'est trouvé, il est tout de même ajouté à la base : il s'agit alors du premier segment d'un LSP.

- *fill_segment* : liste l'ensemble des segments du réseau. D'abord routeur par routeur, avec la fonction *fill_segment_IP*. Ensuite, un assemblage est fait pour regrouper les segments par LSP.
- Pour comprendre l'algorithme d'assemblage, il convient là encore d'apporter deux précisions. *fill_segment_IP* ajoute dans la table Segment, pour chaque segment sortant autant d'entrées que le nombre de segments sortants, ou au moins une s'il n'y a pas de segments entrants. Cependant dans certains cas, certaines entrées ne sont pas nécessairement utilisées. Dans le cas de la figure 45, rien ne garantit en effet par exemple que PE1 ait effectivement un segment sortant vers P3 avec un label 10. D'autre part, un couple segment entrant-

segment sortant peut même être utilisé par plus qu'un LSP, comme le montre la figure 46.

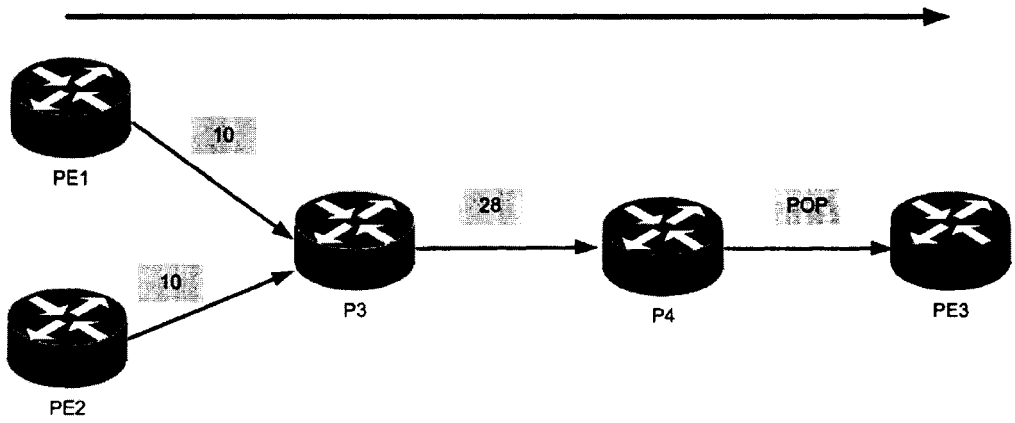

Figure 46 Deux LSP pour un segment

Au niveau du routeur P4, il n'y a qu'un couple segment entrant/sortant, mais deux LSP qui y passent. Cependant P4 ne peut avoir cette information, il faut une vue globale du réseau pour cela.

Forts de ces observations, expliquons l'algorithme de regroupement des segments par LSP ; l'idée est d'avoir pour une entrée de la table *Segment* un seul LSP qui y transite.

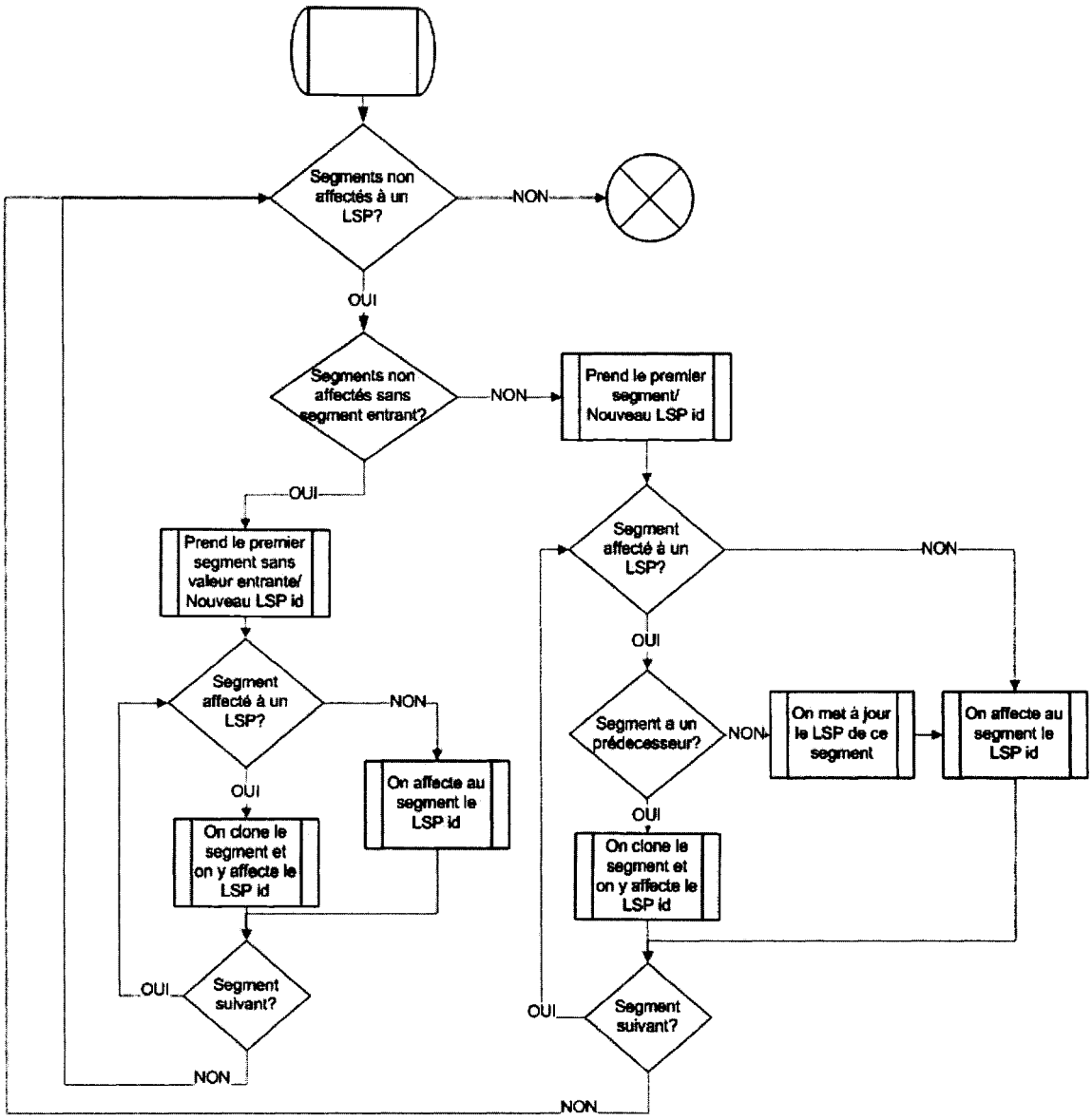

Figure 47 Algorithme de regroupement des segments par LSP

Tant qu'il reste un segment qui n'est pas dans un LSP :

- On cherche premièrement les segments sortants sans segments entrants :
  - On regarde si le segment est déjà dans un LSP : si oui, on clone le segment pour ne pas écraser l'autre LSP ; sinon on affecte au segment le nouvel LSP id.

        - On cherche s'il y a un segment suivant et on réitère l'opération.
    - Si les segments sortants sans segments entrants sont déjà tous affectés à un LSP, on prend le premier segment sans LSP :
        - On regarde si le segment est déjà dans un LSP :
            - Si oui on regarde s'il a un prédécesseur :
                - Si oui on clone le segment
                - Si non, il s'agit d'un segment déjà traité dont on a trouvé un segment antérieur. On met à jour le LSP.
            - Si non on met à jour le LSP.
        - On cherche le segment suivant et on réitère l'opération.

- *fill_tunnel_table* : obtient du réseau les informations concernant les tunnels MPLS TE et remplit la base de données. Les objets utilisés, de la MIB MPLS-TE [28], sont notamment les suivants, pour la plupart les propriétés des tunnels : *mplsTunnelName* (nom du tunnel), *mplsTunnelDescr* (description), *mplsTunnelSetupPrio* (priorité d'établissement), *mplsTunnelHoldingPrio* (priorité de maintien), *mplsTunnelResourceMaxRate* (bande passante réservée), *mplsTunnelIncludeAllAffinity* (affinité), *mplsTunnelAdminStatus* (statut administratif), *mplsTunnelOperStatus* (statut opérationnel), *mplsTunnelRole* (rôle du routeur dans le tunnel, pour identifier si l'on se trouve à la tête, à la queue ou au milieu du tunnel), *mplsTunnelTotalUpTime* (temps de vie du tunnel), *mplsTunnelIfIndex* (numéro d'interface du tunnel), *mplsTunnelXCPointer* (pointeur vers le segment dans la MIB MPLS-LSR). Ensuite, nous avons constaté que le premier segment du tunnel, entre l'ingress et le premier saut n'est pas présent dans la MIB. Il est néanmoins ajouté dans la base de données.
- *fill_VPN* : récupère l'ensemble des informations sur les VRF, en appelant pour chaque routeur la méthode *fill_VPN_IP*.

- *fill_VPN_IP(string IP, string node)* : obtient, pour un routeur donné, les informations relatives aux VRF avec les objets *mplsVpnVrfRouteDistinguisher* (Route Distinguisher du VRF), *mplsVpnVrfDescription* (description du VRF), *mplsVpnVrfOperStatus* (statut opérationnel), *mplsVpnVrfActiveInterfaces* (nombre d'interfaces actives), *mplsVpnVrfConfStorageType* (type de stockage), *mplsVpnVrfPerfCurrNumRoutes* (nombre de routes configurées), *mplsVpnVrfSecIllegalLabelViolations* (nombre violations de labels). Puis cette méthode appelle *fill_vrf_if* et *fill_vrf_route*.
- *fill_vrf_if(string IP, string node, string vrf_index)* : liste, pour un VRF, les différentes interfaces avec ses informations, obtenues avec les objets suivants de la MIB MPLS-VPN [73]: *mplsVpnInterfaceLabelEdgeType* (type de label de bordure), *mplsVpnInterfaceVpnClassification* (classification du VPN), *mplsVpnInterfaceConfStorageType* (type de stockage dans la mémoire), *mplsVpnInterfaceConfRowStatus* (statut opérationnel).
- *fill_vrf_route(string IP, string node, string vrf_index)* : liste, pour un VRF, les différentes interfaces avec ses informations, obtenues avec les objets suivants : *mplsVpnVrfRouteIfIndex* (index l'interface par laquelle passe la route), *mplsVpnVrfRouteType* (type de la route), *mplsVpnVrfRouteProto* (protocole utilisé pour établir la route), *mplsVpnVrfRouteAge* (âge de la route), *mplsVpnVrfRouteRowStatus* (statut opérationnel de la route), *mplsVpnVrfRouteStorageType* (type de stockage de la route).
- *handle_if_down(string IP)* : appelle la méthode *fill_other* pour démarrer *if_down*.
- *handle_tunnel_down(string node, string tunnel_interface)* : appelle la méthode *fill_other* pour démarrer *tunnel_down*.
- *handle_tunnel_up(string node, string tunnel_index_original)* : appelle la méthode *fill_other* pour démarrer *tunnel_up*.
- *if_down(string IP)* : enlève de la base tous les segments passant par l'interface dont l'IP est fournie en argument.

- *order_segments* : pour chaque LSP, numérote les segments dans l'ordre (pour permettre à l'applet d'ordonner les segments).
- *set_table(string table_original)* : comme mentionné précédemment, certaines tables sont doublées, de manière transparente à l'applet même au Webservice. Cette méthode effectue l'opération suivante selon la table fournie en argument:
    - s'il s'agit d'une table doublée, retourne le nom de la table disponible en lecture : par exemple *set_table("LSP")* retourne *LSP0* ou *LSP1*.
    - s'il s'agit d'une table non doublée, retourne le nom tel quel.
- *tunnel_down(string node, string tunnel_interface)* : appelée quand un tunnel tombe, passe un tunnel de l'état d'opérationnel à celui d'éteint dans la base de données (en supprimant notamment le LSP associé).
- *tunnel_up(string node, string tunnel_index_original)* : appelée quand un nouveau tunnel apparaît, cherche dans les MIB les informations relatives à ce nouveau tunnel, et en particulier le LSP emprunté.

Les autres méthodes sont des méthodes de la classe *Database* : en effet, le membre *_database* est privé dans *Control_DB*. Ces méthodes (*select, select_first, insert, update*) appellent donc simplement les méthodes du même nom de *Database*, mais en utilisant la fonction *set_table*. Ainsi grâce à ces méthodes, quand le *Webservice* veut sélectionner une information de la table *LSP* par exemple, il n'a pas besoin de savoir qu'il s'agit en fait de *LSP0* ou *LSP1*.

Control_DB est donc une classe complexe, centrale dans QMA.

4.4.3 MonitorManagement

Ce module a pour charge la surveillance du réseau. Il se compose de trois classes :

a. *Trap_manager* : en charge de l'écoute des *traps*;
b. *Stats_manager* : en charge de la collecte des statistiques;
c. *Control_Monitor* : classe qui gère les deux classes précédentes.

4.4.3.1 Classe Trap_manager

Trap_manager a pour rôle exclusif d'écouter les *traps* et de prendre la décision adéquate. Sa structure est décrite figure 48.

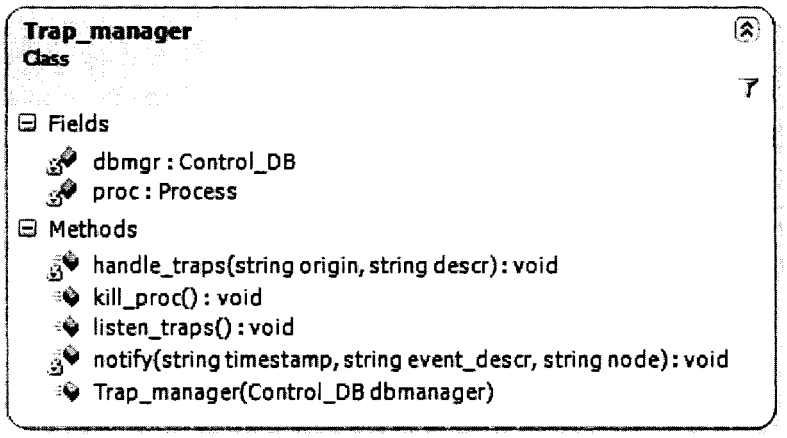

Figure 48 Structure de Trap_manager

Ses membres sont les suivants :

a. *dbmgr* : objet de type *Control_DB* pour l'interaction avec la base de données;
b. *proc* : objet de type *processus*, pour pouvoir contrôler l'écoute ou non de *traps*.

Ses méthodes se composent de :

a. *handle_traps (string origin, string descr) :* prend la décision idoine selon la *trap* reçue. Les *traps* peuvent être des *traps* standards (concernant les interfaces) ou propres à MPLS, définies dans les MIB MPLS. Ainsi :
    a. dans le cas d'un tunnel qui tombe, appelle *handle_tunnel_down* de *Control_DB;*

b. dans le cas d'un tunnel qui apparaît, appelle *handle_tunnel_up* de *Control_DB;*
c. dans le cas d'une interface qui tombe, appelle *handle_if_down* de *Control_DB;*
d. Dans le cas d'une interface qui devient opérationnelle ou de segments créés ou détruits, appelle *fill_all_tables* de *Control_DB.*

b. *kill_proc* : stoppe l'écoute des *traps*;
c. *listen_traps* : démarre l'écoute des *traps* (dans un *Thread*);
d. *notify(string timestamp, string event_descr, string node)* : répertorie un évènement dans la base de données;
e. *Trap_manager(Control_DB dbmanager)* : constructeur qui prend en argument un objet *Control_DB.*

4.4.3.2 Classe Stats_manager

Cette classe collecte les statistiques des tunnels. Pour cela, des tables dédiées à chaque tunnel sont dynamiquement créées. Sa structure est décrite figure 49.

Ses membres sont les suivants :

a. *_dbmgr* : objet *Control_DB* nécessaire pour l'interaction avec la base de données et le réseau;
b. *_dbstats* : objet *Database* pour l'accès à la base de données dédiée aux statistiques;
c. *_timer1* : objet *Timer* utilisé pour aller chercher périodiquement les statistiques dans le réseau;
d. *_tunnel_list* : liste des index des tunnels dont il faut récolter les statistiques.

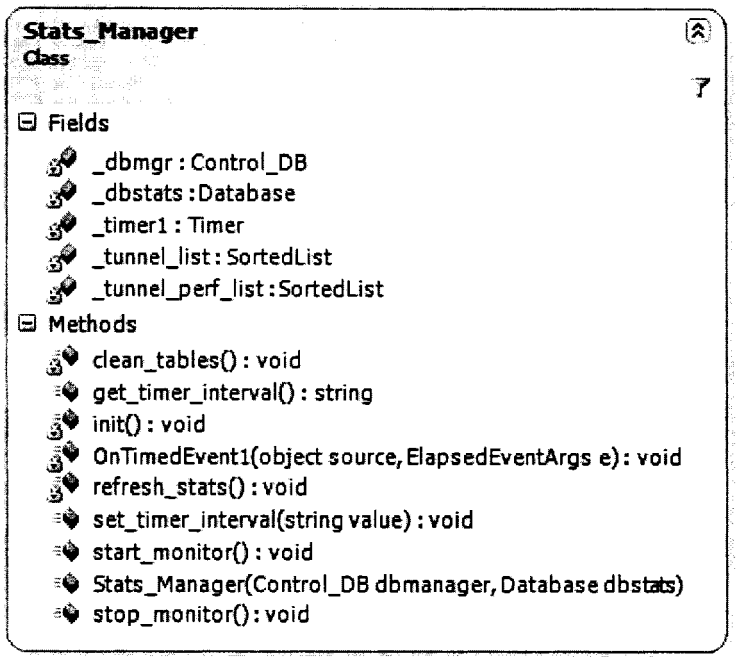

Figure 49 Structure de Stats_manager

Ses méthodes sont composées de :

a. *clean_tables* : supprime les tables des tunnels qui ne sont plus opérationnels;
b. *get_timer_interval* : retourne la valeur de l'intervalle défini pour *_timer1*;
c. *init* : crée les tables pour les tunnels opérationnels;
d. *OnTimedEvent1* : appelle *refresh_stats* quand le minuteur *_timer1* arrive à son terme;
e. *refresh_stats* : interroge le réseau pour obtenir les statistiques. Seuls les nombres d'octets et de paquets sont collectés. Les bandes passantes en octets et paquets en sont déduites d'après la dernière mesure;
f. *set_timer_interval(string value)* : modifie l'intervalle de *_timer1*;
g. *start_monitor* : démarre la collection de statistiques;
h. *stats_manager(Control_DB dbmanager, Database dbstats)* : constructeur prenant en argument un objet Control_DB et un objet Database nécessaires à son fonctionnement;
i. *stop_monitor* : arrête la collection de statistiques.

4.4.3.3 Classe Control_Monitor

La classe *Control_Monitor* est responsable de la surveillance du réseau. Celle-ci est paramétrable pour l'utilisateur. La classe coordonne donc la collection de statistiques, l'écoute des *traps*, et met à jour les paramètres quand ceux-ci sont modifiés par l'utilisateur. Sa structure est présentée figure 50.

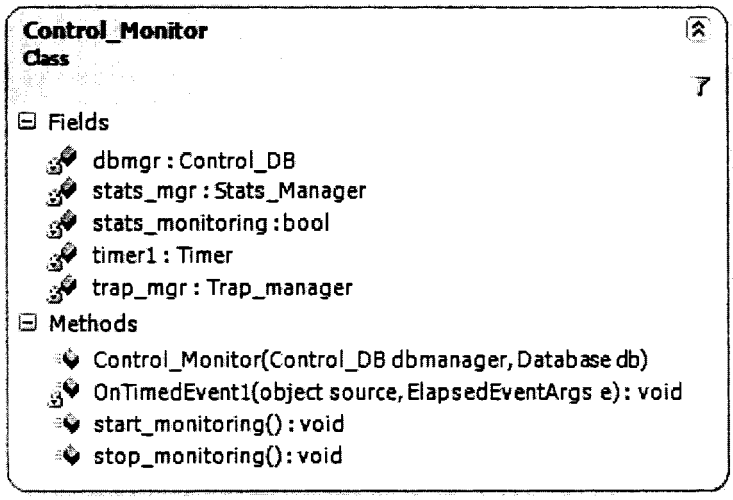

Figure 50 Structure de Stats_manager

Ses membres sont les suivants :

a. *dbmgr* : objet *Control_DB* pour l'interaction avec la base de données et le réseau.
b. *stats_mgr* : instance de *Stats_Manager* pour la collection de statistiques;
c. *Timer1* : minuteur qui sert à vérifier régulièrement la configuration;
d. *trap_mgr* : instance de *Trap_manager* qui sert à l'écoute des *traps*.

Ses méthodes sont composées de :

a. *Control_Monitor (Control_DB dbmanager, Database db)* : constructeur prenant en argument les objets *Control_DB* et *Database* nécessaires pour la classe elle-même ou à ses paramètres;

b. *OnTimedEvent1* : appelée périodiquement quand le minuteur *Timer1* arrive à son terme. Cette méthode vérifie qu'une applet est toujours connectée (ce que l'applet manifeste avec la méthode *refresh* du Webservice) sans quoi la surveillance s'arrête pour éviter d'encombrer inutilement la bande passante du réseau à collecter des statistiques. En outre, si des modifications ont été faites par l'usager, comme l'arrêt ou le lancement de l'écoute des *traps*, le changement de l'intervalle de collecte des statistiques, celles-ci sont prises en considération dans cette méthode;
c. *start_monitoring* : démarre le processus de surveillance;
d. *stop_monitoring* : arrête toute surveillance.

4.4.4 LogManagement

Ce module a pour rôle de consigner certains évènements dans des fichiers textes. Il se compose de la classe *Control_Log* dont la structure est présentée figure 51.

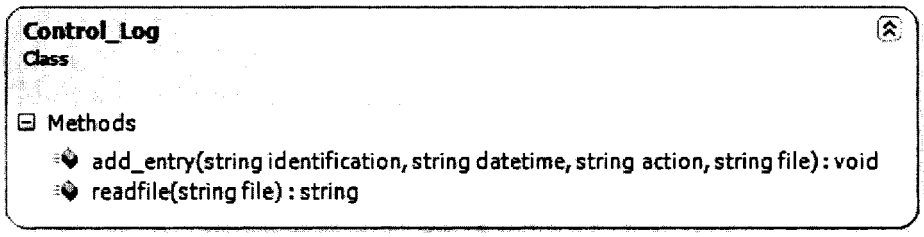

Figure 51 Structure de Control_Log

Ses méthodes sont les suivantes :

a. *add_entry(string identification, string datetime, string action, string file)* : reporte un nouvel évènement (*action*) en le plaçant au début d'un fichier donné (*file*) en précisant l'auteur du message (*identification*) et l'instant auquel celui-ci a été rempli (*datetime*);
b. *readfile(string file)* : retourne le contenu d'un fichier.

4.5 Contrôleur de serveur

Le contrôleur de serveur est une application en charge d'opérations particulières. Située sur le serveur dont l'accès physique est supposément suffisamment protégé, elle ne doit être utilisée que par l'administrateur de l'outil. Ainsi, la mesure d'authentification n'est pas prise en compte comme pour l'applet : si une personne peut y accéder, cela signifie qu'elle dispose déjà d'un accès physique au serveur, ce qui permet en soi de compromettre la sécurité de QMA.

Le contrôleur de serveur s'organise sous forme d'onglets contenant chacun un contrôle utilisateur, à l'instar de l'applet. Outre les fonctions liées à la QoS, non traitées ici, le contrôleur de serveur permet la configuration de la surveillance et la gestion des usagers.

4.5.1 Onglet MPLS

L'onglet MPLS est composé du contrôle utilisateur *MPLSControl*, dont la structure est illustrée figure 52.

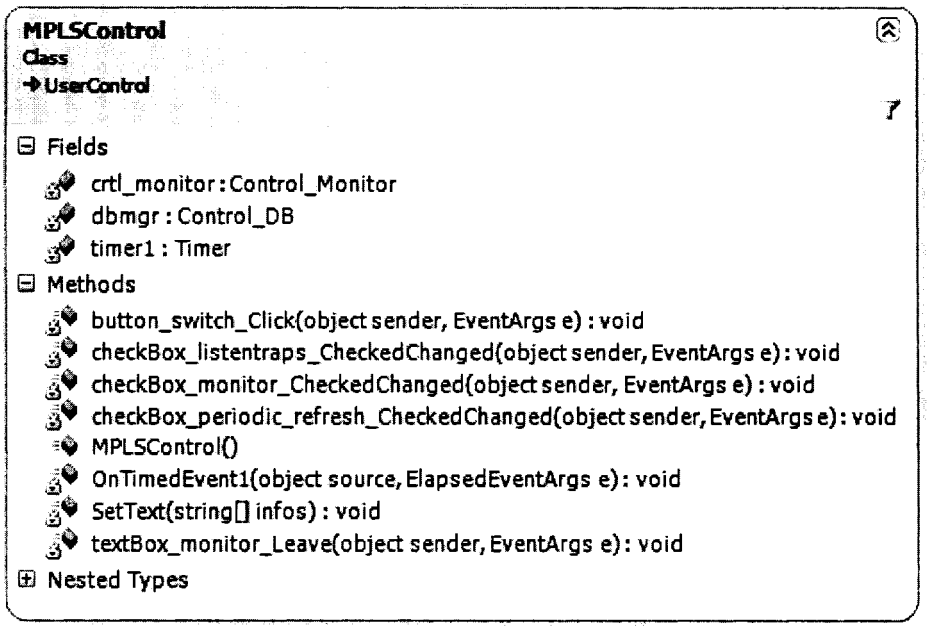

Figure 52 Structure de MPLSControl

Ses membres se composent de :

a. *crtl_monitor* : instance de *Control_Monitor*. Dans toute l'architecture de QMA, c'est ici que la classe en charge de la surveillance est instanciée. Si donc elle n'est point démarrée ici, point de surveillance;
b. *dbmgr* : instance de *Control_DB* nécessaire à l'interaction avec la base de données;
c. *Timer1* : minuteur utilisé pour périodiquement mettre à jour les boutons et composants graphiques affichant les paramètres de surveillance du réseau, et aussi pour relancer *crtl_monitor* si celui-ci s'était arrêté. En effet, *Control_Monitor* stoppe ses fonctions si aucune applet n'est connectée, mais l'objet est toujours instancié. Alors, quand une applet se connecte à nouveau il faut relancer les fonctions de *Control_Monitor*.

Ses méthodes sont composées de :

a. *button_switch_click* : bouton « interrupteur » qui instancie *crtl_monitor* s'il n'est pas démarré ou sinon qui stoppe la surveillance;
b. Les autres méthodes sont appelées chaque fois que l'utilisateur clique sur un des *TextBox* ou *SetBox* pour modifier des paramètres de configuration;
c. *MPLSControl* est le constructeur.

4.5.2 Onglet User

L'onglet utilisateur, en charge de la gestion des utilisateurs, est composé du contrôle utilisateur *UserMgtControl*, présenté figure 53. Celui-ci permet donc la création, modification et suppression de compte. Pour cela, c'est le module *AuthManagement* qui est sollicité.

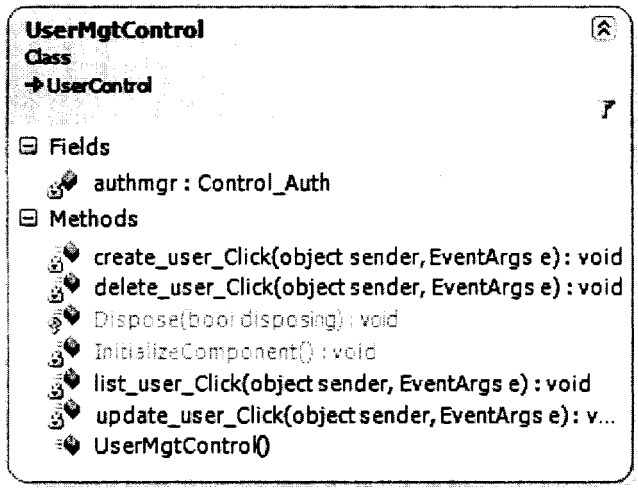

Figure 53 Structure de UserMgtControl

Ce contrôle, en plus des composants graphiques, nécessite comme membre une instance de *Control_Auth.*

Ses méthodes sont les suivantes :

a. *create_user_Click* : appelée par un bouton pour créer un utilisateur. Appelle la méthode *create_user* de *Control_Auth*;
b. *delete_user_Click* : appelée par un bouton pour créer un utilisateur. Appelle la méthode *delete_user* de *Control_Auth*;
c. *list_user_Click* : appelée par un bouton pour créer un utilisateur. Appelle la méthode *list_users* de *Control_Auth*;
d. *update_user_Click* : appelée par un bouton pour créer un utilisateur. Appelle la méthode *update_user* de *Control_Auth.*

4.6 Exemples de fonctionnement

Après avoir vu chacun des composants de QMA, examinons avec quelques exemples d'utilisation de l'outil, comment ceux-ci interagissent entre eux. Là encore établir une liste exhaustive de tous les cas d'utilisation serait fastidieux, donc seuls des cas typiques sont considérés ici, les autres fonctionnant de manière similaire. Par exemple, lister les Tunnels et les LSP sont deux opérations semblables.

4.6.1 Authentification réussie

Le premier diagramme de séquence présente une authentification réussie. Comme nous l'avons expliqué précédemment, au lancement de l'applet, l'onglet d'authentification est sélectionné par défaut, invitant l'usager à fournir son nom d'utilisateur et son mot de passe. La validation est automatiquement effectuée quand il souhaite changer d'onglet. La figure 54 illustre le processus.

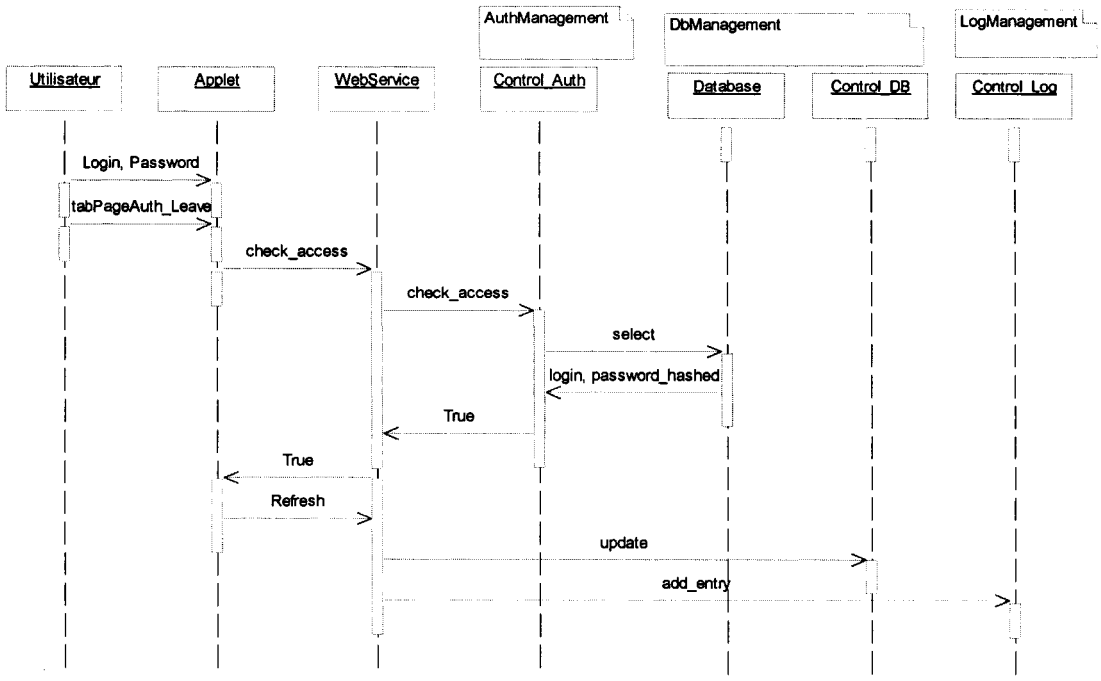

Figure 54 Diagramme de séquence d'authentification

Les étapes sont les suivantes :

- L'utilisateur fournit ses informations (nom d'utilisateur, mot de passe).
- En voulant changer d'onglet, la méthode *tabPageAuth_leave* de l'applet est appelée ; celle-ci vérifie que les données entrées sont correctes en appelant la méthode *check_access du Webservice*.
- Pour vérifier l'authentification, l'applet appelle la méthode *check_access* du Webservice, laquelle appelle la méthode de même nom de la classe *Control_Auth*.
- La méthode *check_access* de *Control_Auth* lit les informations sur l'usager contenues dans la base de données des usagers, et compare les informations.
- Comme les informations concordent, *Control_Auth* renvoie *True* au Webservice lequel transmet cette réponse à l'applet.
- L'utilisateur peut alors changer d'onglet, il est authentifié.

- L'applet lance la méthode *refresh* du Webservice pour signifier sa présence. Cette fonction appelée périodiquement cela permet de savoir qu'une applet est connectée même si l'usager ne fait aucune opération.
- La méthode *refresh* met à jour un champ dans une table avec la méthode update de *Control_DB*, et inscrit l'opération dans le fichier de logs avec la méthode *add_entry* de *Control_Log*.

4.6.2 Échec d'authentification

La figure 55 illustre ce qui se produit quand l'usager entre des informations d'authentification incorrectes.

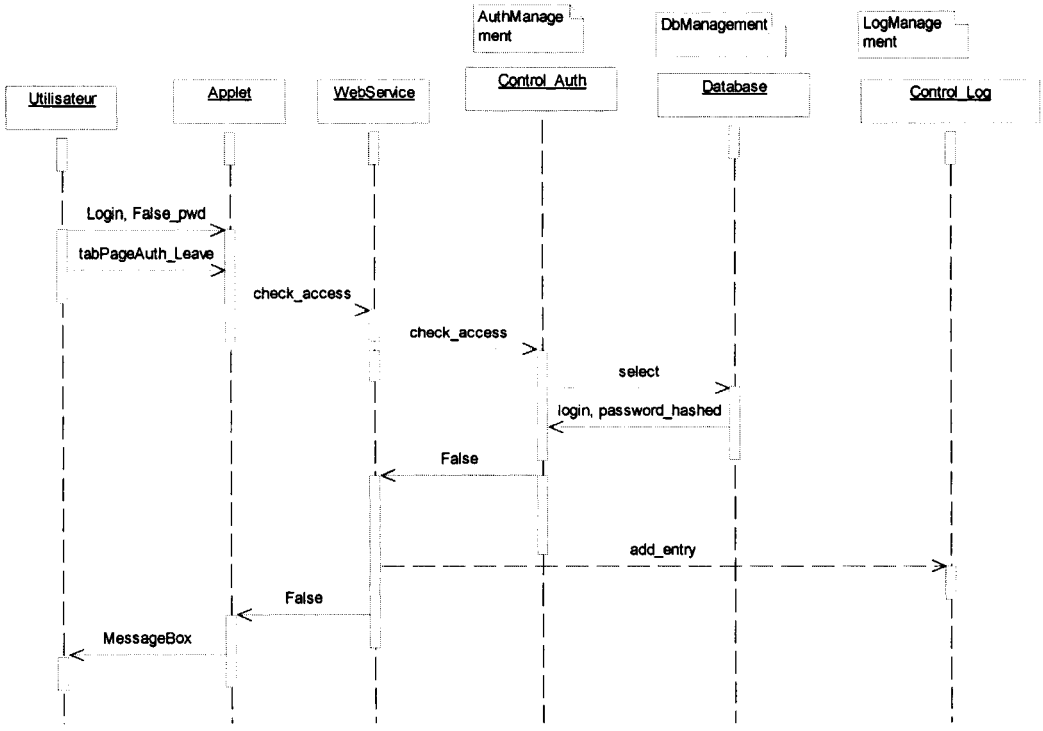

Figure 55 Diagramme de séquence d'échec d'authentification

Les étapes sont les suivantes :

- L'utilisateur fournit ses informations (nom d'utilisateur, mot de passe).
- En voulant changer d'onglet, la méthode *tabPageAuth_leave* de l'applet est appelée ; celle-ci vérifie que les données entrées sont correctes en appelant la méthode *check_access du Webservice.*
- Pour vérifier l'authentification, l'applet appelle la méthode *check_access* du Webservice, laquelle appelle la méthode de même nom de la classe *Control_Auth.*
- La méthode *check_access* de *Control_Auth* lit les informations sur l'usager contenues dans la base de données des usagers, et compare les informations.
- Comme les informations ne concordent pas, *Control_Auth* renvoie *False* au Webservice lequel transmet cette réponse à l'applet et notifie l'opération dans le fichier de logs avec la méthode *add_entry* de *Control_Log.*
- L'applet prévient l'usager alors par un *MessageBox* de l'échec de l'authentification. Elle ne permet alors pas de changer d'onglet.

4.6.3 Visualisation des tunnels

La figure 56 présente les échanges qui ont lieu lorsque l'utilisateur souhaite obtenir les informations sur les tunnels du réseau. On suppose que l'usager est déjà dans l'onglet Network de l'applet et donc qu'il s'est déjà authentifié.

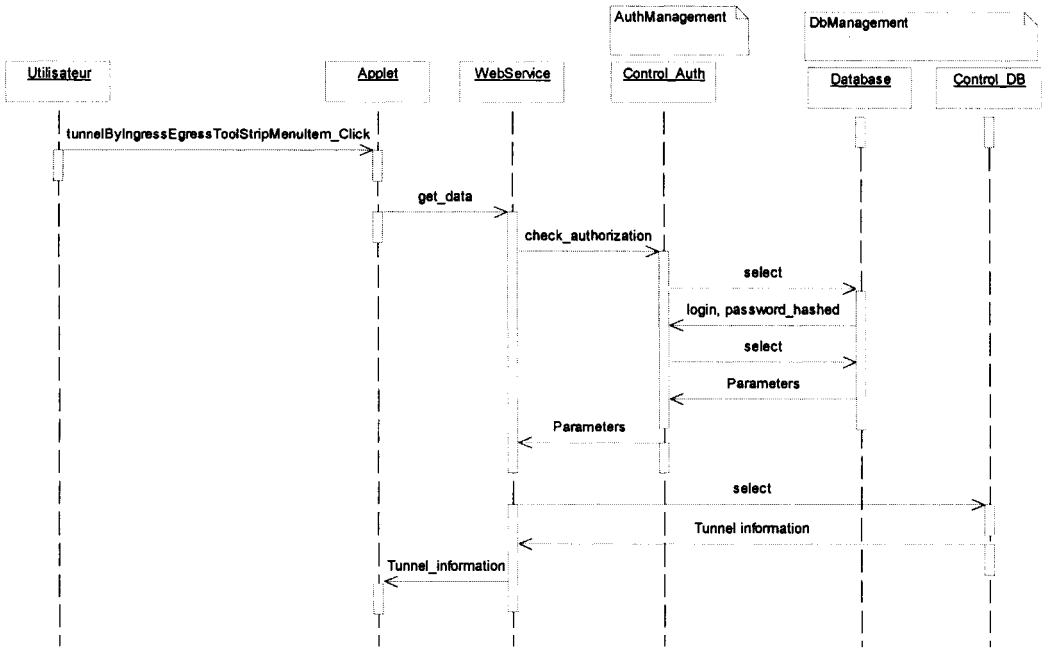

Figure 56 Diagramme de séquence de visualisation des tunnels

Les étapes sont les suivantes :

- L'utilisateur requiert l'affichage des tunnels par origine-destination en cliquant sur le bouton prévu à cet effet, lequel appelle la méthode *tunnelByIngressEgressToolStripMenuItem_Click.*
- L'applet appelle alors la méthode *get_data* du Webservice pour obtenir les informations des tunnels.
- Le Webservice s'assure d'abord de la légitimité de la requête, en appelant la méthode *check_authorization* de *Control_Auth.*
- *Check_authorization* vérifie tout d'abord le nom d'utilisateur et le mot de passe puis obtient les paramètres de connexion à la base de données contenant les informations du réseau.
- Celles-ci sont retournées au Webservice qui les utilise pour interroger la base et obtenir les configurations des tunnels.

- Le Webservice renvoie alors les informations à l'applet qui se charge de les traiter pour les présenter d'une manière ergonomique à l'utilisateur.

4.6.4 Réception d'une trap

La figure 57 présente l'interaction entre les différents composants, lorsqu'une *trap* est reçue signalant un tunnel tombé.

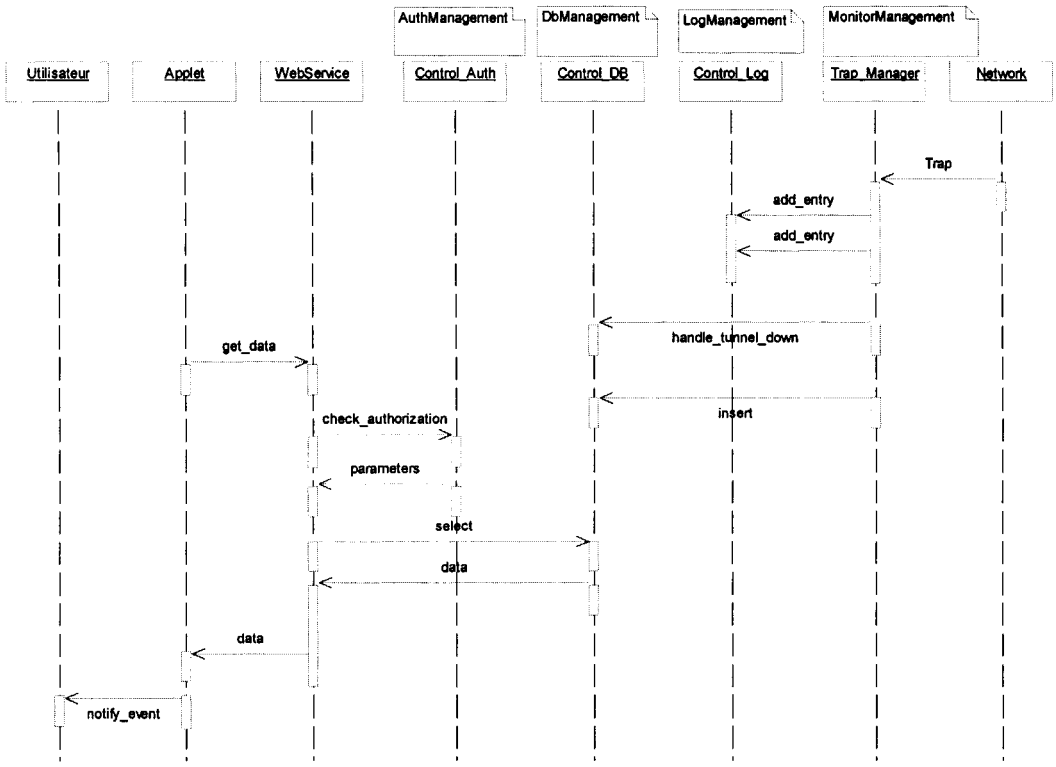

Figure 57 Diagramme de réception d'une trap d'un tunnel tombé

Les étapes sont les suivantes :

- Un routeur du réseau envoie une *trap* pour signaler un tunnel tombé.

- L'instance de *Trap_Manager* reçoit la *trap* et utilise la méthode *add_entry* de *Control_Log* pour la consigner dans le fichier de logs ainsi que dans le fichier de *traps*.
- *Trap_Manager* analyse la *trap*, reconnaît un tunnel tombé et appelle en conséquence la méthode *handle_tunnel_down* de *Control_DB*.
- Enfin, *Trap_Manager* signale comme évènement la *trap* dans la table *Event* avec la méthode *insert* de *Control_DB*.
- Entre temps, l'applet, qui vérifie périodiquement s'il y a de nouveaux évènements : la méthode *get_data* du Webservice est ainsi appelée.
- Comme nous avons vu précédemment, lorsque que *get_data* est appelée, le Webservice instancie un objet *Control_Auth* pour vérifier la légitimité de la requête et retourne les paramètres nécessaire à l'interrogation de la base de données. Une fois les données obtenues, le Webservice les renvoie vers l'applet.
- L'applet en détectant un nouvel évènement (le tunnel tombé) avertit l'utilisateur par un message dans l'icône de notification.

4.6.5 Création d'un nouvel utilisateur

La figure 58 présente la création d'un utilisateur.

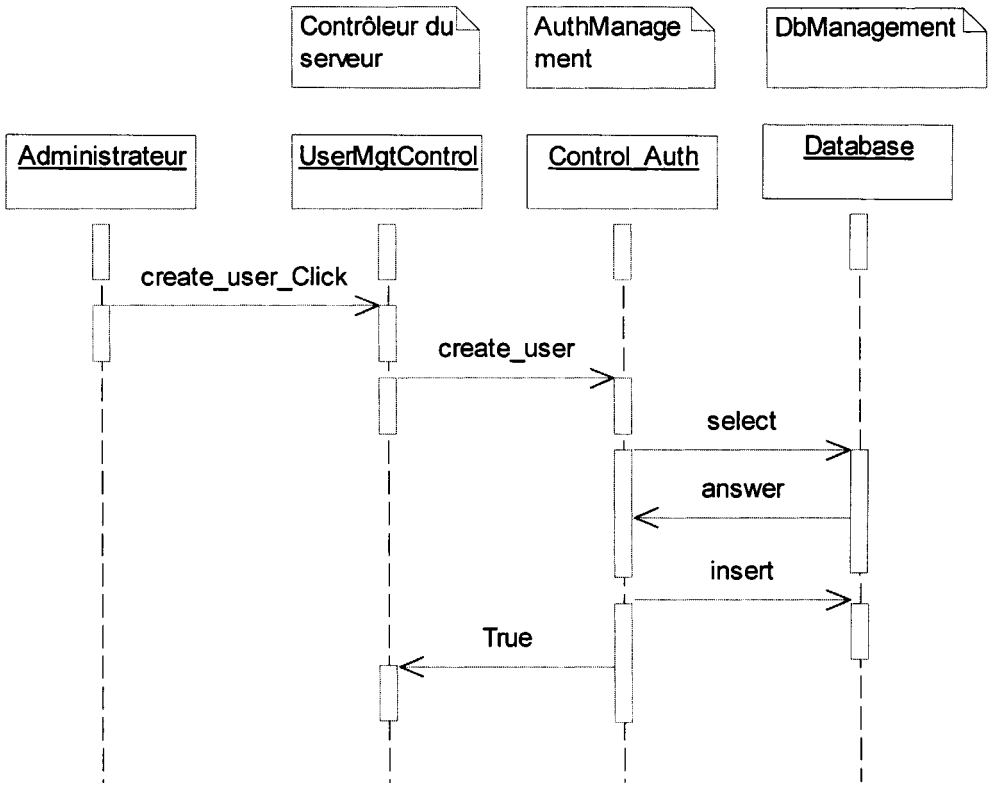

Figure 58 Diagramme de création d'utilisateur

Les étapes sont les suivantes :

- L'utilisateur clique sur le bouton de création d'un utilisateur.
- *UserMgtControl* appelle alors la méthode *create_user* de *Control_Auth*: celle-ci vérifie d'abord qu'il n'existe pas déjà le nom d'utilisateur avec la méthode *select* de *Database*, puis crée le compte avec la méthode *insert* et enfin retourne la valeur *True* pour confirmer le succès de l'opération.

4.7 Aspects non implémentés

Les sections précédentes témoignent de la richesse et de la complexité de QMA. Pour autant, le lecteur averti remarquera que certains points abordés dans le chapitre précédent ne sont pas traités. Cette différence témoigne de l'écart entre les documents théoriques servant de référence et les implémentations effectives dans les systèmes. En effet, l'étude conceptuelle de QMA se base sur la littérature disponible. Cependant, certains des aspects évoqués dans celle-ci ne se retrouvent pas dans la pratique, ou alors de manière incomplète. Ceci limite de facto les fonctionnalités de QMA.

Tout d'abord, le constructeur Cisco nous a confirmé ne pas avoir implémenté jusque là dans son IOS la MIB dédiée à la correspondance entre FEC et LSP, en l'occurrence la MIB MPLS-FTN [30]. Ainsi, il est impossible d'obtenir par SNMP le trafic qui y est envoyé dans chaque LSP. Le protocole SSH étant rarement installé sur l'IOS, la seule option disponible serait alors d'obtenir ces informations par Telnet. Mais ceci compromettrait la sécurité de QMA, et génèrerait beaucoup plus de trafic que des requêtes SNMP. De plus, cela nécessiterait un traitement supplémentaire pour l'outil, puisque les informations affichées par Telnet sont faites pour être lues par un utilisateur. Il a donc été jugé plus sage de réserver cette fonctionnalité quand la MIB MPLS-FTN sera disponible.

Pour la fonctionnalité de surveillance du réseau, là encore une MIB pour BFD est en développement mais pas encore implémentée [75]. Pour les mêmes raisons évoquées précédemment, il apparaît plus raisonnable d'attendre la disponibilité de cette MIB pour réaliser la fonctionnalité. En effet, sans la MIB la problématique consiste alors à trouver un moyen d'interaction avec les routeurs pour passer des commandes, moyen qui doit être sécurisé et indépendant du constructeur ce qui paraît difficile à réaliser en pratique. Cependant, l'écoute de *traps*, préconisée par la méthode de surveillance explicitée au chapitre précédent, est quant à elle réalisée dans QMA et donc bien disponible.

Enfin, l'algorithme de prédiction de trafic n'a pas été réalisé, car, comme nous l'avons expliqué lors de la définition des exigences, en tant que telle cette fonctionnalité a un intérêt limité, mais sert de base pour des travaux d'approfondissement dans ce domaine.

L'implémentation de QMA a naturellement dû s'adapter aux contraintes de disponibilité des MIB, limitant pour le moment certaines de ses fonctionnalités. Néanmoins, cet outil complexe répond déjà à son objectif de permettre la visualisation des informations d'un réseau, indépendamment de la marque des équipements, puisqu'il se base uniquement sur des MIB standards. Le chapitre suivant prouve la validité de QMA.

CHAPITRE 5

EXPÉRIMENTATION

« Que la stratégie soit belle est un fait, mais n'oubliez pas de regarder le résultat. »
Winston Churchill

Ce chapitre présente les résultats de l'expérimentation qui a été menée sur QMA afin d'éprouver l'outil et de voir s'il satisfait bien les exigences évoquées au chapitre 3. La première section explique la méthodologie de test et de suivi des bogues. Puis des résultats validant les spécifications sont fournis. Il ne s'agit pas ici de décrire chaque bouton ; là encore, rappelons que des guides d'installation et d'utilisation sont fournis en annexe 2 et 3, expliquant notamment l'interface.

5.1 Méthode de suivi des bogues

En plus des tests unitaires, un autre membre du projet a été chargé de tester QMA dans son ensemble, c'est-à-dire la partie MPLS et la partie QoS. Ceci permet d'avoir un regard neuf et objectif sur l'outil.

Pour mener à bien le suivi des bogues, l'outil *Bugzilla* [76] a été mis en place. Il s'agit d'un logiciel serveur permettant le suivi rigoureux des fautes, offrant notamment une interface Web et un système automatique de courriels. Ainsi, au fur et à mesure que les bogues ou améliorations ont été soumis, ceux-ci ont été corrigés, conduisant à plusieurs versions, stockées sur un serveur dédié. La figure 59 présente un aperçu de *Bugzilla*.

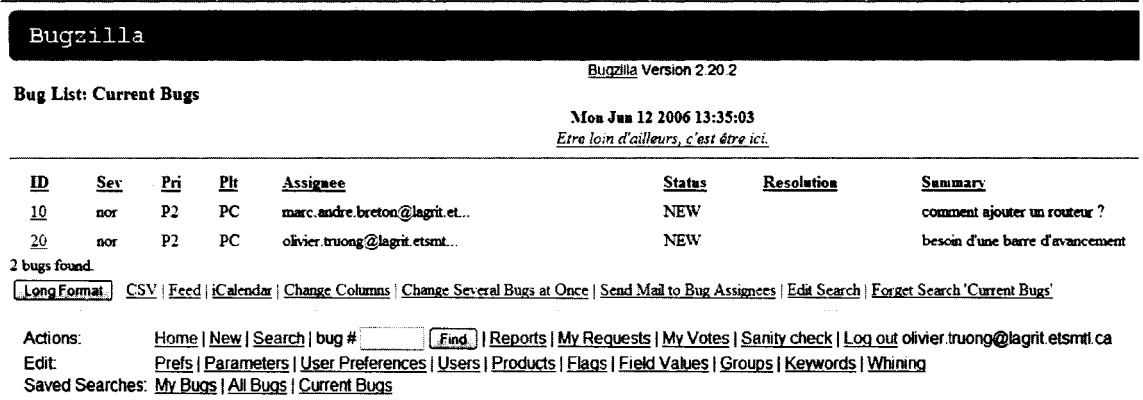

Figure 59 Aperçu de Bugzilla

5.2 Validation des spécifications

5.2.1 Méthodologie et environnement de test

Pour prouver que QMA répond bien aux besoins initiaux, la méthodologie choisie ici est de reprendre les spécifications une par une et de montrer les résultats de l'outil. Les tests présentés ici sont effectués sur un réseau de tests de Bell Canada, comprenant six routeurs et une centaine de LSP. La partie serveur de QMA est installée sur un PC avec comme processeur un Pentium cadencé à 3GHz et 1Go de RAM. Le navigateur Web utilisé est Internet Explorer.

5.2.2 Affichage de la topologie du réseau

Dans l'onglet *Network* de l'applet, les routeurs sont dessinés et déplaçables à souhait par l'utilisateur. De plus, le bouton "*General Information*" du menu permet l'affichage des informations du réseau. En cliquant dessus, trois options sont proposées :

a. Lister les adresses IP disponibles : en cliquant sur l'une d'entre elle, le routeur auquel elle appartient est surligné et ses informations sont représentées;

b. Lister les routeurs par nom : en cliquant sur l'un d'entre eux, le routeur en question est surligné et ses informations sont représentées;
c. Lister les liens physiques : permet d'obtenir pour chaque lien les informations caractéristiques : adresses IP, type de lien et bande passante.

La figure 60 présente cette fonctionnalité. Dans cet exemple l'adresse *172.20.254.12* a été sélectionnée, correspondant à l'adresse de boucle locale du routeur PE2. Toutes les interfaces (et sous interfaces) de ce routeur sont alors listées.

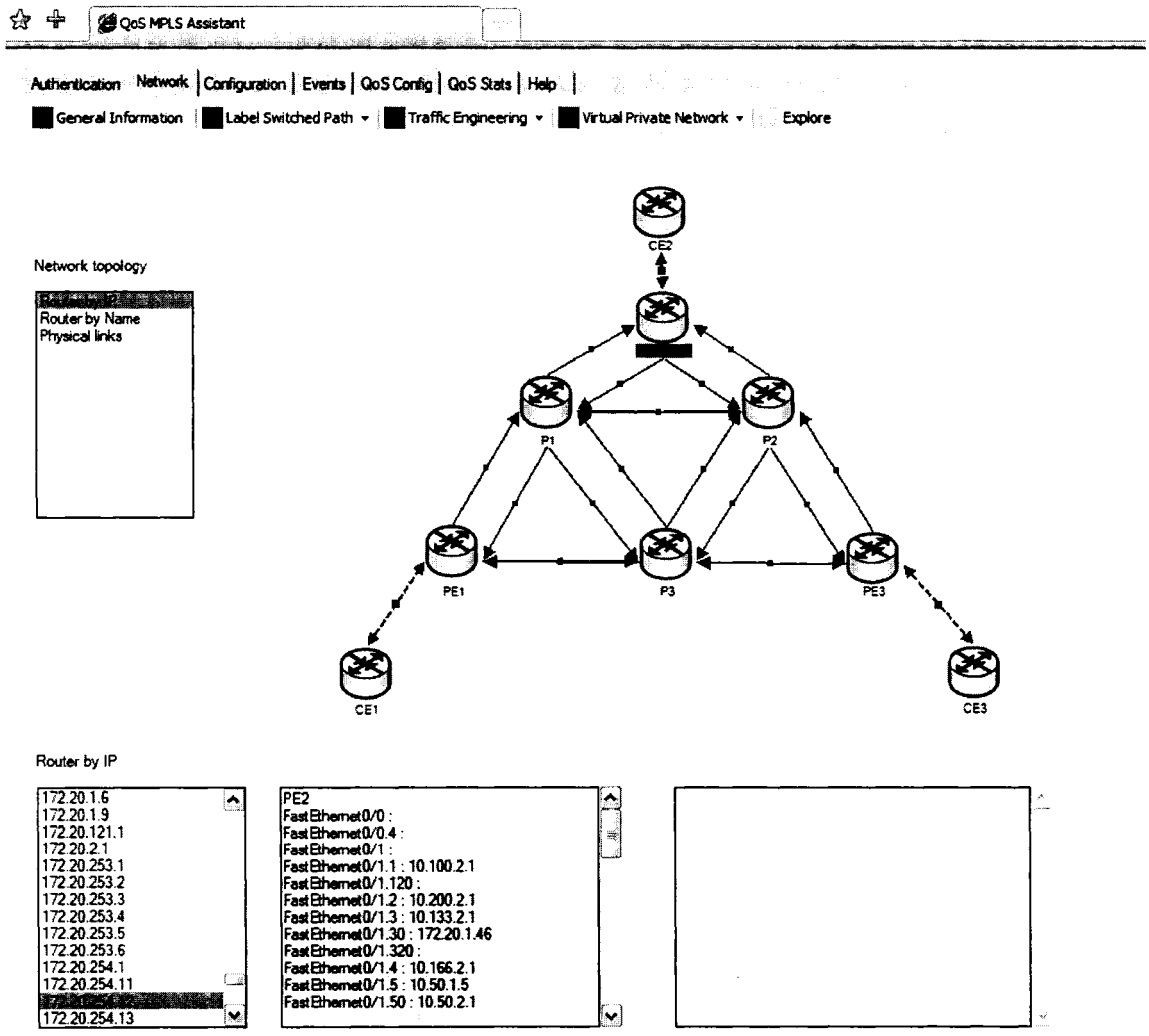

Figure 60 Affichage de la topologie du réseau

De plus, en passant le curseur sur un des routeurs, les informations de celui-ci sont également affichées par *tooltip* comme illustré sur la figure 61 qui s'affiche quand le curseur est placé sur le routeur P1. Ceci permet une consultation rapide sans clics.

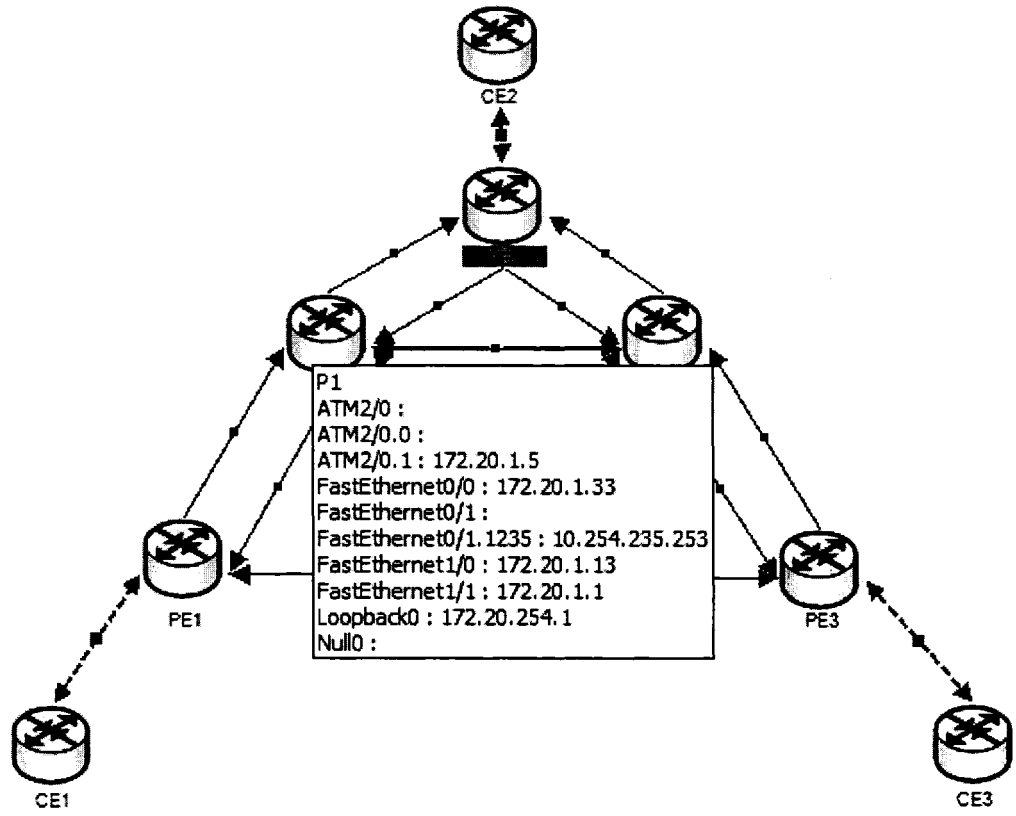

Figure 61 Affichage des informations par tooltip

5.2.3 Affichage des LSP

Dans l'onglet *Network* de l'applet, il est possible de lister les LSP par origine-destination. QMA dresse alors tous les LSP correspondants. En sélectionnant un LSP particulier, sa décomposition en segments est affichée. La figure 62 présente cette fonctionnalité, pour un LSP entre PE1 et PE2. Elle dessine notamment le LSP composé des segments *PE1-P1* avec le label *31* et *P1-PE2* avec le label *19*.

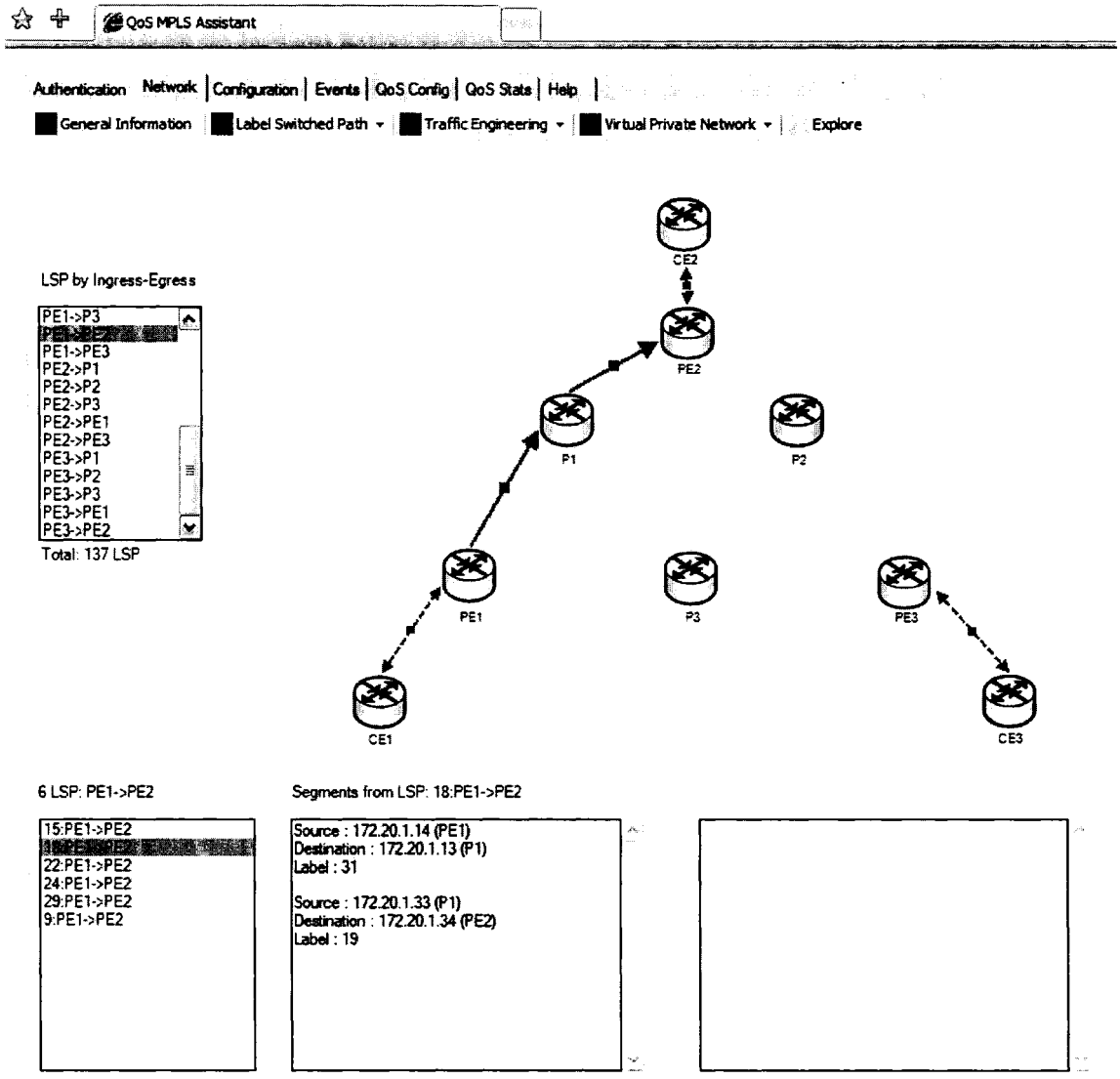

Figure 62 Affichage des LSP

On peut vérifier cette information en tapant sur le routeur avec la commande *show mpls forwarding-table* comme le montre la figure 63.

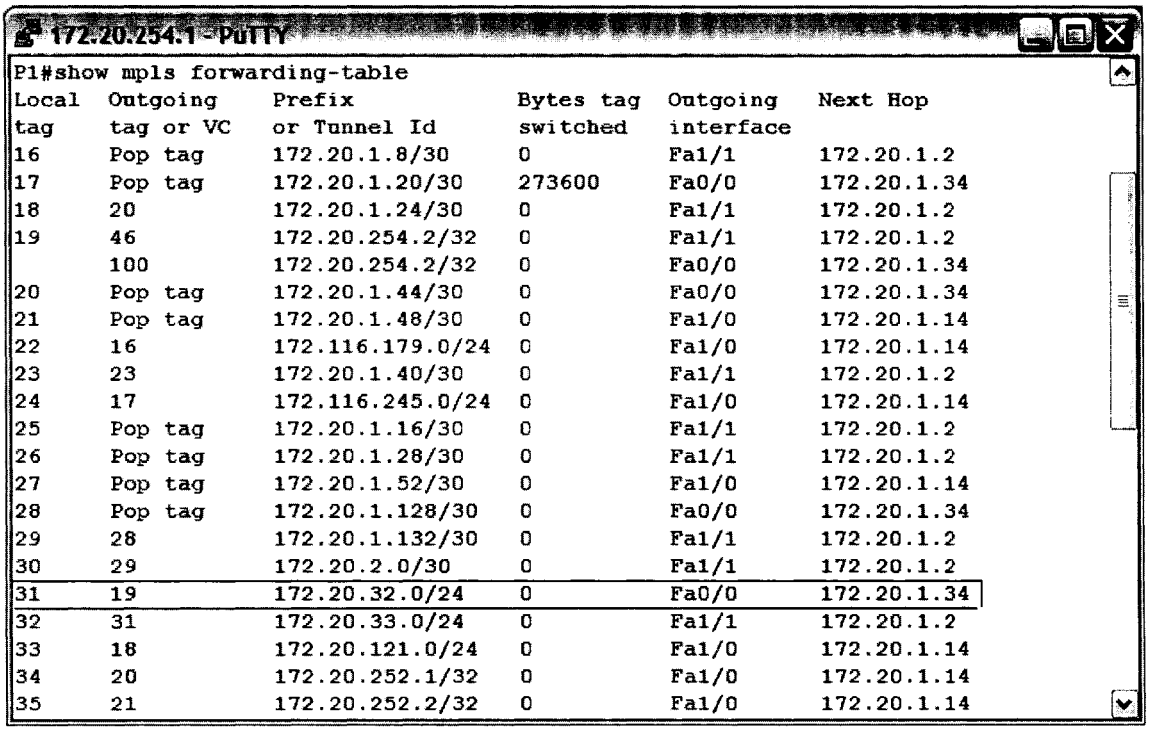

Figure 63 Configuration des LSP vue du routeur P1

On retrouve notamment le LSP affiché figure 62. On peut vérifier la correspondance des labels 31 (label entrant) et 19 (label sortant) et l'adresse du prochain saut (*172.20.1.34*). Comme expliqué précédemment, QMA n'affiche pas l'adresse à laquelle est affecté le LSP, puisque la MIB MPLS-FTN n'est à ce jour pas implémentée. Par ailleurs, certaines entrées de la table du routeur P1 peuvent ne pas se retrouver dans QMA, si celles-ci sont liées à des interfaces non répertoriées dans la base de données de l'outil. QMA ne considère en effet que les LSP sur la topologie connue.

5.2.4 Affichage des tunnels

À l'instar des LSP, QMA permet l'affichage des tunnels MPLS TE avec leur configuration et, s'ils sont opérationnels, des statistiques sur le trafic qui y transite. La figure 64 illustre un exemple de tunnel affiché par QMA. Ses propriétés de configuration

sont présentées, et, étant opérationnel (qu'on peut voir avec *Oper status : up*), des statistiques régulièrement rafraîchies sont affichées.

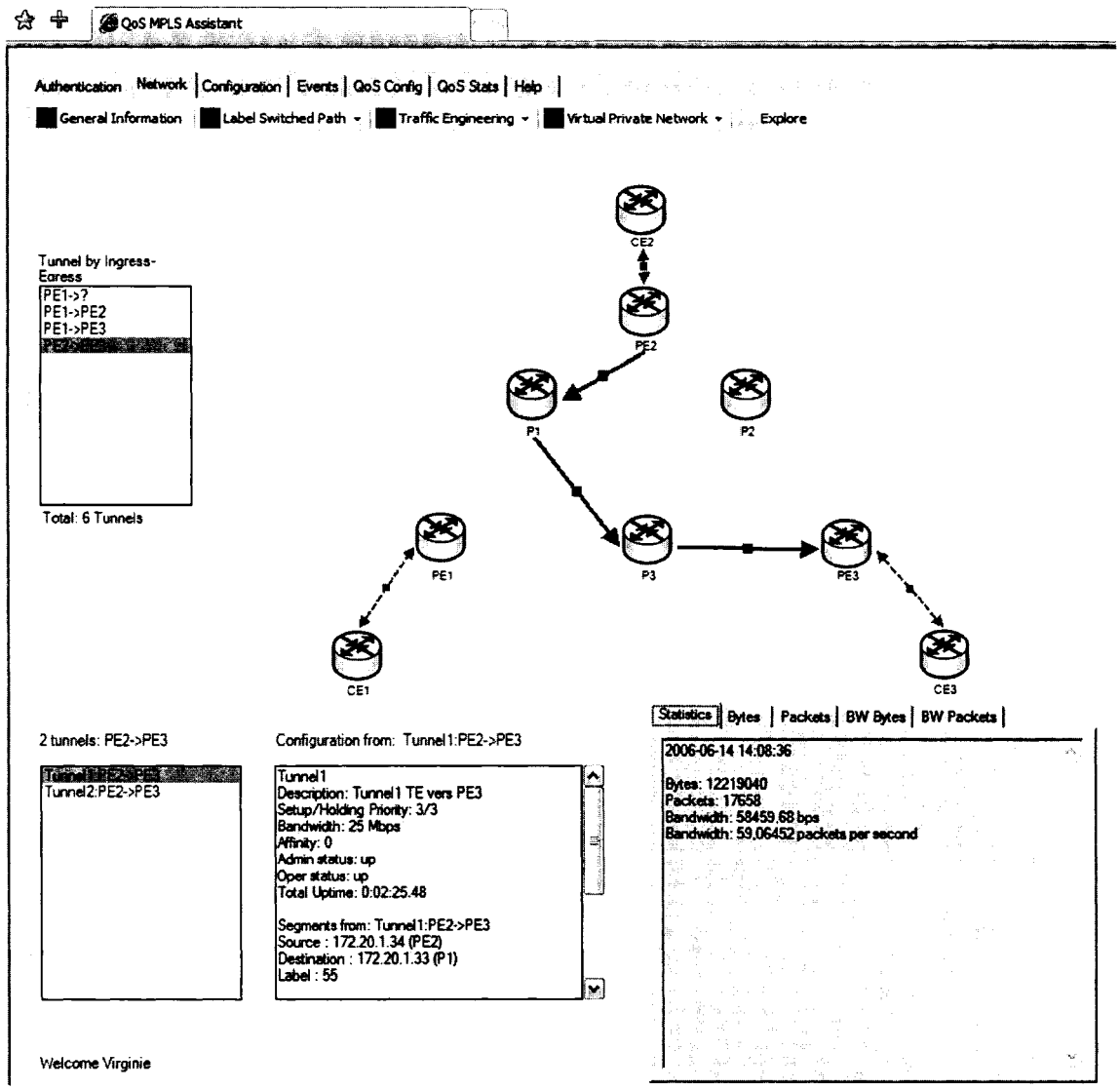

Figure 64 Affichage des tunnels

Comme expliqué dans le chapitre précédent, quatre types de données sont fournies (nombre d'octets, nombre de paquets, bande passante en octets par seconde et bande passante en paquets par seconde) répartis chacun dans un onglet. La figure 65 fournit un meilleur aperçu, avec la bande passante en paquets par seconde.

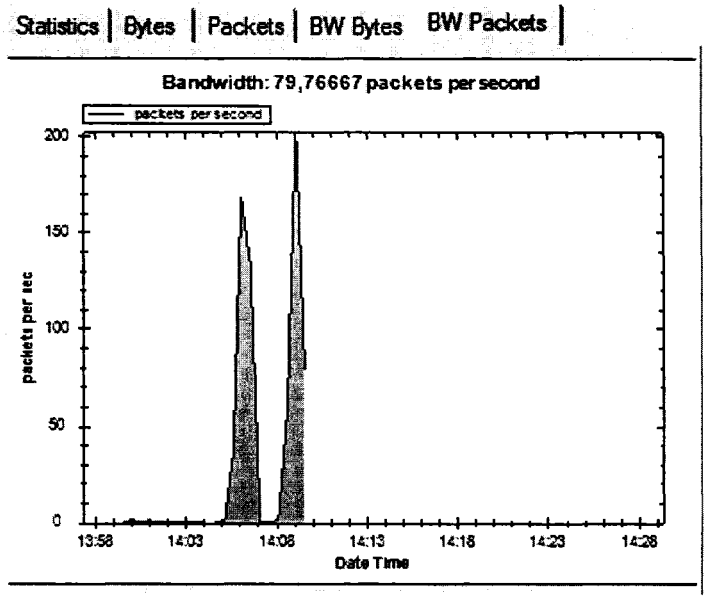

Figure 65 Affichage de statistiques d'un tunnel

Une fois de plus, ces informations peuvent être vérifiées en se connectant par *Telnet* au routeur. La figure 66 montre le résultat affiché dans le terminal.

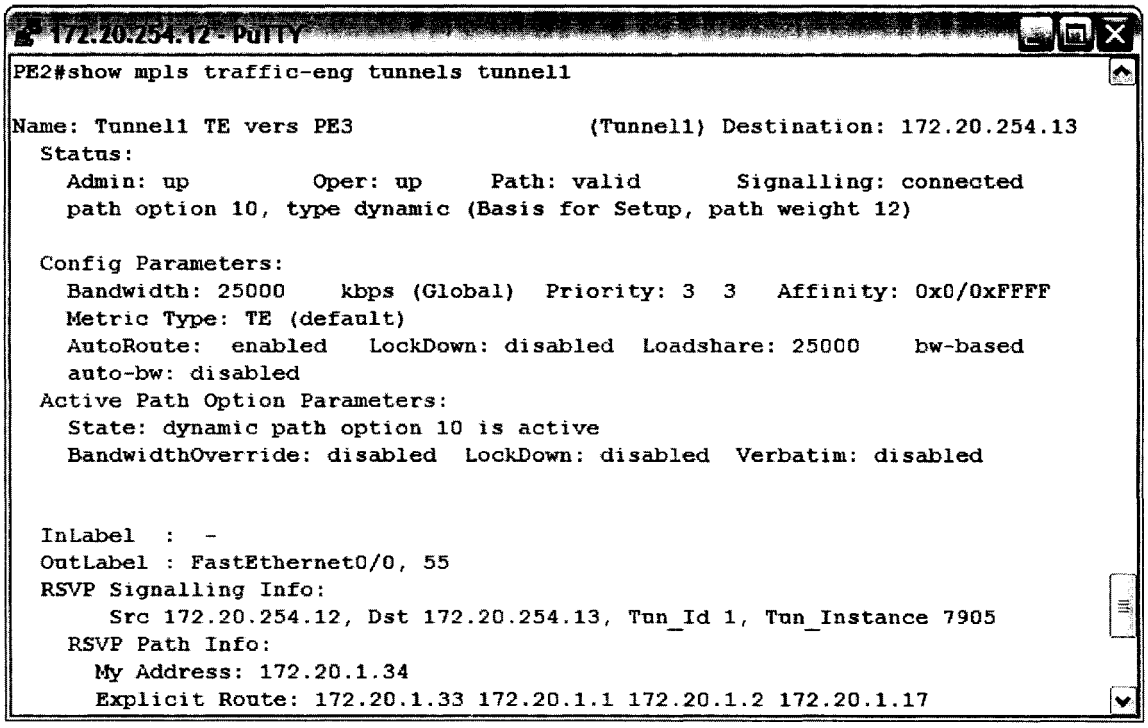

Figure 66 Configuration d'un tunnel vu du routeur

Une rapide comparaison avec la figure 64 démontre la correspondance des informations :

- Nom du tunnel : *Tunnel1 TE vers PE3*
- Statut administratif : *up*
- Statut opérationnel : *up*
- Destination : *172.20.254.13*
- Bande passante : *25000 kbps*
- Label sortant : *55*

Les informations obtenues avec une ligne de commande par une connexion Telnet sont alors disponibles en quelques clics avec QMA.

5.2.5 Affichage des VPN

QMA permet l'affichage des VPN en listant des propriétés de chaque VRF. Il est possible de lister les VRF par VPN ou par routeur. En choisissant un VRF dans la liste, ses propriétés, interfaces et routes sont alors affichées La figure 67 en donne un exemple, avec l'affichage du VRF *cust1* sur PE1.

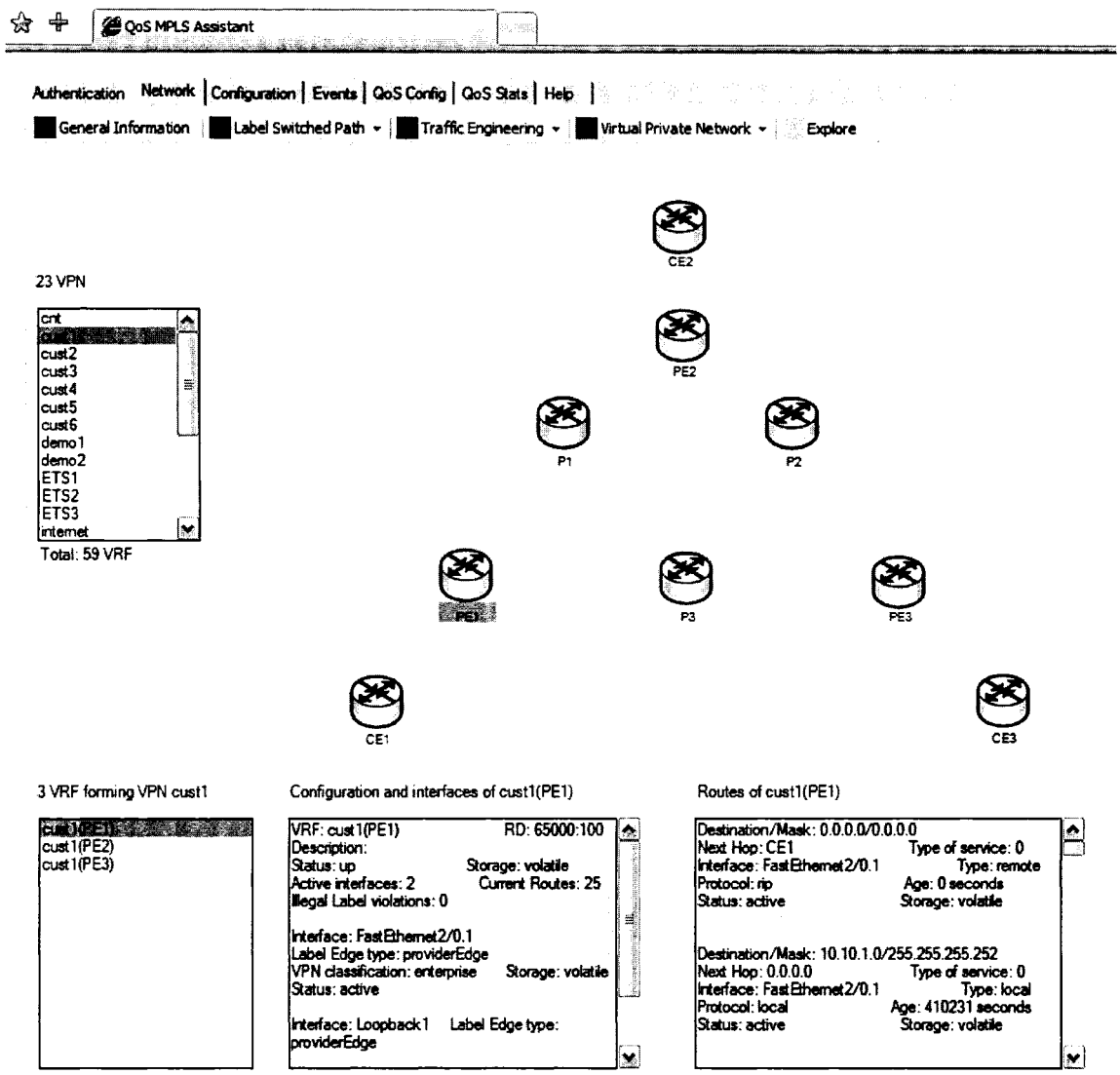

Figure 67 Affichage d'un VRF

Une connexion *Telnet* au routeur permet une fois de plus de corroborer l'information, comme le montre la figure 68.

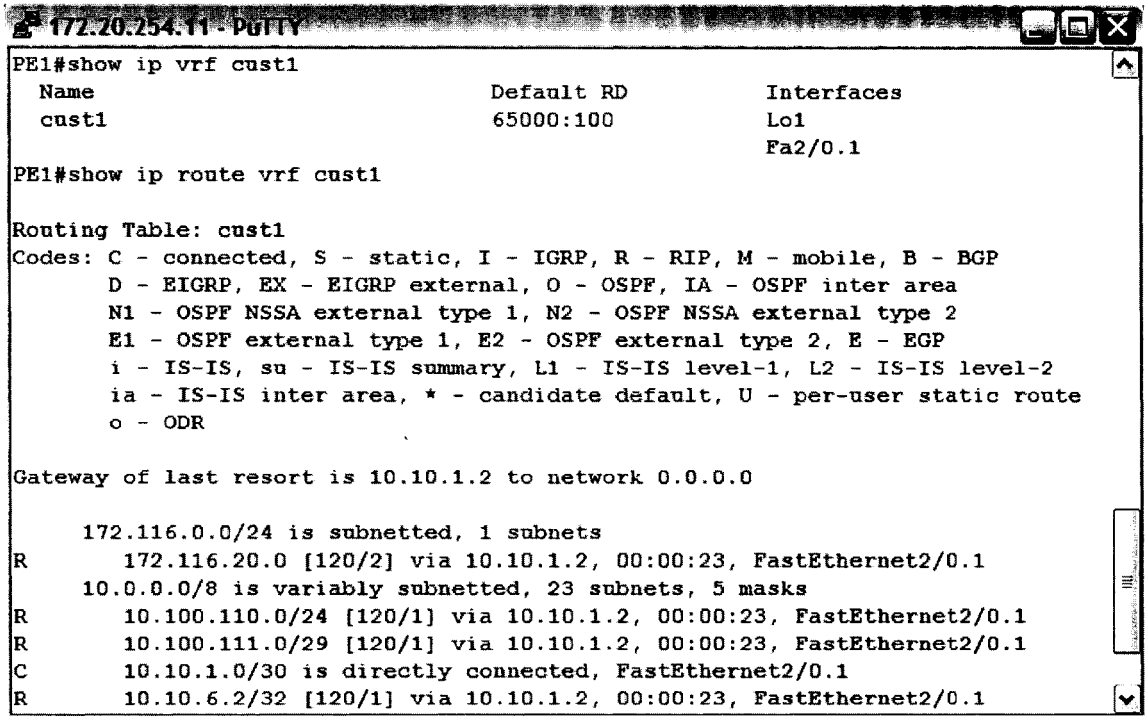

Figure 68 Configuration d'un VRF vu du routeur

On a bien un *Route Distinguisher* de *65000 :100*, deux interfaces (*Loopback1* et *Fa2/0.1*) ainsi que plusieurs routes, par exemple *10.10.1.0* connectée à l'interface *FastEthernet2/0.1*.

5.2.6 Surveillance du réseau

La fonctionnalité de surveillance du réseau a été développée selon les MIB disponibles, comme expliqué dans le chapitre précédent. QMA permet l'écoute de *traps* des routeurs et lors de la réception de l'une d'entre elles, l'outil analyse leur contenu et prend les mesures adéquates. Par exemple, lorsqu'un tunnel tombe (provoqué ici en appliquant sur le tunnel la commande *shutdown* de l'IOS), ou si une interface tombe, une *trap* est générée. QMA reçoit cette alerte et prévient l'utilisateur par un message dans la barre de notification, comme le montre la figure 69.

Figure 69 Notification d'un nouvel évènement

En cliquant sur l'onglet *Event*, cet évènement est répertorié comme illustré sur la figure 70.

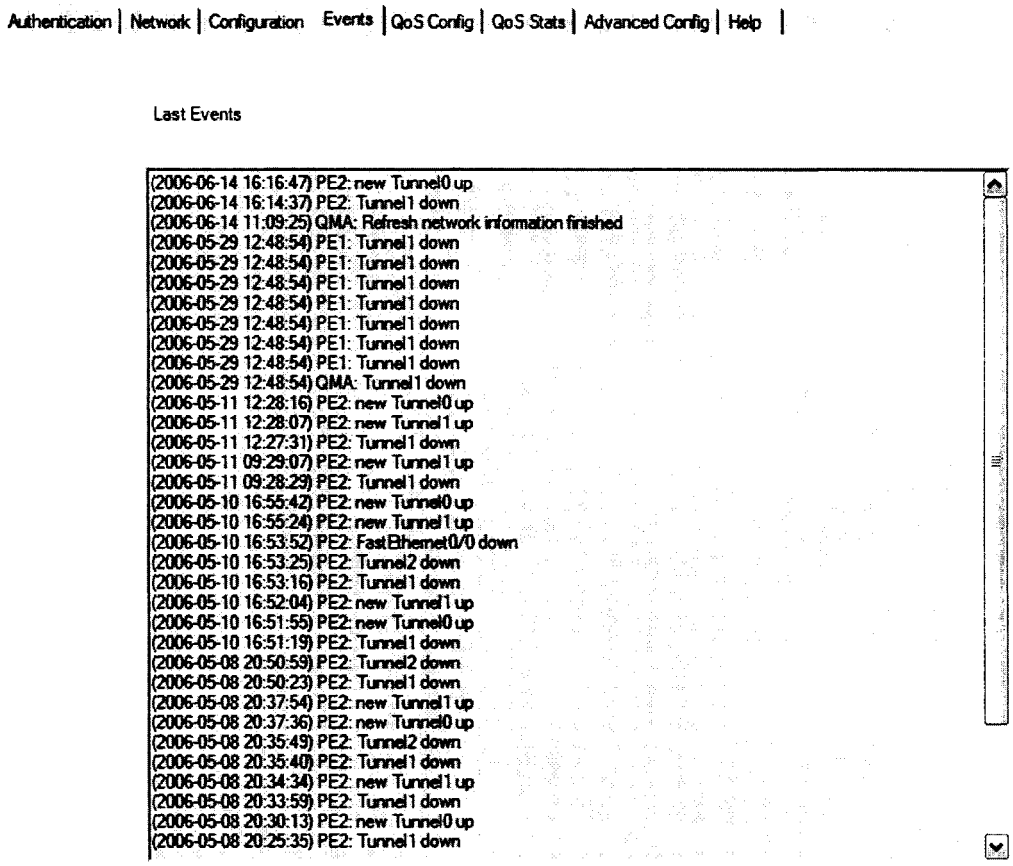

Figure 70 Nouvel évènement dans l'onglet Event

Enfin, l'utilisateur peut avoir l'information complète en visualisant le sous-onglet propre aux *traps* dans l'onglet *Advanced Config* comme le montre la figure 71.

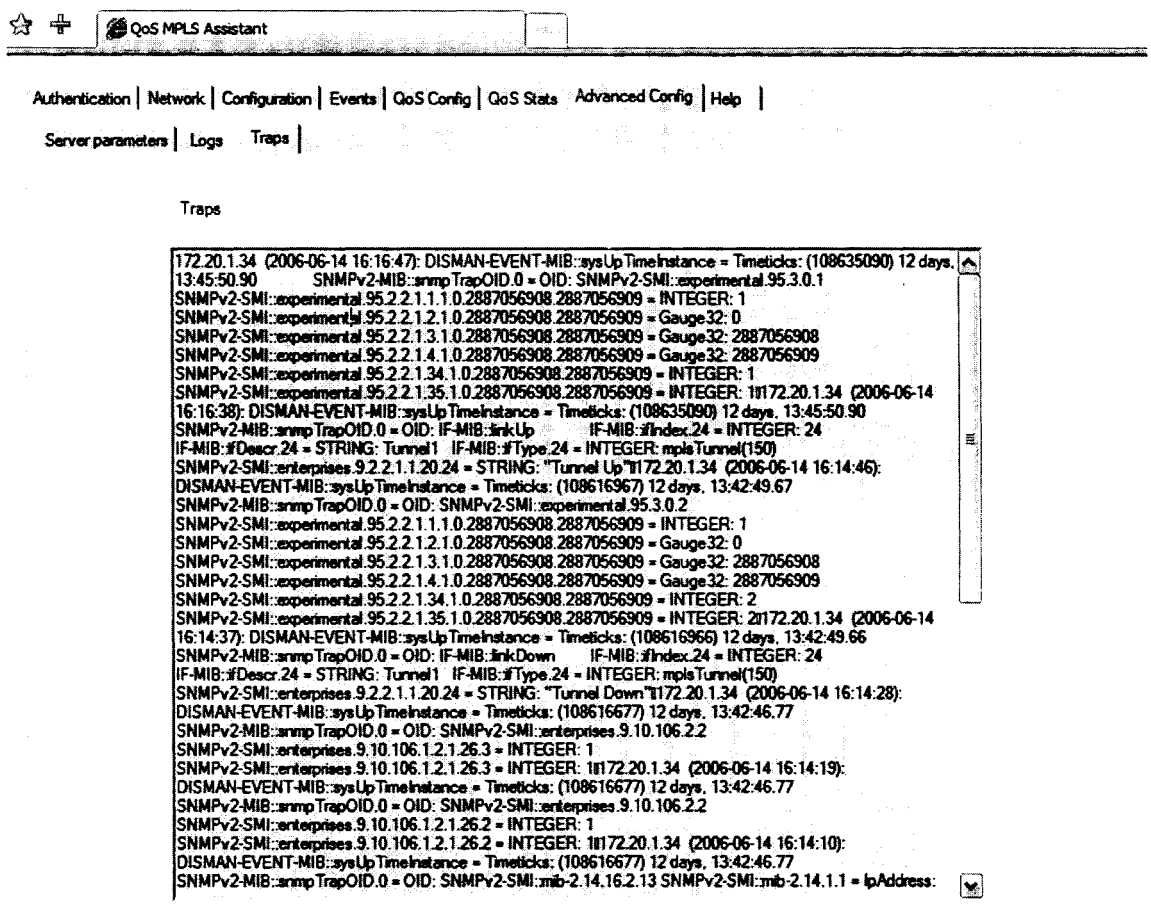

Figure 71 Affichage des traps dans l'onglet traps

5.2.7 Authentification

QMA inclut un mécanisme d'authentification par nom d'utilisateur/mot de passe et ne permet pas d'accéder aux fonctionnalités de l'outil tant que la vérification des paramètres n'est pas effectuée. La figure 72 présente l'onglet d'authentification et la figure 73 le message affiché en cas d'informations incorrectes.

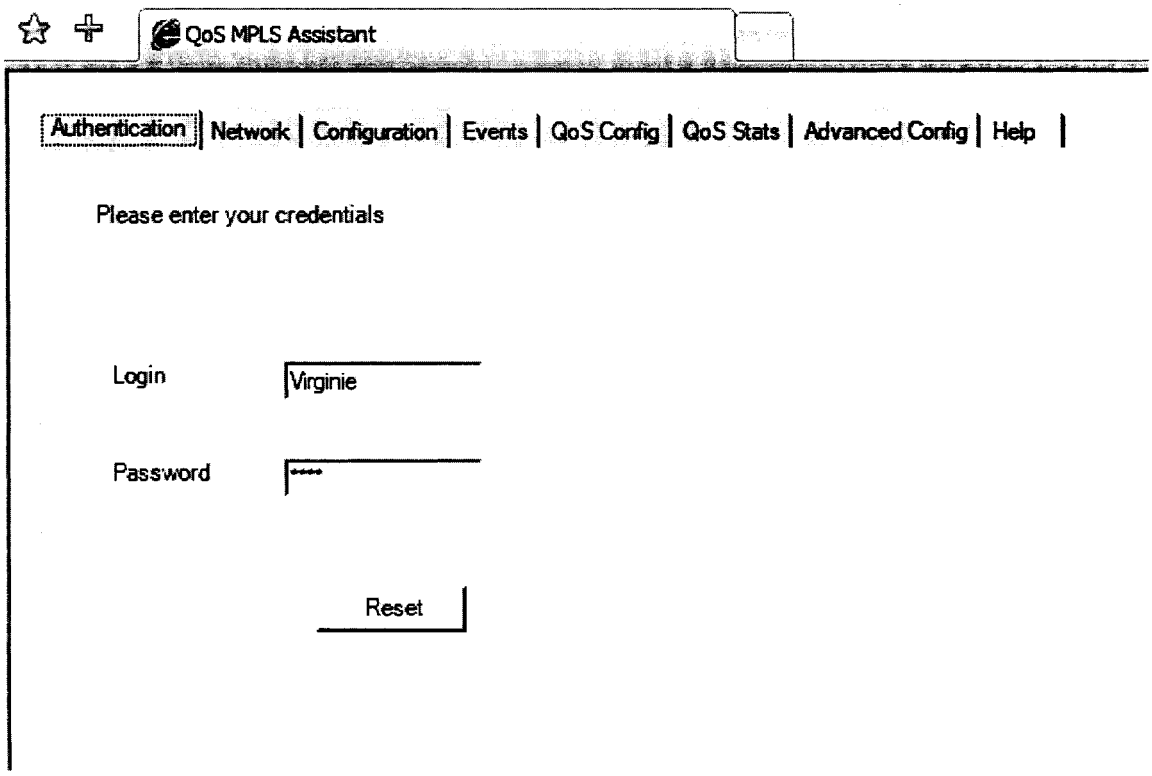

Figure 72 Onglet d'authentification

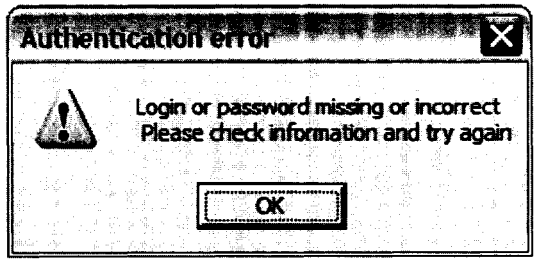

Figure 73 Message d'erreur lors d'authentification incorrecte

5.2.8 Multi utilisateur

L'architecture client-serveur permet d'avoir plusieurs utilisateurs simultanément. Nous avons donc testé QMA sur cinq ordinateurs différents en même temps, en effectuant sur chacun des opérations comme visualiser les tunnels, LSP, VPN et modifier la configuration de la surveillance. La figure 74 montre, à travers l'onglet de logs, les cinq clients connectés en même temps.

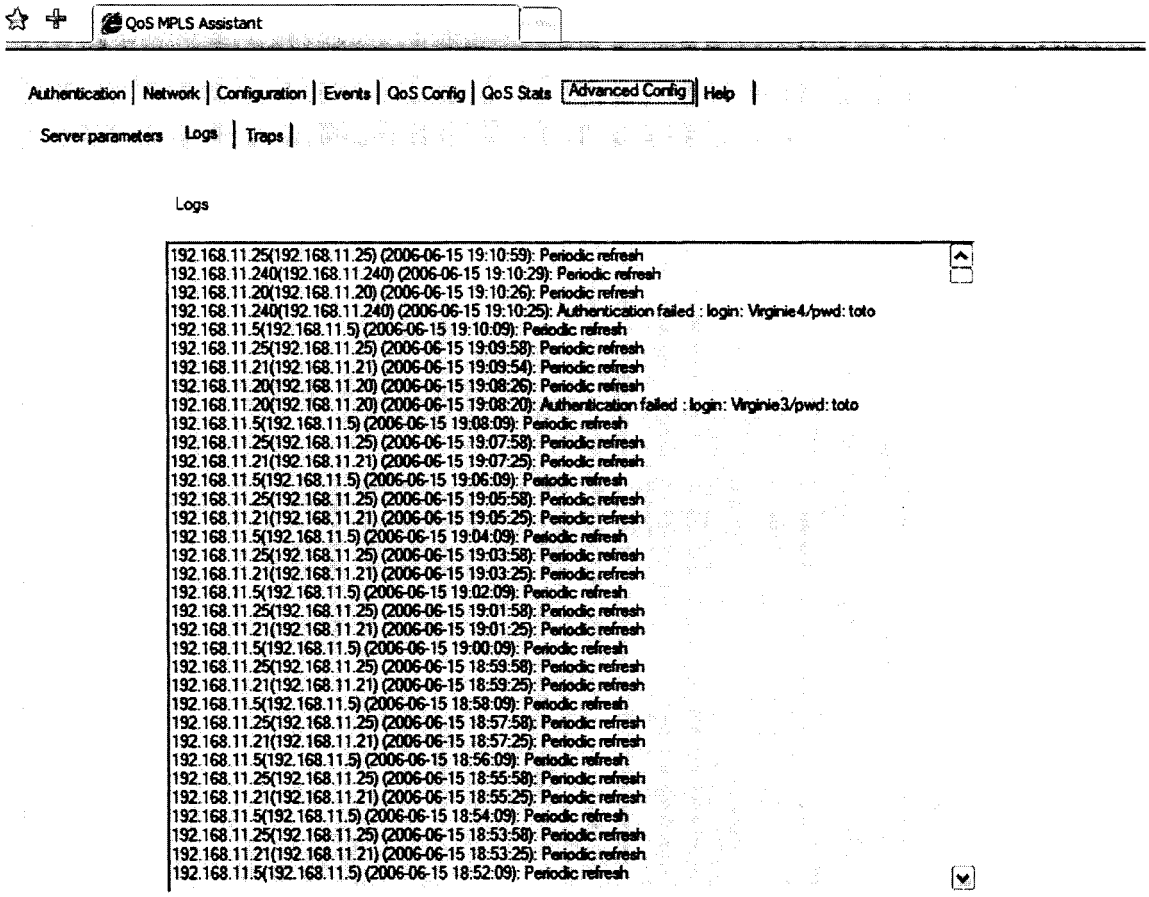

Figure 74 Logs de cinq utilisateurs simultanés

Il est notamment possible de connaître les adresses IP des postes clients. Dans le cadre de ce test il s'agit de *192.168.11.25*, *192.168.11.240*, *192.168.11.20*, *192.168.11.5* et *192.168.11.21*.

5.2.9 Disponibilité

La disponibilité de QMA repose exclusivement sur la fiabilité du serveur IIS, et en particulier du Webservice. Cette spécification n'est donc pas directement liée au code de QMA et nous nous reposons ici sur la technologie .Net.

5.2.10 Confidentialité

Comme mentionné précédemment, l'utilisation de la technologie SSL permet une communication confidentielle entre l'applet et le Webservice. SNMP v3 garantit quant à lui la confidentialité entre les routeurs et le serveur. La figure 75 montre un aperçu par *Ethereal* de l'échange par protocole sécurisé entre le client *192.168.11.5* et le serveur *192.168.11.21* lors de la phase d'authentification.

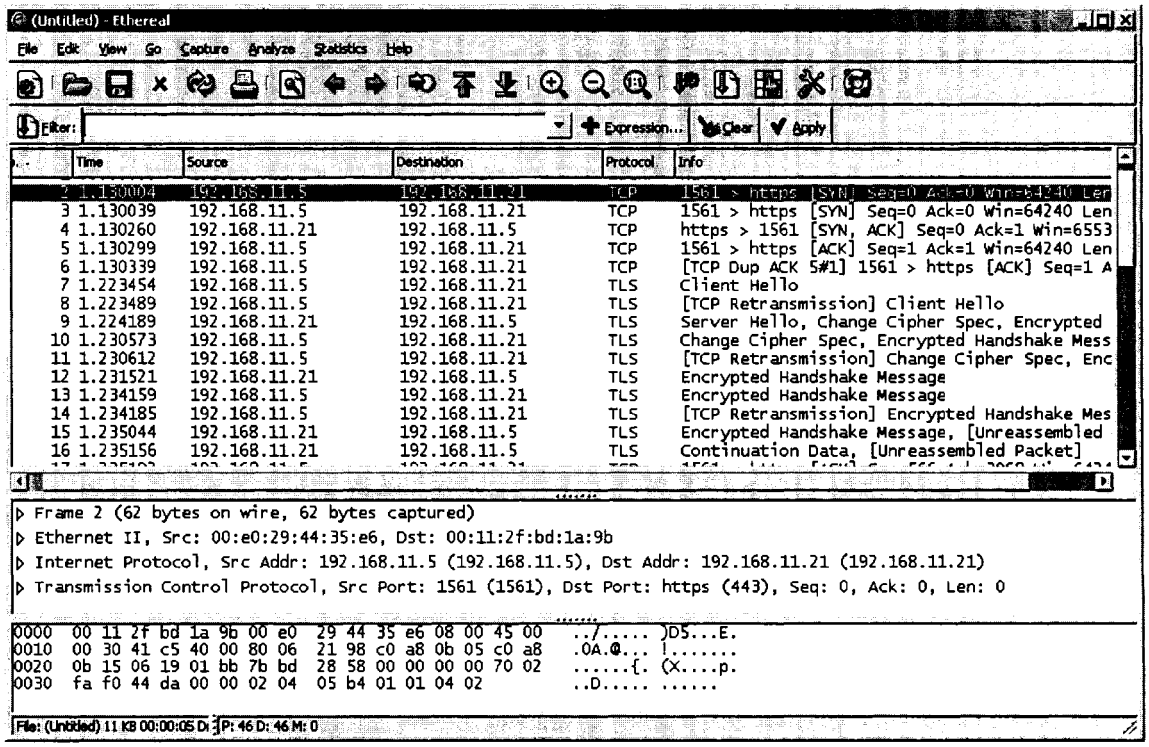

Figure 75 Aperçu du trafic échangé entre client et serveur QMA

5.2.11 Évolutivité

La structure modulaire de QMA permet d'ajouter ou de modifier facilement un élément sans affecter outre mesure l'architecture globale de l'outil. Certains modules ne contiennent actuellement qu'une classe, pour permettre des évolutions. En outre, les principes de couplage et cohésion ont été respectés : chaque classe répond à un besoin précis, et l'interconnexion entre les classes a été minimisée, même si certains éléments sont beaucoup sollicités, notamment ceux du module *DbManagement*.

5.2.12 Simplicité, ergonomie

La répartition en onglets permet une présentation structurée des fonctionnalités, et le passage de l'une à l'autre en peu de clics de souris. L'utilisation de la technologie .Net a

facilité l'ajout d'éléments graphiques conviviaux, telle l'icône dans la barre de notification qui permet le signalement d'évènements.

5.2.13 Indépendance des constructeurs

Les MIB utilisées sont des MIB définies dans des RFC. Ainsi, elles ne dépendent pas d'un seul constructeur. En pratique, il pourrait arriver que l'équipementier ne respecte pas le standard ; dans un tel cas, il suffirait alors de définir différentes classes par constructeur, dérivant des classes originales comme *Control_DB* et d'instancier l'une ou l'autre dépendamment de la marque du routeur.

5.2.14 Impact minimum sur le réseau

Pour accomplir toutes ses fonctionnalités, QMA collecte beaucoup d'informations, sollicitant ainsi les routeurs en conséquence. Néanmoins, des précautions ont été prises pour en minimiser l'impact. En effet, le protocole utilisé pour communiquer avec le réseau est SNMP, lequel utilise UDP, plus léger que TCP. De plus, lorsque plusieurs paramètres doivent être récupérés, la requête SNMPBULKGET a été préférée à plusieurs SNMPGET consécutifs. Ainsi, il semble difficile de réduire l'impact de QMA sur le réseau, si ce n'est de collecter moins d'informations, ce qui se traduirait par une détérioration des fonctionnalités de l'outil.

Hormis les aspects non implémentés, expliqués dans le chapitre précédent, QMA répond aux exigences initiales et permet ainsi, comme nous l'avons vu, de visualiser rapidement des informations accessibles certes par une connexion directe avec le routeur, mais de manière beaucoup moins conviviale.

CONCLUSION

La gestion des réseaux est un domaine vaste et complexe, quelle que soit la technologie à laquelle elle s'applique, y compris MPLS. Devenue incontournable dans la mutation vers des réseaux nouvelle génération, celle-ci requiert une attention particulière en matière de gestion. Des équipes de recherche en ont abordé plusieurs aspects, notamment la configuration dynamique de tunnels. Des professionnels ont quant à eux adapté des outils déjà existants. Cependant, aucun outil ne permet de présenter toutes les informations liées à MPLS (LSP, tunnels et VPN) d'une façon simple et claire à l'administrateur. QMA a donc été conçu pour permettre une visualisation rapide des informations tout en offrant de la surveillance de réseau. Contrairement aux simulations où l'on peut supposer l'environnement idéal, il a fallu prendre en compte les éléments existants actuellement dans les routeurs pour son implémentation, limitant de fait certaines des fonctionnalités de l'outil. De plus un système de gestion de réseaux complet dépasse largement le cadre d'un mémoire de maîtrise. Néanmoins, QMA est pleinement fonctionnel comme les tests l'ont prouvé.

Ce mémoire a donc été l'occasion d'apporter une contribution à la recherche d'une optimisation de la gestion des réseaux MPLS, en proposant un outil indépendant du constructeur des routeurs. Les aspects de développement théorique qui n'ont pu aboutir à la réalisation expérimentale permettront quant à eux une meilleure intégration de nouvelles fonctionnalités de QMA à l'avenir.

RECOMMANDATIONS

Comme nous l'avons évoqué précédemment, nous n'avons pas ici la prétention en un mémoire de proposer un outil qui répond à chacun des aspects de la gestion des réseaux MPLS. C'est pourquoi QMA a été conçu de manière modulaire, pour faciliter son évolution. Seule l'imagination des développeurs peut limiter les fonctionnalités que l'on pourrait y ajouter ; voici néanmoins quelques suggestions.

Quand la MIB MPLS-FTN sera disponible, il serait important de récupérer les informations liant la FEC aux LSP et tunnels. Ainsi l'utilisateur sera en mesure de connaître, pour chaque route MPLS, le type de trafic qui y est envoyé. Ceci permettra le rapprochement entre les fonctionnalités de MPLS et celles de QoS de QMA.

Également, quand la MIB liée à BFD sera implémentée dans les routeurs, QMA pourrait détecter tout bris de LSP et tunnel par ce moyen complémentaire de l'écoute des *traps* déjà mis en place.

Il pourrait s'avérer judicieux également de développer une méthode permettant la détection automatique des différents routeurs accessibles, QMA s'adaptant ainsi automatiquement à la topologie du réseau à administrer.

Un module en charge des statistiques pourrait y être ajouté, permettant la génération automatique de rapports, diagrammes ou tout autre document utile à un administrateur, voire utilisable par une application tierce.

À l'instar de TEAM [35], un module de configuration automatique des LSP et tunnels pourrait être ajouté à QMA : il comprendrait notamment un modèle de décision basé sur l'étude de prédiction de trafic effectuée dans ce mémoire.

Si d'autres équipementiers ne respectent pas les MIB MPLS standards, il serait judicieux d'ajouter une couche d'abstraction au niveau des objets standards dans les MIB. Ensuite, des fichiers de configuration permettraient d'adapter les différentes MIB présentes chez les constructeurs, rendant ainsi QMA interopérable même si les standards proposés dans les RFC ne sont pas respectés.

Pour l'administration de larges réseaux, lesquels nécessiteraient trop de temps pour récupérer toutes les informations, on pourrait diviser le réseau en sous réseaux, et pour chacun y affecter un serveur QMA. Puis un serveur principal récolterait les informations de chacun des QMA pour regrouper le tout. De plus, à grande échelle, une infrastructure *Remote Authentication Dial-In User Service (*RADIUS) pourrait être utilisée pour gérer l'authentification.

Enfin, comme l'a suggéré notre partenaire industriel Bell Canada, il serait vraiment intéressant d'étudier la faisabilité d'une intégration de QMA dans un outil professionnel de gestion de réseaux, tel *What's up* [39].

ANNEXE 1

CODE MATLAB DE L'ALGORITHME HOLT-WINTERS

Organisation du code

Le code utilisé est composé de cinq fichiers :

a. *Holt_winters* : implémente l'algorithme du même nom;
b. *Mse* : calcule l'erreur carrée moyenne (*mean square error*);
c. *Optimise* : calcule les coefficients de pondération optimaux pour minimiser l'erreur;
d. *Pred_f* : applique l'algorithme d'Holt_Winters sur une suite donnée et retourne l'erreur de prédiction;
e. *Simu* : teste l'algorithme d'Holt_Winters sur 1 000 séries.

Holt_winters

```
function x_prev = holt_winters( x, h, p, alpha, beta, gamma )
%applique la méthode de Holt-Winters (lissage exponentiel triple) pour
%prévoir les prochaines valeurs
%x : série de base
%h : horizon de prédiction
%p : période
%alpha, beta, gamma : coefficients de pondération
%  Detailed explanation goes here

x_prev=[x zeros(1,h)];
N=length(x);
S=zeros(1, N+h);
b=zeros(1, N+h);
I=zeros(1, N+h);

%http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc435.htm

%initialisation des suites (sauf S)
```

```
for i=1:p
    b(1)=b(1)+x(p+i)-x(i);
end
b(1)=b(1)/(p^2);

k=floor(N/p);
A = zeros(1,k);
for j=0:k
   A(j+1)=0;
   for i=1:p
   A(j+1)=A(j+1)+x(k*j+i);
   end
end

for i=1:p
    for j=0:k-1
    I(i)=I(i)+x(k*j+i)/A(j+1);
    end
I(i)=I(i)/k;
end


for i=2:N
if i-p > 0
    S(i)=alpha*(x(i)/I(i-p))+(1-alpha)*(S(i-1)-b(i-1));
else
    S(i)=alpha*(x(i)/I(i))+(1-alpha)*(S(i-1)-b(i-1));
end
    b(i)=gamma*(S(i)-S(i-1))+(1-gamma)*b(i-1);
if (i-p > 0)
    I(i)=beta*(x(i)/S(i))+(1-beta)*I(i-p);
end
end


i=N;
   for j=1:h
        x_prev(i+j)=(S(i)+j*b(i))*I(i-p+j);
   end
```

Mse

```
function erreur = mse(x, y)
%Calcule l'erreur carrée moyenne
%x : série 1
%y : série 2
L = length(x);
erreur =0;
if L ~= length(y)
    disp ('séries impossibles à comparer!!!');
```

```
else
    for i=1:L
    erreur = erreur + (x(i)-y(i))^2;
    end
    erreur = erreur/L;
end
```

Optimise

```
function abg = optimise(x, y, h, p)
%Calcule les coefficients de pondération optimaux pour minimiser
l'erreur
%carrée
%x : série à prévoir
%y : série de référence
%h : horizon de prédiction
%p : période

pas=0.1;
abg=zeros(1,3);
erreur = 10^10;

for alpha=0.1:pas:1
    for beta=0:pas:1
        for gamma=0:pas:1
           x_prev = holt_winters( x, h, p, alpha, beta, gamma );
           if mse(x_prev,y) < erreur
           abg(1)=alpha;
           abg(2)=beta;
           abg(3)=gamma;
           erreur = mse(x_prev,y);
           end
        end
    end
end
```

Pred_f

```
function erreur=pred_f

N=500;
range=100;
periode =100;
p=periode;
y=zeros(1,N+range);
tendance=zeros(1,N+range);
a=01;
b=800;
saison=zeros(1,N+range);
```

```
aleatoire =100*rand(1,N+range);

for i=1:N+range
    tendance(i) = a*i+b;
    q = floor(i/periode);
    r = i-q*periode;
    if (r<11)
        saison(i)=1;
    elseif (r>10) & (r<21)
        saison(i)=0.005*r+1;
    elseif (r>20) & (r<51)
        saison(i)=1.1;
    elseif (r>50) & (r<81)
        saison(i)=-0.005*r+1.355;
    else
        saison(i)=0.955;
    end
    saison(i)=1*saison(i);

    %saison(i)=sin(2*pi*i/100);
    y(i)=tendance(i)*saison(i)+aleatoire(i);
end

xx=zeros(1,N);
for i=1:length(xx)
   xx(i)=y(i);
end

h=4;
xx2=zeros(1,N+h);
for i=1:length(xx2)
   xx2(i)=y(i);
end

xx3=zeros(1,N-h);
for i=1:length(xx3)
   xx3(i)=y(i);
end


x_ref=zeros(1,N);

for i=1:N
    x_ref(i)=y(i);
end


temoin=[];
temoin2=[];

for j=N:h+1:N+range-h
```

```
%ces séries servent à calculer les coefficients alpha, beta et gamma
%optimaux, en se basant sur les valeurs passées il y a une période
temoin=[temoin y(j)];
temoin2=[temoin2 j];

x_ref(j)=y(j);

xx4=zeros(1,j-p);
for i=1:length(xx4)
   xx4(i)=x_ref(i);
end

xx5=zeros(1,j-p+h);
for i=1:length(xx5)
   xx5(i)=x_ref(i);
end
%plot(y);

%prediction maintenant

alpha=0;
beta=0;
gamma=0;
h=4;

% opt= optimise(xx, xx2, h, p); %ideal ms on a pas le deoit
% opt= optimise(xx3, xx, h, p); %bouh pas bien
opt= optimise(xx4, xx5, h, p); %ouais!!!! pas pire!!

alpha = opt(1);
beta=opt(2);
gamma=opt(3);

x_ref = holt_winters( x_ref, h, p, alpha, beta, gamma );

end

x_ref(N+range)=y(N+range);


serie_diff=zeros(1,N+range);
serie_diff=abs(y-x_ref);

%hold on
 %plot(x_ref, 'r');
%plot(y, 'k');
%plot(serie_diff, 'b');
%legend('Prédiction','Suite originale','Erreur');
%title('Prévision de Holt-Winters');
%plot(y, 'g');
% plot(temoin2, temoin, 'bo');
```

```
erreur = mse(y,x_ref)*(N+range)/range;
```

Simu

```
%parametres
clear all;
close all;
clc;

N=1000;
xx = zeros(1,N);
tic
for i=1:N
    xx(i)=pred_f;
end
toc
mean(xx)
```

ANNEXE 2

GUIDE D'INSTALLATION DE QMA

Cette annexe présente comment installer QMA. Elle comprend les étapes suivantes :

- Installation de Net-Snmp
- Configuration de la base de données
- Création des fichiers de logs
- Installation du WebService
- Configuration de l'Applet
- Configuration des clients

Installation de Net-Snmp

Pour sa communication avec le réseau par SNMP, QMA utilise la bibliothèque Net-Snmp[72]. L'outil suppose donc que celle-ci est installée sur le serveur, dans le répertoire d'installation par défaut (*C:\usr\bin*).

Configuration de la base de données

QMA ne peut évidemment pas fonctionner sans une base de données correctement configurée. Pour cela, un script est livré avec l'outil, permettant la génération automatique des bases de données requises. Le modèle des différentes tables est expliqué à l'annexe 4.

Création des fichiers de logs

QMA requiert deux fichiers de logs pour permettre à l'administrateur un suivi des opérations de l'outil. Deux fichiers doivent donc être créés, avec les droits de lecture et d'écriture pour le compte ASPNET :

a. C:\Temp\ log12345.txt;
b. C:\Temp\ traps.txt.

Installation du WebService

L'installation du WebService peut sembler être une opération déroutante pour quiconque n'y est pas habitué. C'est pourquoi il semble utile de préciser les quelques étapes à respecter :

- Après avoir recopié le répertoire WS_QMA de l'outil dans le répertoire d'IIS, il faut déclarer auprès du serveur Web, ce répertoire comme une application ASP, en cliquant sur le bouton *Create* dans le champ de propriétés, comme illustré sur la figure 76. Une fois ce bouton pressé, il faut vider le champ *Application name*.

- Il faut bien s'assurer que la version d'ASP utilisée dans l'onglet ASP .Net qui doit être 2.0.* comme le montre la figure 77.

- Enfin, il faut s'assurer que les droits d'exécutions sont correctement configurés pour ce répertoire. En particulier, les comptes *IUSR_NOMDUPC* et *IWAM_NOMDUPC* doivent avoir les droits de lecture et d'exécution.

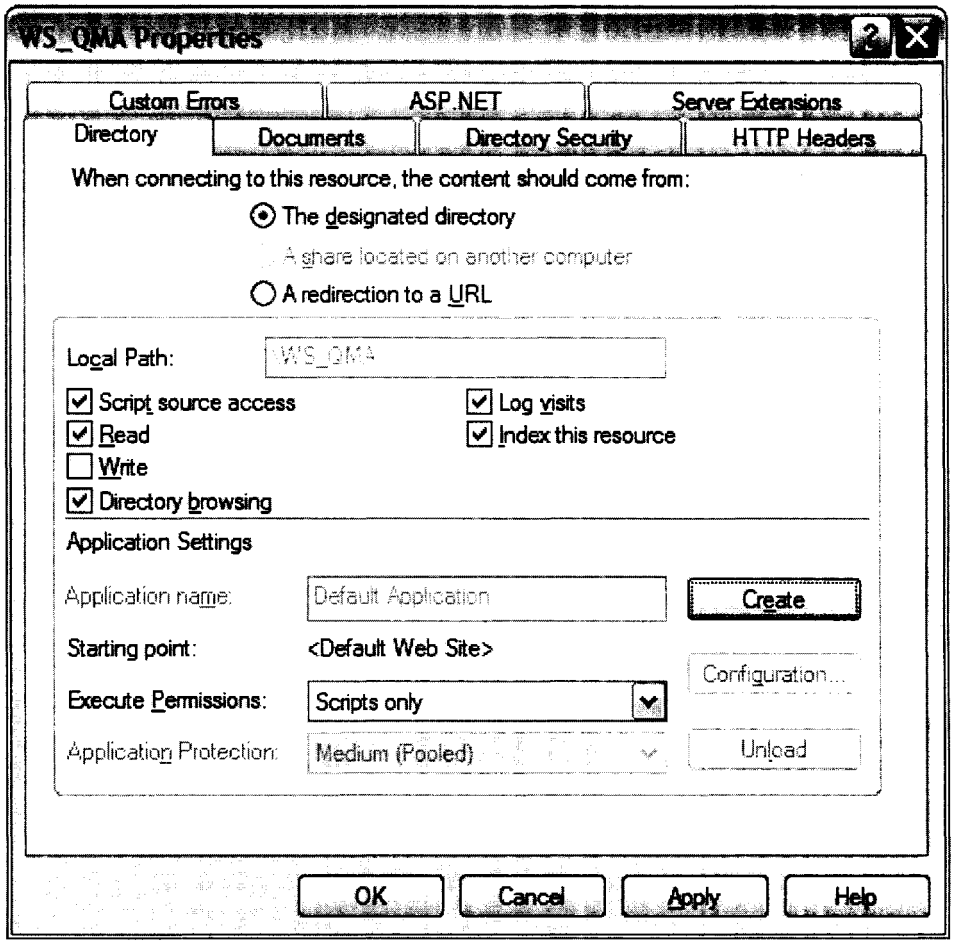

Figure 76 Configuration de IIS pour le WebService

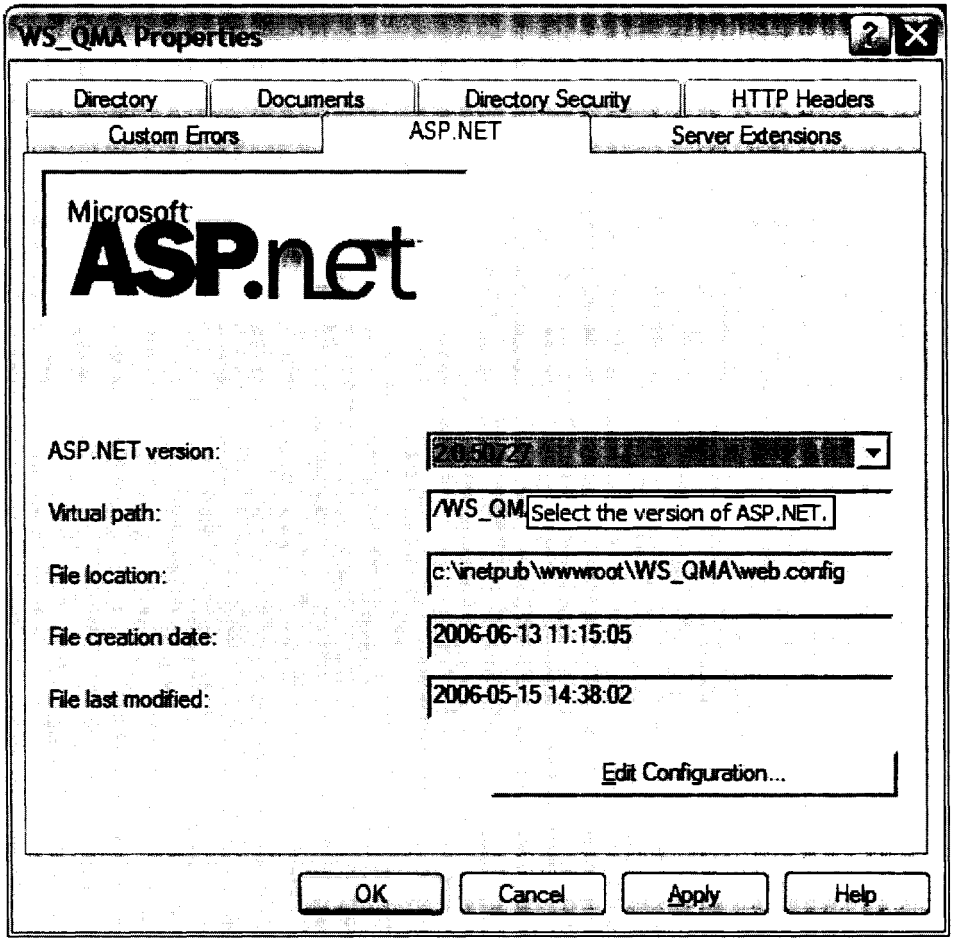

Figure 77 Vérification de la version du WebService

Configuration de l'applet

Avant de mettre l'applet à disposition des utilisateurs, il faut juste configurer l'adresse du WebService dans le code. Celui-ci n'est pas visible dans l'interface graphique pour des raisons de confidentialité. Certes, un usager insistant pourra le retrouver en analysant les paquets échangés depuis son poste, ou en décompilant l'applet, mais de telles actions requièrent déjà un effort particulier.

Il faut donc dans le code du projet préciser l'adresse IP où se trouve le WebService et le nom du répertoire web. Une recherche de la chaîne « *https* » permet notamment d'effectuer un remplacement automatique là des références vers le WebService.

Configuration des clients

L'utilisation d'une interface Web dispense l'usager de devoir installer un programme client spécifique. Pour autant certaines configurations sont nécessaires. Tout d'abord, le Framework .Net 2.0 *Software Development Kit* (SDK) doit être installé. Celui-ci est disponible gratuitement [77]. Il faut ensuite ajouter la bibliothèque ZedGraph [70] dans la liste des *assemblies* de .Net à l'aide du menu de configuration de celui-ci comme le montre la figure 78.

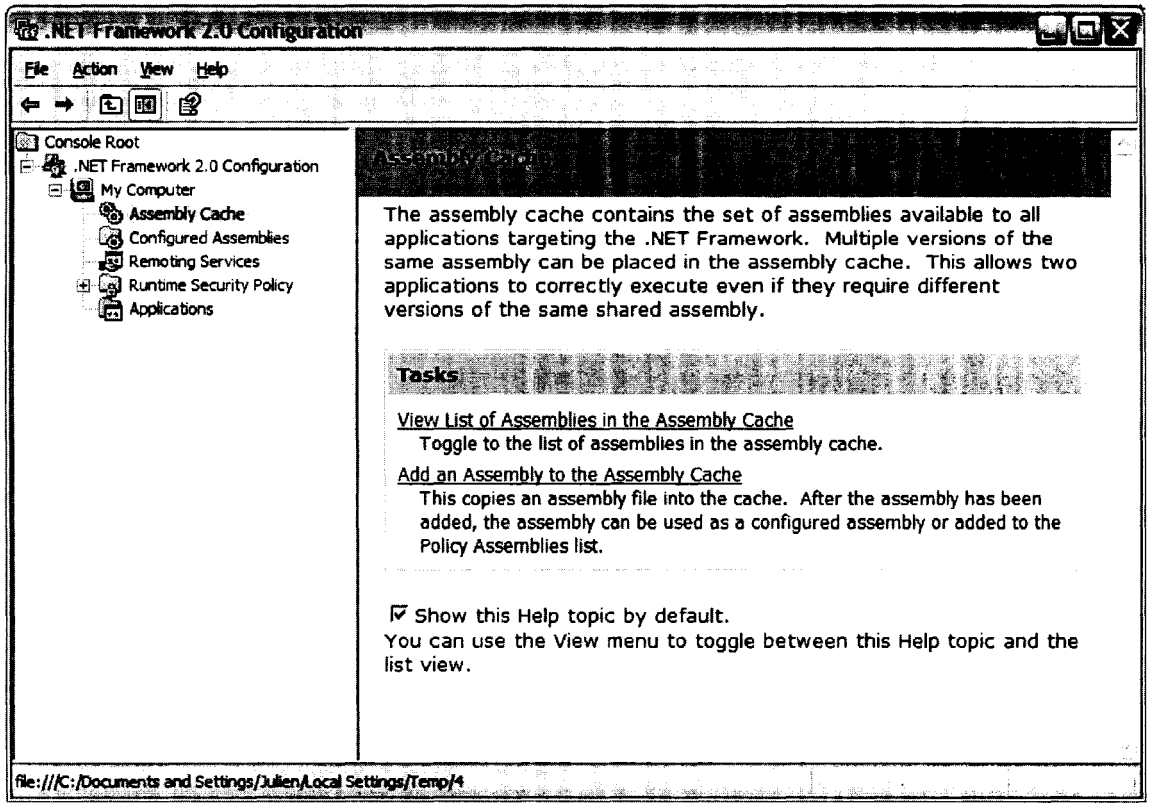

Figure 78 Configuration du Famework .NET

Enfin, il faut répertorier le site où se trouve l'Applet comme un site de confiance (*Trusted Sites*) dans Internet Explorer puis configurer le Framework pour permettre à ces sites d'avoir le niveau de confiance maximum (*Full Trust*), dans le champ *Adjust Zone Security* comme présenté sur la figure 79.

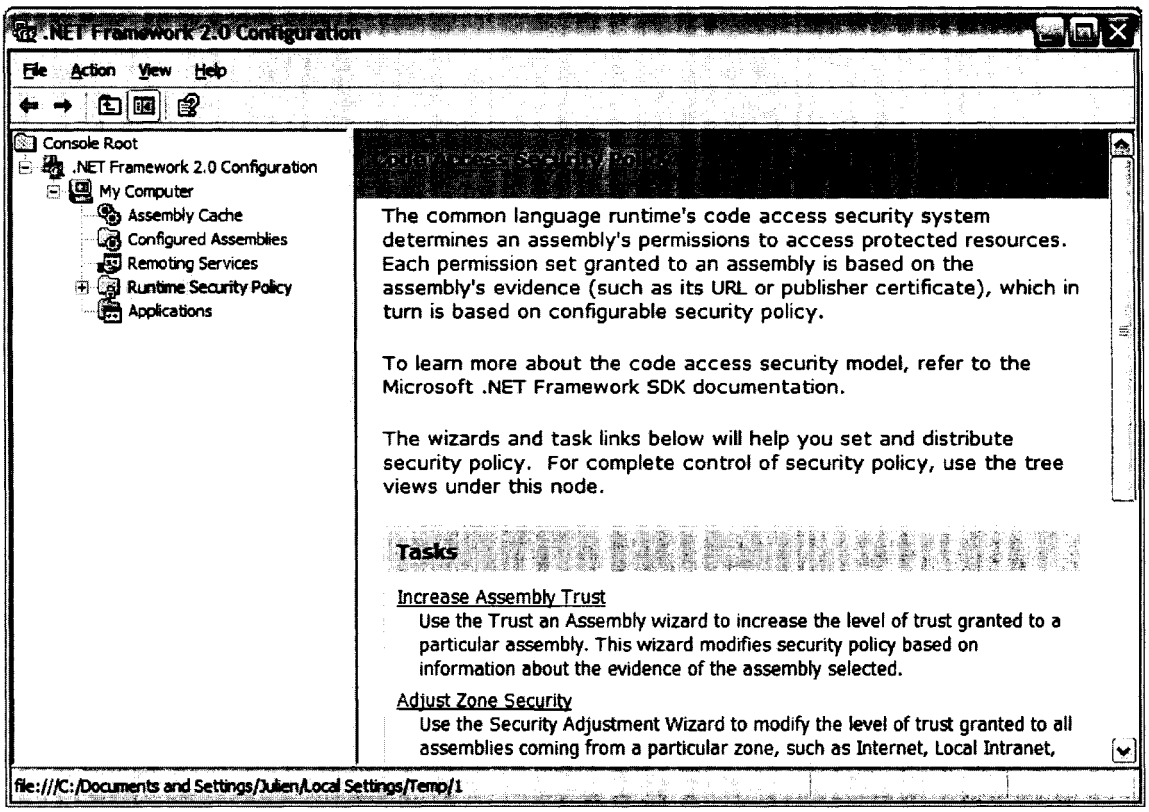

Figure 79 Ajustement de la sécurité dans .Net

ANNEXE 3

GUIDE D'UTILISATION DE QMA

Cette annexe présente comment utiliser QMA. Elle s'organise ainsi : tout d'abord les configurations préalables sont explicitées. Puis les fonctionnements du contrôleur du serveur puis de l'applet sont détaillés.

Préalable

Avant d'utiliser QMA, quelques configurations doivent avoir été effectuées.

Configuration côté réseau

QMA communique avec le réseau via SNMP v3. Il faut donc que chaque routeur soit configuré pour traiter les requêtes SNMP. Ainsi, un utilisateur avec un mot de passe doit être défini. Sous l'IOS Cisco, voici la démarche à suivre :

- Créer un groupe SNMPv3. Par exemple pour créer un groupe du nom de *project* :

```
PE2(config)#snmp-server group project v3 auth
```

- Créer un compte SNMPv3. Par exemple pour créer un compte du nom de *student* avec comme mot de passe *password* :

```
PE2(config)#snmp-server user student project v3 auth md5
password
```

De même, pour la surveillance du réseau, il faut configurer les routeurs dont on souhaite écouter les *traps*. La procédure est la suivante :

- On précise les catégories de *traps* souhaitées : ici on souhaite les traps SNMP standards ainsi que les *traps* propres au MPLS TE :

```
PE2(config)#snmp-server enable traps snmp
```

```
PE2(config)#snmp-server enable traps mpls traffic-eng
```

- Enfin, on définit l'adresse où se trouve le serveur de l'outil, où envoyer les *traps*, (par exemple *172.116.102.33*) :

```
PE2(config)#snmp-server host 172.116.102.33 informs version 2c
public
```

Configuration de la base de données

Il est nécessaire que la table *Router* ainsi que la table *Link* soient préalablement remplies. Il ne s'agit a priori pas de configurations censées changer fréquemment. De plus, une méthode dans la partie QoS de QMA permet de mettre à jour automatiquement la table *Router*.

Utilisation du contrôleur du serveur

Le contrôleur du serveur un programme situé sur le serveur. Il ne doit donc être accessible que pour l'administrateur, qui dispose d'un accès physique à la machine. Outre la partie QoS, il sert à surveiller le réseau. L'onglet propre à MPLS est l'onglet *MPLS Config*, illustré à la figure 80.

Onglet MPLS Config

Cet onglet sert à la surveillance du réseau. Il est nécessaire de cliquer sur *Start Monitoring* pour que celle-ci soit effective. Tant qu'elle n'est pas lancée ici aucune surveillance n'a lieu. Une fois démarrée, les différents paramètres s'affichent :

a. Possibilité d'écouter les *traps* (*Listen to traps*);
b. Possibilité de collecter les statistiques des tunnels (*Stats collection*) en précisant la fréquence (*Frequency*).

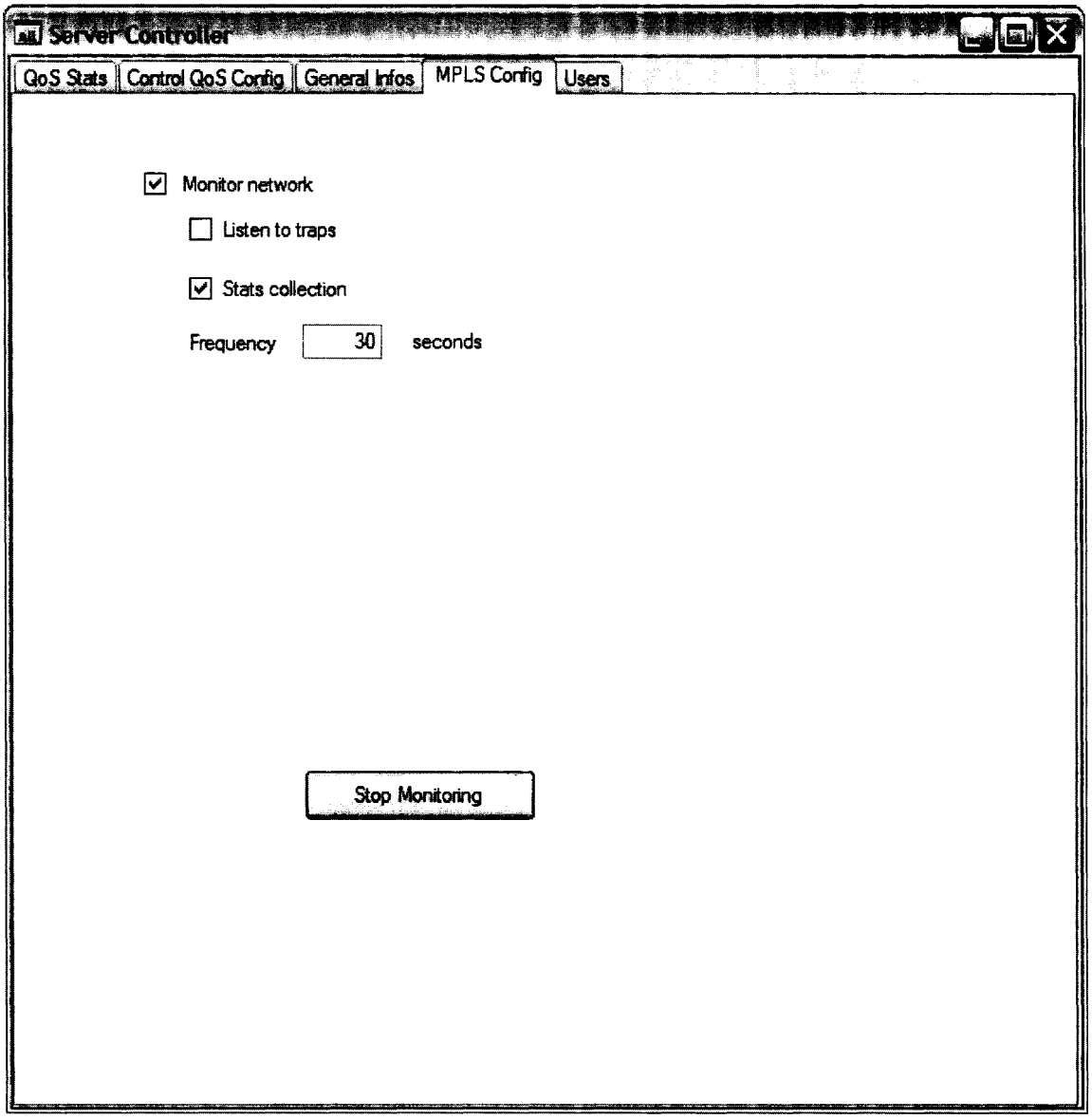

Figure 80 Onglet *MPLS Config*

Il est possible qu'une erreur survienne lors du démarrage de la surveillance de type illustré à la figure 81.

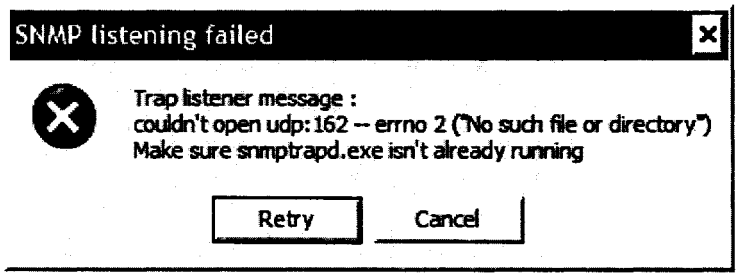

Figure 81 Erreur lors du démarrage de la surveillance

Ce message signifie que QMA démarre le processus *snmptrapd.*exe pour écouter les *traps* mais que celui-ci s'exécute déjà. Il faut donc tuer le processus existant puis cliquer sur *Retry.*

Onglet Users

Cet onglet permet de gérer les différents utilisateurs. Il en existe deux classes : les administrateurs qui disposent de tous les droits, et les usagers « normaux » qui sont privés de certaines fonctionnalités avancées, détaillées plus tard. Un onglet est dédié à la création d'utilisateurs (figure 82), l'autre à la modification de mot de passe et la suppression de mots de passe (figure 83). Dans le second onglet, la liste des différents utilisateurs se retrouve automatiquement en cliquant sur le bouton *Refresh.*

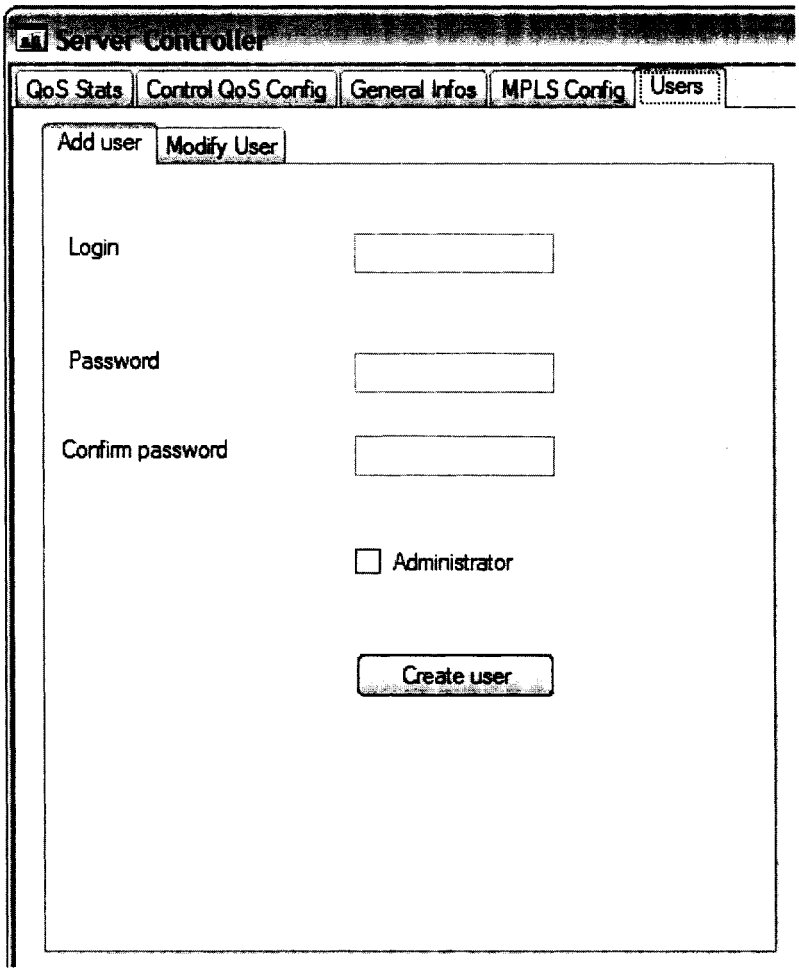

Figure 82 Onglet *User : création d'utilisateurs*

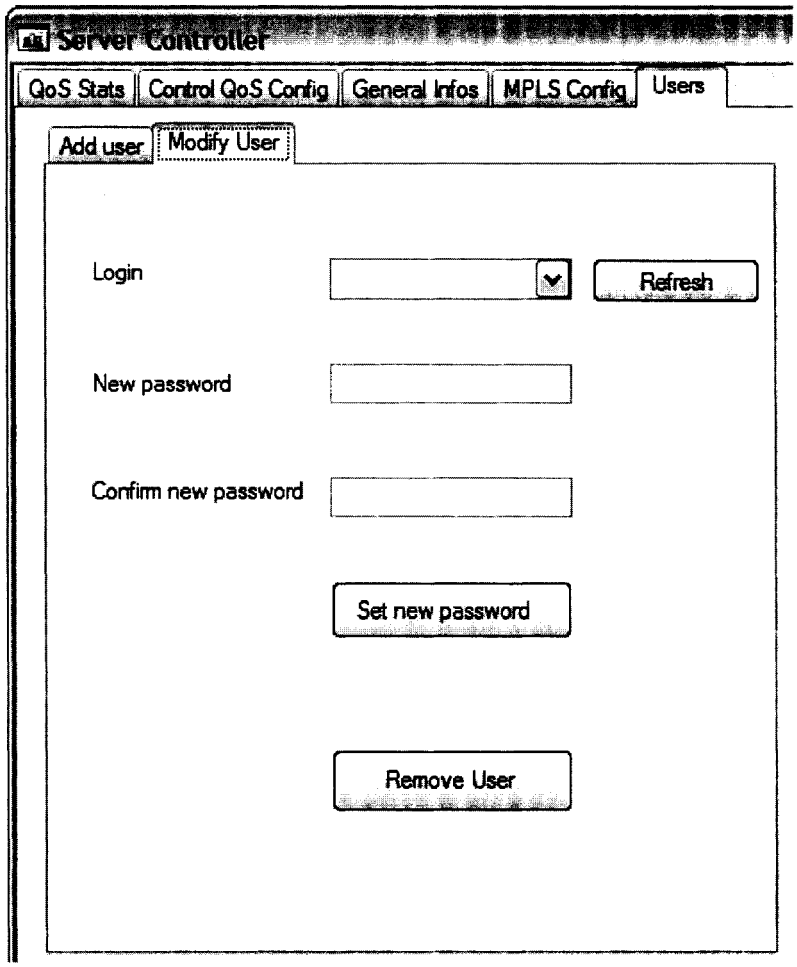

Figure 83 Onglet *User : modification d'utilisateurs*

La modification d'un utilisateur (suppression du compte ou changement de mot de passe) ne peut être réalisée qu'à condition que celui-ci ne soit pas actuellement connecté. Pour s'assurer de cela, un délai pouvant aller jusqu'à trois minutes est nécessaire au contrôleur de serveur entre la déconnexion effective de l'usager et la possibilité de modifier son compte.

Utilisation de l'applet

L'applet est organisée en onglets.

Onglet Authentification

Lors du chargement de l'applet, le premier onglet présenté est celui de l'authentification. Il est illustré figure 84 L'usager doit rentrer son nom d'utilisateur et le mot de passe puis peut cliquer sur un autre onglet. Si l'authentification se solde par un échec l'usager est automatiquement redirigé sur cet onglet.

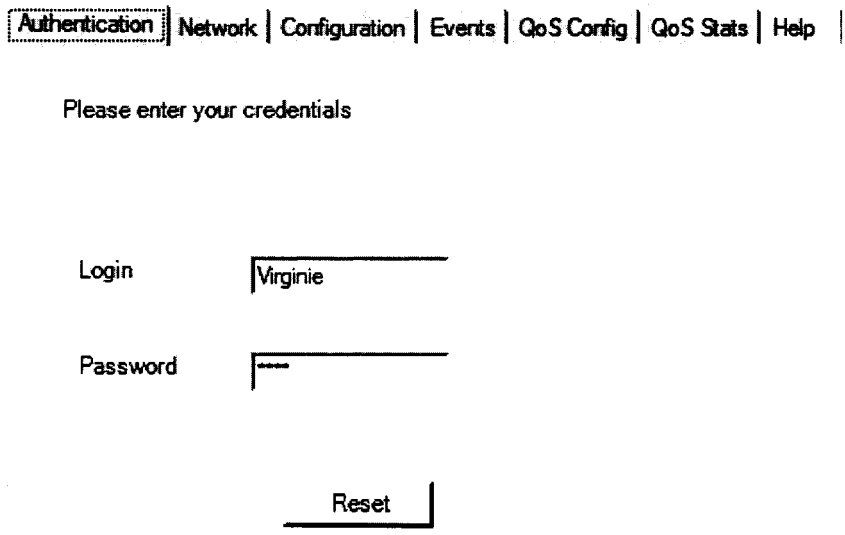

Figure 84 Onglet *Authentication*

Onglet Network

Cet onglet présente les informations du réseau : il est composé d'un menu, de champs d'affichage des informations (2 *listbox* et 2 *textbox*) et de la représentation du réseau. Un aperçu visuel est fourni figure 85. Listons ses fonctionnalités :

Affichage de la topologie du réseau

En cliquant sur « *General Information* », la *ListBox1* propose trois alternatives :

a. Lister les adresses IP du réseau (*Router by IP*) : en cliquant, la *ListBox2* dresse la liste des adresses du réseau et en en sélectionnant une, les informations du routeur possédant cette IP s'affichent dans le *TextBox1*. Le routeur est alors surligné en vert;

b. Lister les noms des routeurs (*Router by Name*) : en cliquant, la *ListBox2* dresse la liste des noms des routeurs et en en sélectionnant une les informations de ce routeur s'affichent dans la *TextBox1*. Le routeur est alors surligné en vert;

c. Lister les liens (*Physical links*) : en cliquant, la ListBox2 liste les liens par IP d'origine, IP de destination. Le lien sélectionné est surligné en vert et ses informations (type de lien, bande passante) sont affichées dans la *TextBox1*.

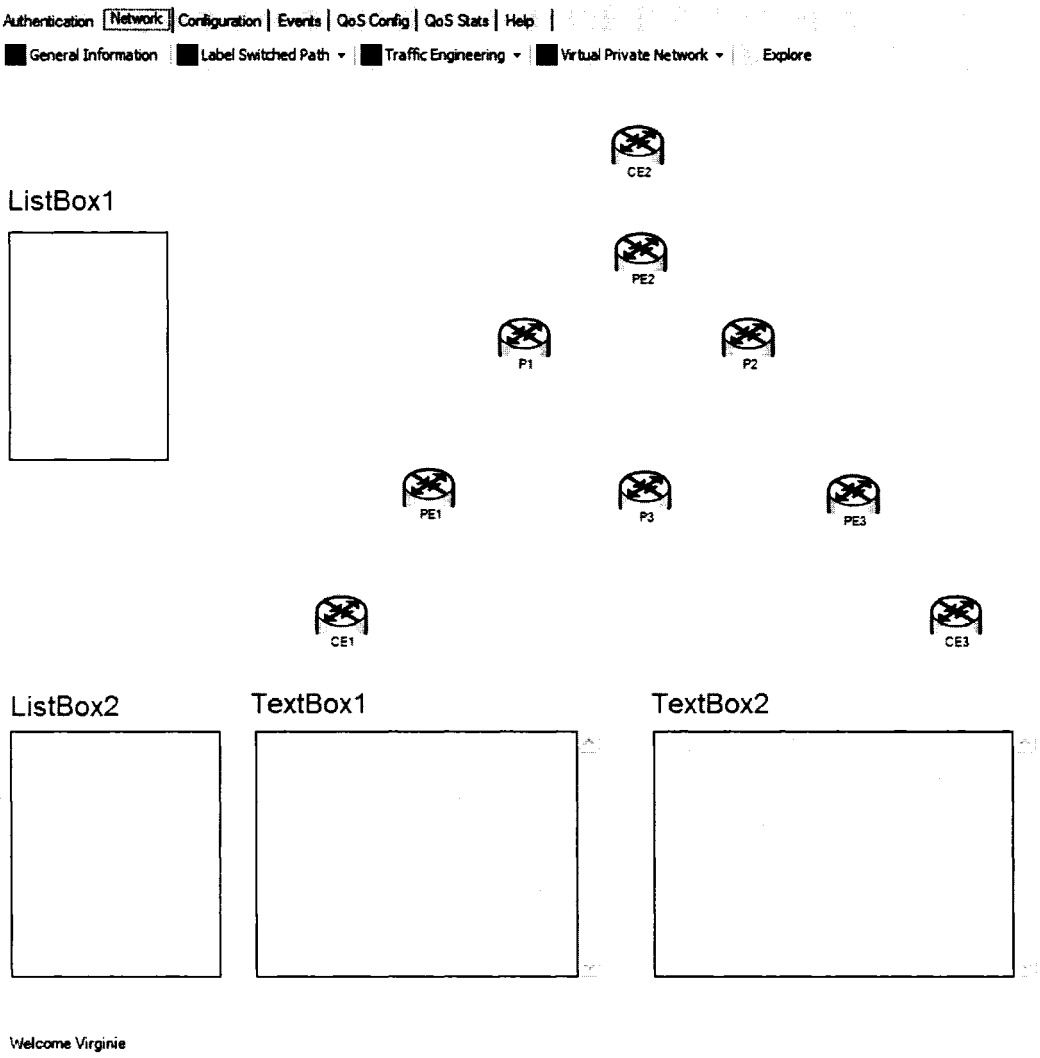

Figure 85 Onglet *Network*

Affichage des LSP

Le menu permet deux alternatives :

a. Afficher les LSP par origine-destination (*LSP by Ingress-Egress*) : en sélectionnant cette option, la *ListBox1* liste les différents couples origine/destination desservis par les LSP. En sélectionnant un des items, la *ListBox2* affiche les différents LSP pour les origine/destination données. Quand un LSP particulier est sélectionné, il est dessiné

sur le schéma du réseau et sa décomposition en segments est présentée dans la *TexBox1*.

b. Afficher les LSP par Routeur (*LSP by Router*) : en sélectionnant cette option, la *ListBox1* liste les différents routeurs par lesquels passent des LSP. En sélectionnant un des items, la *ListBox2* affiche les différents LSP passant par le routeur sélectionné. Quand un LSP particulier est sélectionné, il est dessiné sur le schéma du réseau et sa décomposition en segments est présentée dans la *TexBox1*.

Affichage des Tunnels

Le menu permet deux alternatives :

a. Afficher les tunnels par origine-destination (*Tunnel by Ingress-Egress*) : en sélectionnant cette option, la *ListBox1* liste les différents couples origine/destination desservis par les tunnels. En sélectionnant un des items, la *ListBox2* affiche les différents tunnels pour les origine/destination données. Quand un tunnel particulier est sélectionné, ses propriétés sont présentées dans la *TexBox1*. S'il est opérationnel, il est dessiné sur le schéma du réseau, sa décomposition en segments est présentée dans la TextBox1 et ses statistiques sont affichées dans un contrôle utilisateur à l'emplacement de la *TextBox2*.

b. Afficher les tunnels par Routeur (*Tunnel by Router*) : en sélectionnant cette option, la *ListBox1* liste les différents routeurs par lesquels passent des tunnels. En sélectionnant un des items, la *ListBox2* affiche les différents tunnels qui passent par le routeur sélectionné. Quand un tunnel particulier est sélectionné, ses propriétés sont présentées dans la *TexBox1*. S'il est opérationnel, il est dessiné sur le schéma du réseau, sa décomposition en segments est présentée dans la *TextBox1* et ses statistiques sont affichées dans un contrôle utilisateur à l'emplacement de la *TextBox2*.

Affichage des VPN

Le menu permet deux alternatives :

a. Afficher les VPN par nom (*VPN by Name*): en cliquant sur cette option, la *ListBox1* liste les différents VPN présents. En sélectionnant un des VPN, la *ListBox2* dresse la liste des VRF associés à ce VPN. En sélectionnant un VRF particulier, les informations de bases et les interfaces configurées pour celui-ci s'affichent dans la *TextBox1*, et les routes dans la *TextBox2*.
b. Afficher les VRF par routeur (*VPN by Router*): en cliquant sur cette option, la *ListBox1* liste les différents routeurs où des VRF sont configurés. En sélectionnant un des routeurs, la *ListBox2* dresse la liste des VRF configurés. En sélectionnant un VRF particulier, les informations de bases et les interfaces configurées pour celui-ci s'affichent dans la *TextBox1*, et les routes dans la *TextBox2*.

Rafraîchissement des informations

En cliquant sur le bouton *Explore* un rafraîchissement des informations est démarré. Une fois celui-ci effectué, une bulle d'information apparaît dans la barre de notifications pour signaler la fin de l'opération. Ceci n'est valable que si QMA est configuré pour vérifier de nouveaux évènements, comme expliqué dans la section suivante.

Onglet Configuration

Cet onglet comprend deux volets : la configuration au niveau de l'applet et celle au niveau du serveur, réservée aux administrateurs. Il est présenté figure 86.

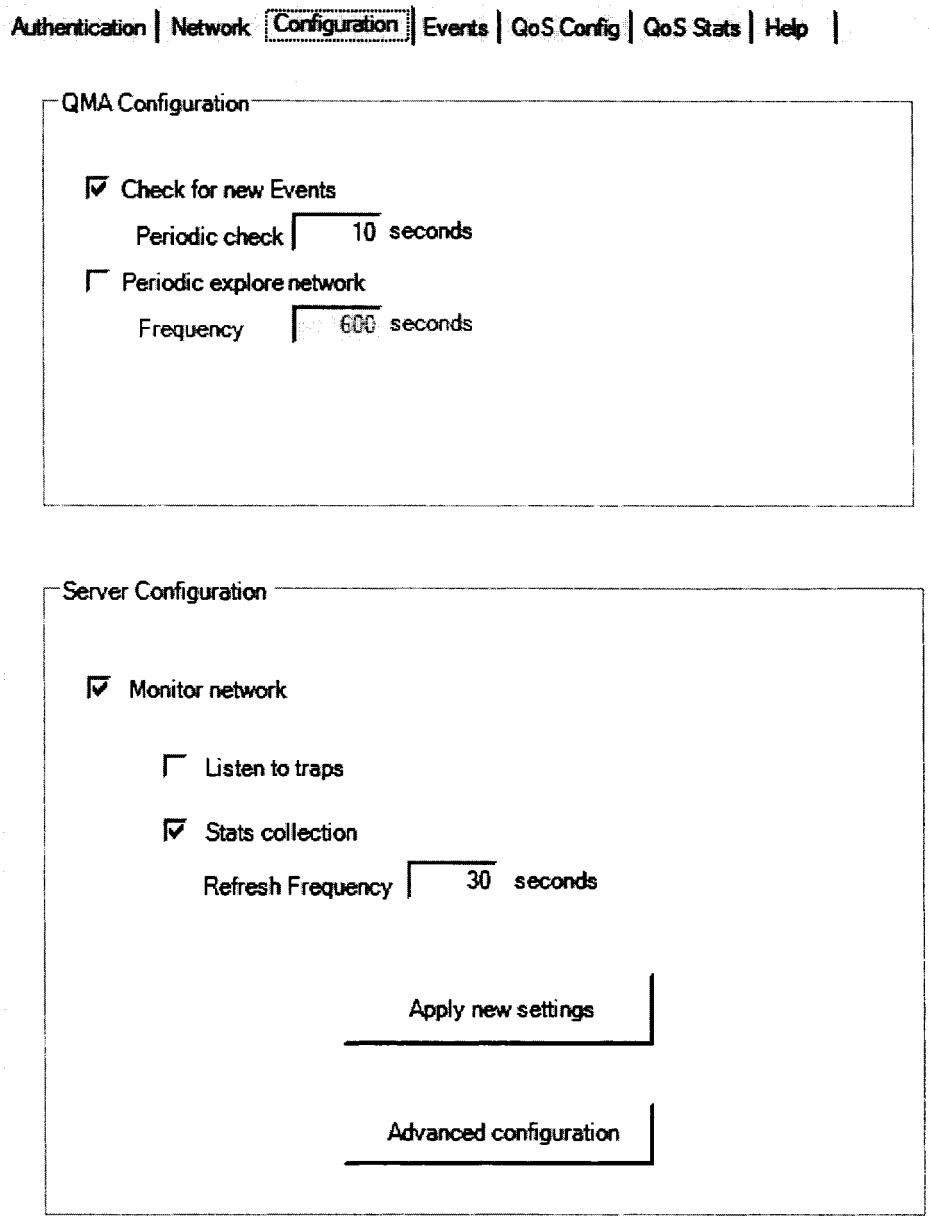

Figure 86 Onglet *Configuration*

Les paramètres de configuration de l'applet (*QMA Configuration*) comprennent :

a. La possibilité de vérifier de nouveaux évènements survenus (*Check for new Events*) en précisant la fréquence de vérification (*Periodic check*). Le cas

échéant, une bulle apparaît dans la barre de notification, comme le montre la figure 87.

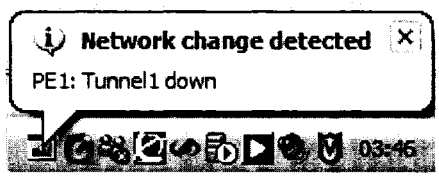

Figure 87 Exemple de notification

b. La possibilité de lancer périodiquement un rafraîchissement complet de la base de données (*Periodic explore network*) en précisant la fréquence (*Frequency*). L'utilisateur est notifié lors de la fin de l'opération, à durée variable dépendamment de la taille du réseau. Il faut donc veiller à ne pas mettre une fréquence inférieure ou même trop proche du temps nécessaire à la tâche.

Chaque modification de ces paramètres est prise en compte instantanément.

Les paramètres de configuration du serveur (*Server Configuration*) comprennent :

a. Le fait ou non de faire de la surveillance (Monitor network);
b. La possibilité d'écouter les *traps* (*Listen to traps*);
c. La possibilité de collecter des statistiques sur les tunnels MPLS TE (*Refresh frequency*) en précisant la fréquence (*Refresh Frequency*).

Comme il s'agit des paramètres du serveur, chaque modification de ces paramètres doit être validée par le bouton *Apply new settings* avant d'être effective. Le bouton *Advanced Configuration* ouvre un onglet explicité ultérieurement.

Onglet Event

Cet onglet a pour rôle de lister les différents évènements du plus récent au plus ancien comme le montre la figure 88.

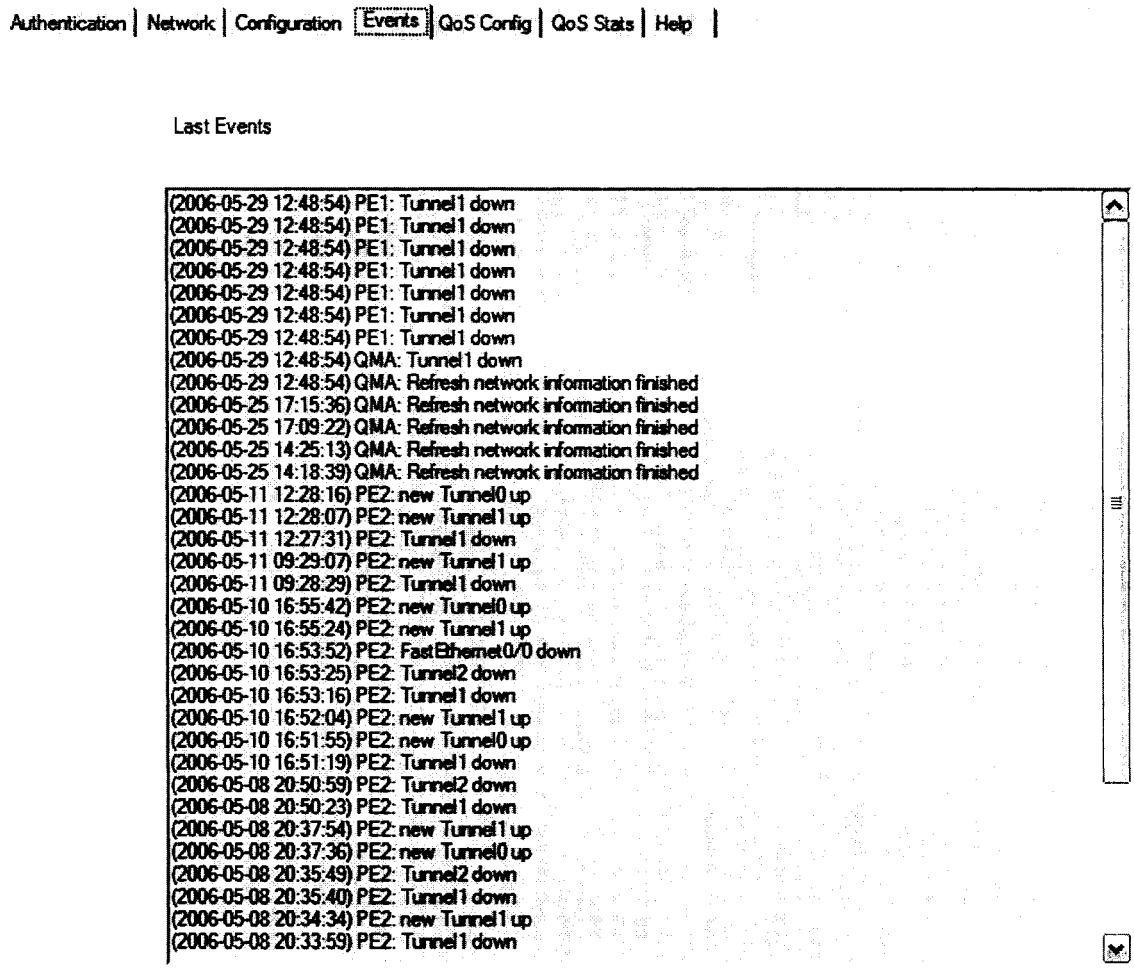

Figure 88 Onglet *Events*

Onglet AdvancedConfig

Cet onglet, réservé aux administrateurs, contient les paramètres avancés. Il se divise lui-même en trois sous onglets :

a. Le premier est la configuration des différents paramètres, exposés à la figure 89 : nom du serveur de la base de données, nom des deux bases, nom d'utilisateur et mot de passe, ainsi que les crédits nom d'utilisateur/mot de passe pour SNMP. Pour rendre les modifications effectives, il faut valider avec le bouton *Upload Settings*. Les boutons *Load Settings from a file* et *Save Settings in a file* permettent respectivement de charger la configuration d'un fichier texte et de la sauvegarder.

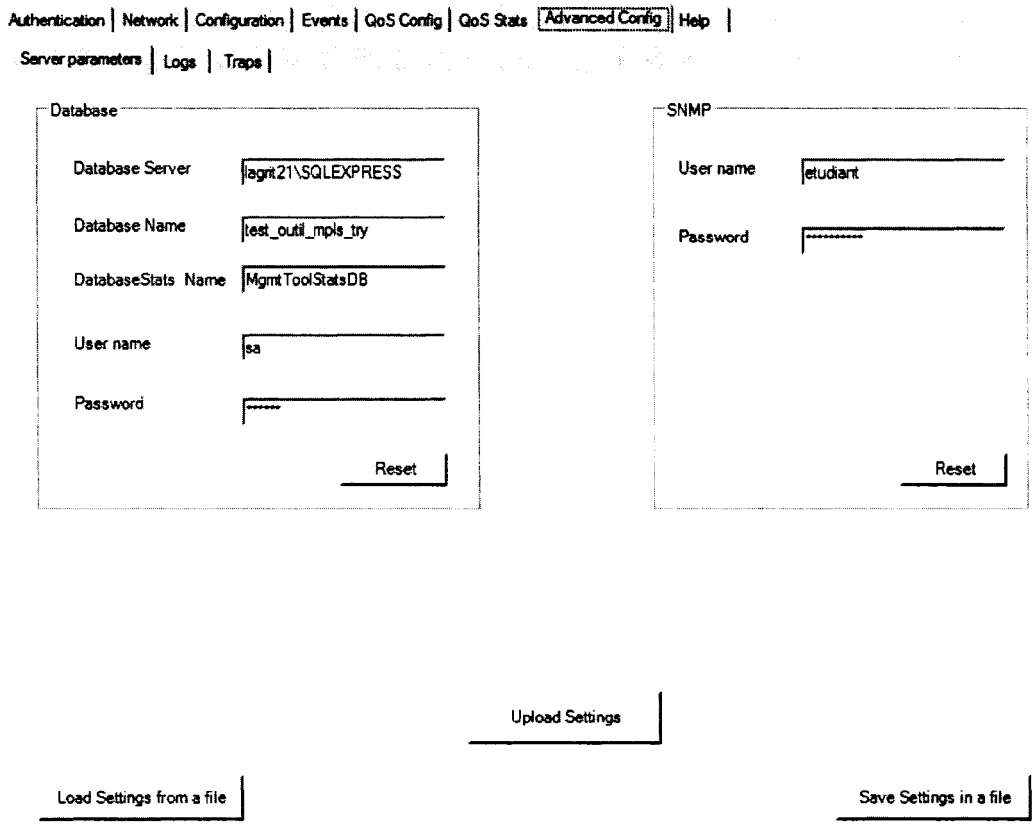

Figure 89 Onglet *Advanced Config*

b. L'onglet *Logs* affiche le fichier de log de QMA, répertoriant les opérations effectuées.
c. L'onglet *Traps* liste les *traps* reçues ; il peut être pratique si l'utilisateur souhaite avoir plus d'informations que ce que l'onglet *Events* contient.

ANNEXE 4

MODÈLE DE BASE DE DONNÉES DE QMA

Introduisons tout d'abord quelles données sont nécessaires à QMA et comment celles-ci sont organisées. Deux catégories de données sont utilisées dans QMA :

a. Données propres au réseau : données collectées par QMA en utilisant SNMP ; Celles-ci comprennent des informations de configuration et des statistiques;
b. Données dites « de gestion » : données crées par QMA et nécessaires à son bon fonctionnement.

Données du réseau

Le cœur de QMA est de fournir à l'utilisateur des informations sur un réseau MPLS. La collecte d'informations est donc centrale. Pour mener à bien cette opération, QMA a besoin de connaître la topologie du réseau. Deux tables doivent donc être préalablement remplies :

a. *Router_MPLS* : contient les informations nécessaires pour chaque routeur MPLS considéré;
b. *Link* : contient les informations sur les liens reliant les routeurs.

Ces tables servent de référence pour QMA, et ne sont donc que rarement modifiées.

À partir des éléments contenus dans ces tables, QMA peut récolter les informations désirées, regroupées dans sept tables :

a. *Segment* : contient les informations des segments MPLS;
b. *LSP* : permet de lister tous les LSP;
c. *Tunnel* : informations relatives aux tunnels MPLS TE;

d. *VPN* : permet de lister les VPN;
e. *VRF* : permet de lister les VRF;
f. *VRF_IF* : contient les informations pour chaque interface configurée dans le cadre d'un VRF;
g. *VRF_ROUTE* : contient les informations pour chaque route configurée dans le cadre d'un VRF.

En pratique, les sept tables précédentes sont doublées, pour éviter de priver l'utilisateur de la consultation de ces tables pendant que celles-ci sont mises à jour, laps de temps pendant lesquelles la base de données est considérée dans un état instable. Ainsi, il y a la table Segment0 et Segment1, LSP0 et LSP1 etc. ... Pendant que la table Segment0 est remplie, l'utilisateur peut donc toujours consulter la table Segment1 et inversement.

Enfin des tables de statistiques sont créées dynamiquement, une par Tunnel dont on souhaite évaluer le trafic. Les tables se nomment *Tunnel_index_du_tunnel.*

Examinons en détail chacune des tables.

Router_MPLS

Router_MPLS contient les informations nécessaires pour chaque routeur MPLS considéré. Il ne s'agit pas à proprement parler une table, mais une vue de la table Router, rempli par l'outil pour la QoS. En effet, celui-ci nécessite une connaissance précise des routeurs, notamment la liste et les informations de chaque interface et sous-interface. Il a donc été jugé préférable de reprendre cette table Router, en élaguant les informations superflues dans le cadre de MPLS. D'où la vue. Chaque entrée correspond à une entrée dans la MIB interfaces. Sa structure est illustrée au tableau I.

Tableau I

Vue Router_MPLS

Nom de la colonne	Type de données
RouterID	int
IP	nvarchar(15)
Node	nvarchar(50)
Name	nvarchar(50)
Interface_index	nvarchar(10)
Lsr_id	nvarchar(50)

Les colonnes sont les suivantes :

a. *RouterID* : identifiant dans la base de données : clé primaire;
b. *IP* : adresse IP de l'interface ou de Loopback;
c. *Node* : nom du routeur;
d. *Name* : nom de l'interface;
e. *Interface_index* : index de l'interface au sein de la MIB Interfaces [78];
f. *Lsr_id* : identifiant du routeur utilisé dans la MIB MPLS-TE [28].

Un exemple de remplissage pour les routeurs PE1 et PE3 est proposé au tableau II.

Tableau II

Exemple de Router_MPLS

Router_ID	IP	Node	Name	Interface_index	Lsr_id
28	172.20.1.14	PE1	FastEthernet2/1	4	2887056907
30	172.20.254.11	PE1	Loopback0	6	2887056907
81	172.20.1.34	PE2	FastEthernet0/0	5	2887056908
83	172.20.1.22	PE2	FastEthernet3/0	7	2887056908
86	172.20.254.12	PE2	Loopback0	10	2887056908
132	172.20.1.26	PE3	FastEthernet0/0	3	2887056909
133	172.20.2.1	PE3	FastEthernet0/1	4	2887056909
135	172.20.1.18	PE3	FastEthernet6/0	6	2887056909
142	172.20.254.13	PE3	Loopback0	13	2887056909

Link

Link contient les informations sur les liens reliant les routeurs. Chaque entrée correspond à un lien physique unilatéral entre deux routeurs. Sa structure est illustrée au tableau III.

Tableau III

Table Link

Nom de la colonne	Type de données
Id	Int
IP_Source	nvarchar(15)
IP_Dest	nvarchar(15)
Type	nvarchar(50)
Speed	nvarchar(50)

Les colonnes sont les suivantes :

a. *Id* : identifiant dans la base de données : clé primaire;
b. *IP_Source* : adresse IP de l'origine du lien;
c. *IP_Dest* : adresse IP de la destination du lien;
d. *Type* : type du lien;
e. *Speed* : bande passante du lien, en bits par seconde.

Un exemple de remplissage pour les routeurs PE1 et PE3 est proposé au tableau IV.

Tableau IV

Exemple Link

Id	IP_Source	IP_Dest	Type	Speed
1	172.20.1.29	172.20.1.30	FastEthernet	100000000
3	172.20.1.30	172.20.1.29	FastEthernet	100000000

Segment0, Segment1

Segment contient les informations sur les segments composant les LSP. Il permet donc d'en avoir une vue détaillée. Chaque entrée correspond à un segment entre deux routeurs. Sa structure est illustrée au tableau V.

Tableau V

Table Segment

Nom de la colonne	Type de données
Id	int
LSP	int
Link	int
Destination	nvarchar(50)
Label	nvarchar(50)
In_Label	nvarchar(10)
In_Interface	nvarchar(10)
Node	nvarchar(50)
Pred	int
Reference	nvarchar(50)
Segment_order	int

Les colonnes sont les suivantes :

a. *Id* : identifiant dans la base de données : clé primaire;
b. *LSP* : identifiant du LSP auquel ce segment est rattaché (cf. table LSP);
c. *Link* : identifiant du lien par lequel passe le segment (cf. table Link);
d. *Destination* : nom du routeur destination du LSP;
e. *In_Label* : label du segment précédent (segment entrant);

f. *In_Interface* : index de l'interface (cf. table Router_MPLS) par lequel arrive le segment précédent;
g. *Node* : nom du routeur depuis lequel part le segment;
h. *Pred* : valeur à 0 ou 1 qui indique si ce segment possède ou non un segment le précédant;
i. *Reference* : index du segment dans la MIB MPLS-LSR [29];
j. *Segment_order* : détermine l'ordre du segment dans le LSP (utile pour afficher les segments dans l'ordre).

Un exemple de remplissage pour un LSP entre PE2 et PE3 est proposé au tableau VI.

Tableau VI

Exemple de table Segment

Id	LSP	Link	Destination	Label	In_Label	In_Interface	(Suite)
35349	205	16	PE3	55	NULL	NULL	
34917	205	2	PE3	POP	56	1	
35146	205	7	PE3	56	55	2	

(Suite)	Node	Pred	Reference	Segment_order
	PE2	0	NULL	1
	P3	1	1676582204	3
	P1	1	1678910528	2

Ce LSP (numéro 205) est ainsi composé de trois segments :

De PE2 à P1 avec le label 55

De P1 à P3 avec le label 56

De P3 à PE2 sans label (POP).

LSP0, LSP1

La table LSP permet d'avoir une vue d'ensemble de tous les LSP. Chaque entrée correspond à un LSP. Elle s'organise de la manière illustrée tableau VII.

Tableau VII

Table LSP

Nom de la colonne	Type de données
Id	int
Ingress	nvarchar(50)
Egress	nvarchar(50)
Tunnel_Id	int
First_Segment	int

Les colonnes sont les suivantes :

a. *Id* : identifiant du LSP dans la base de données (clé primaire);
b. *Ingress* : nom du routeur origine du LSP;
c. *Egress* : nom du routeur destination du LSP;
d. *Tunnel_Id* : dans le cas où le LSP est un tunnel TE, identifiant de ce tunnel
e. *First_Segment* : utile seulement dans le cadre d'un tunnel, valeur servant à indiquer si le premier segment a été traité ou non.

Le tableau VIII donne un exemple de deux LSP, l'un tunnel l'autre non.

Tableau VIII

Exemple de Table LSP

Id	Ingress	Egress	Tunnel_Id	First_Segment
203	P1	PE3	0	NULL
205	PE2	PE3	465	1

Tunnel0, Tunnel1

La table tunnel contient les informations de configuration des tunnels MPLS-TE. Chaque entrée correspond à un tunnel. Elle s'organise de la manière illustrée tableau IX.

Tableau IX

Table Tunnel

Nom de la colonne	Type de données
Id	int
Name	nvarchar(50)
Descr	nvarchar(50)
Ingress	nvarchar(50)
Egress	nvarchar(50)
Setup_Priority	nvarchar(50)
Holding_Priority	nvarchar(50)
BW	nvarchar(50)
Affinity	nvarchar(50)
Admin_Status	nvarchar(50)
Oper_Status	nvarchar(50)
Total_Uptime	nvarchar(50)
Mib_index	nvarchar(50)
IfIndexnvarchar	nvarchar(50)

Les colonnes sont les suivantes :

a. *Id* : identifiant du tunnel dans la base de données : clé primaire;
b. *Name* : nom du tunnel;
c. *Descr* : description du tunnel;
d. *Ingress* : nom du routeur d'origine du tunnel;
e. *Egress* : nom du routeur destination du tunnel;
f. *Setup_Priority* : priorité d'établissement du tunnel;
g. *Holding_Priority* : priorité de maintien du tunnel;
h. *BW* : bande passante réservée du tunnel;

i. *Affinity* : affinité du tunnel;
j. *Admin_Status* : statut administratif du tunnel;
k. *Oper_Status* : statut opérationnel du tunnel;
l. *Total_Uptime* : temps pendant lequel le tunnel est resté opérationnel;
m. *Mib_index* : index du tunnel dans la MIB MPLS TE [28];
n. *IfIndex* : index de l'interface dans la MIB Interfaces.

Deux tunnels sont présentés au tableau X.

Tableau X

Exemple de table Tunnel

Id	Name	Descr	Ingress	Egress	Setup_ Priority	Holding_ Priority	(suite)
465	Tunnel1	Tunnel1 TE vers PE3	PE2	PE3	3	3	
467	Tunnel2	PE1_t2	PE1	PE3	5	5	

(suite)	BW	Affinity	Admin_ Status	(suite)
	25000000	0	up	
	0	0	up	

(suite)	Oper_ Status	Total_ Uptime	Mib_index	If Index
	up	16:35:01.66	1.4615.2887056908.2887056909	24
	down	0:00:00.00	2.0.2887056907.2887056909	23

VPN0, VPN1

VPN est une vue qui liste les différents VPN à partir de la table VRF. Chaque entrée correspond à un VPN. Elle s'organise de la manière présentée au tableau XI.

Tableau XI

Vue VPN

Nom de la colonne	Type de données
Name	nvarchar(50)
VRF_index	nvarchar(50)

Les colonnes sont les suivantes :

a. *Name* : nom du VPN;
b. *VRF_index* : index dans la MIB MPLS-VPN [73].

Le tableau XII illustre un exemple de la vue VPN.

Tableau XII

Exemple de vue VPN

Name	Vrf_index
demo1	5.100.101.109.111.49
ETS1	4.69.84.83.49

Remarquons que l'index du VRF contient en fait le nom de celui-ci ; en effet, en faisant abstraction du premier, chaque chiffre correspond au code *American Standard Code for Information Interchange* (ASCII) d'un caractère. Ainsi 100 est le code du caractère « d », 101 pour « e », 109 pour « m » etc. ...

VRF0, VRF1

VRF liste les différents VRF présents dans les LER. Chaque entrée correspond à un VRF. Sa structure est décrite tableau XIII.

Tableau XIII

Table VRF

Nom de la colonne	Type de données
Id	int
Node	nvarchar(50)
Vrf_index	nvarchar(50)
Name	nvarchar(50)
RD	nvarchar(50)
Description	nvarchar(50)
Oper_status	nvarchar(50)
Conf_storagetype	nvarchar(50)
Active_interfaces	nvarchar(50)
Current_routes	nvarchar(50)
Illegal_label_violations	nvarchar(50)

Les colonnes sont les suivantes :

a. *Id* : identifiant du VRF dans la base de données : clé primaire;
b. *Node* : nom du routeur où se trouve ce VRF;
c. *Vrf_index* : index dans la MIB MPLS-VPN [73];
d. *Name* : nom du VRF;
e. *RD* : Route Distinguisher;
f. *Description* : description du VRF;

g. *Oper_status* : Statut opérationnel;
h. *Conf_storagetype* : type de stockage;
i. *Active_interfaces* : nombre d'interfaces actives;
j. *Current_routes* : nombre de routes actuelles;
k. *Illegal_label_violations* : nombre de violations de labels.

Le tableau XIV illustre un exemple de la table VRF.

Tableau XIV

Exemple de table VRF

Id	Node	Vrf_index	Name	RD	Description	(suite)
811	PE1	2.105.113	iq	65000:840	First vrf	
812	PE1	3.99.110.116	cnt	65000:830	Second Vrf	

(suite)	Oper_ status	Conf_ storagetype	Active_ interfaces	Current_ routes	Illegal_ Label_violations
	up	volatile	3	18	0
	up	volatile	5	13	0

VRF_IF0, VRF_IF1

VRF_IF contient les informations pour chaque interface configurée dans le cadre d'un VRF. À chaque entrée correspond une interface configurée pour un VRF donné. Sa structure est présentée tableau XV.

Tableau XV

Table VRF_IF

Nom de la colonne	Type de données
Node	nvarchar(50)
VRF_index	nvarchar(50)
If_index	nvarchar(50)
If_name	nvarchar(50)
Label_edge_type	nvarchar(50)
Vpn_classification	nvarchar(50)
Conf_storagetype	nvarchar(50)
Oper_status	nvarchar(50)

Les colonnes sont les suivantes :

a. *Node* : nom du routeur où se trouve l'interface;
b. *VRF_index* : index du VRF auquel est rattachée l'interface;
c. *If_index* : index de l'interface (cf. vue Router_MPLS);
d. *If_name* : nom de l'interface;
e. *Label_edge_type* : type de « label de bordure » (client ou fournisseur);
f. *Vpn_classification* : classification du VPN;
g. *Conf_storagetype* : type de stockage;
h. *Oper_status* : statut opérationnel.

Le tableau XVI illustre un exemple de la table VRF_IF.

Tableau XVI

Exemple de table VRF_IF

Node	Vrf_index	If_index	If_name	(suite)
PE1	2.105.113	21	Loopback841	
PE1	2.105.113	32	FastEthernet0/0.404	

(suite)	Label_edge_type	Vpn_classification	Conf_storagetype	Oper_status
	providerEdge	enterprise	volatile	active
	providerEdge	enterprise	volatile	active

VRF_ROUTE0, VRF_ROUTE1

VRF_ROUTE contient les informations pour chaque route configurée dans le cadre d'un VRF. À chaque entrée dans la table correspond donc une route. La structure de la table est fournie au tableau XVII.

Tableau XVII

Table VRF_ROUTE

Nom de la colonne	Type de données
Node	nvarchar(50)
VRF_index	nvarchar(50)
Destination	nvarchar(50)
Mask	nvarchar(50)
Next_hop	nvarchar(50)
Tos	nvarchar(50)
Conf_storagetype	nvarchar(50)
If_name	nvarchar(50)
Type	nvarchar(50)
Protocol	nvarchar(50)
Age	nvarchar(50)
Oper_status	nvarchar(50)
Conf_storagetype	nvarchar(50)

a. *Node* : nom du routeur où se trouve la route;
b. *Vrf_index* : index du VRF auquel est rattachée la route;
c. *Destination* : adresse de destination de la route;
d. *Mask* : masque de l'adresse de destination;
e. *Next_hop* : adresse du prochain saut;
f. *Tos* : champ Type Of Service d'IP;
g. *If_name* : nom de l'interface configurée pour cette route, s'il y a lieu;
h. *Type* : type de la route (locale, à distance...);
i. *Protocol* : protocole utilisé pour établir la route;
j. *Age* : âge en secondes de la route;

k. *Oper_status* : statut opérationnel;

l. *Conf_storagetype* : type de stockage.

Le tableau XVIII illustre un exemple de la table VRF_ROUTE.

Tableau XVIII

Exemple de table VRF_ROUTE

Node	Vrf_index	Destination	Mask	Next_hop	ToS	(suite)
2.105.113	PE1	10.99.0.0	255.255.248.0	PE3	0	
2.105.113	PE1	10.99.8.0	255.255.248.0	PE3	0	

(suite)	If_name	Type	Protocol	Age	Oper_status	Conf_storagetype
	0	remote	bgp	625774	active	volatile
	0	remote	bgp	625774	active	volatile

Statistiques des tunnels

Les statistiques des tunnels permettent de savoir si du trafic transite ou non, et à quantifier ce trafic. À chaque entrée de la table correspond une mesure. Pour des raisons de cohérence avec la partie QoS de QMA, les tables relatives aux statistiques sont placées dans une base de données différente. La structure de la table est fournie au tableau XIX.

Tableau XIX

Table Tunnel_*index_du_tunnel*

Nom de la colonne	Type de données
Time	nvarchar(50)
Octets	nvarchar(50)
Packets	nvarchar(50)
BW_octets	nvarchar(50)
BW_packets	nvarchar(50)

Les colonnes sont les suivantes :

a. *Time* : date et heure de la mesure;
b. *Octets* : nombre d'octets ayant transité dans le tunnel au moment de la mesure;
c. *Packets* : nombre de paquets ayant transité dans le tunnel au moment de la mesure;
d. *BW_octets* : bande passante (en octets) estimée entre cette mesure et la précédente;
e. *BW_packets* : bande passante (en paquets) estimée entre cette mesure et la précédente.

Le tableau XX illustre un exemple.

Tableau XX

Table Tunnel_1_0_2887056908_2887056909

Time	Octets	Packets	BW_octets	BW_packets
2006-05-29 12:57:09	72947660	1145110	41,6129	0,6774194
2006-05-29 12:57:39	72948874	1145129	40,46667	0,6333333

Données de gestion

En plus des tables contenant des données qui sont récoltées du réseau par SNMP, QMA utilise des tables nécessaires à son fonctionnement. Il s'agit des cinq tables :

a. *Management* : gère l'accès aux tables;
b. *Monitor* : contient les informations relatives à la surveillance du réseau;
c. *Event* : contient les évènements jugés importants;
d. *Authentication* : contient les données d'authentification des utilisateurs;
e. *Configuration* : contient les données de configuration nécessaires à QMA.

Manager

La table *Manager* a pour rôle de s'assurer du bon emploi des tables. Comme nous l'avons mentionné plus tôt, dans un souci de disponibilité de l'outil, les tables propres à la configuration MPLS sont doublées, pour ne pas empêcher de consultation lors de leur mise à jour. Manager va notamment permettre de connaître quelles tables sont disponibles. En outre, elle facilite la gestion de plusieurs méthodes concurrentes voulant modifier les tables. Sa structure est présentée au tableau XXI.

Tableau XXI

Table Manager

Nom de la colonne	Type de données
Ready	int
Modifiable	int
Being_modified	int
Method_waiting	nvarchar(50)
Max_attempts	int

Ses colonnes sont les suivantes :

a. *Ready* : valeur à 0 ou 1 indiquant quelles tables sont disponibles pour consultation (LSP0, Tunnel0 ou LSP1, Tunnel1 ...);
b. *Modifiable* : valeur à 0 ou 1 indiquant quelles tables sont disponibles pour modification (LSP0, Tunnel0 ou LSP1, Tunnel1 ...). Ready et Modifiable ont donc toujours des valeurs opposées;
c. *Being_modified* : valeur à 0 ou 1 indiquant si des tables sont en train d'être mises à jour;
d. *Method_waiting* : nom de la méthode de mise à jour des tables mise en attente;
e. *Max_attempts* : nombre de fois où une méthode en attente essaie d'accéder à la mise à jour de la table avant d'abandonner.

Les algorithmes de gestion des méthodes concurrentes sont présentés au chapitre 4.

Un exemple de la table Manager est proposé au tableau XXII.

Tableau XXII

Exemple de table Manager

Ready	Modifiable	Being_modified	Method_waiting	Max_attempts
0	1	1	fill_database	100

Dans ce cas ci, les tables doublées disponibles sont les tables finissant par 0 (Segment0, LSP0 etc. ...) ; les tables finissant par 1 sont présentement en cours de mise à jour, et une méthode *fill_database* attend de pouvoir mettre à jour les tables, ce qu'elle essaiera de faire 100 fois.

Monitor

La table Monitor contient les informations de configuration de la surveillance du réseau. Sa structure est présentée au tableau XXII.

Tableau XXIII

Table Monitor

Nom de la colonne	Type de données
Admin_status	bit
DateTime_check	datetime
Refresh_stats	bit
Refresh_freq	int
Listen_traps	bit

Ses colonnes sont les suivantes :

a. *Admin_status* : valeur booléenne indiquant s'il faut ou non effectuer de la surveillance;
b. *DateTime_check* : date et time du dernier pointage de l'applet : permet de savoir si un client est actuellement connecté;
c. *Refresh_stats* : valeur booléenne indiquant s'il faut ou non effectuer collecter des statistiques;
d. *Refresh_freq* : fréquence (en millisecondes) de collecte des statistiques;
e. *Listen_traps* : valeur booléenne indiquant s'il faut ou non écouter les traps.

Un exemple de la table Manager est proposé au tableau XXIV.

Tableau XXIV

Exemple de table Monitor

Admin_status	DateTime_check	Refresh_stats	Refresh_freq	Listen_traps
True	2006-05-30 15:23:31	True	30000	False

Event

La table Event permet de lister tous les évènements jugés importants, par exemple un tunnel MPLS TE qui tombe. Sa structure est présentée au tableau XXV.

Tableau XXV

Table Event

Nom de la colonne	Type de données
Id	int
DateTime	nvarchar(50)
Change	nvarchar(50)
Node	nvarchar(50)

Ses colonnes sont les suivantes :

a. *Id* : identifiant de l'évènement dans la base de données : clé primaire;
b. *DateTime* : date et heure de l'évènement;
c. *Change* : description de l'évènement;
d. *Node* : nom du routeur affecté par l'évènement.

Un exemple de la table *Event* est proposé tableau XXVI.

Tableau XXVI

Exemple de table Event

Id	DateTime	Change	Node
33	2006-05-08 17:17:00	Tunnel1 down	PE2
34	2006-05-08 19:09:53	new Tunnel1 up	PE2

Authentication

La table *Authentication* sert à vérifier les accès. Sa structure est présentée au tableau XXVII.

Tableau XXVII

Table Authentication

Nom de la colonne	Type de données
Login	nvarchar(50)
Password	nvarchar(50)
Subset	nvarchar(50)
Administrator	bit
LastTime	DateTime

Ses colonnes sont les suivantes :

a. *Login* : nom d'utilisateur;
b. *Password* : mot de passe haché avec un sel (SHA256);
c. *Subset* : ensemble auquel le compte appartient (cf. table Configuration);
d. *Administrator* : valeur indiquant si l'utilisateur est administrateur ou non;
e. *LastTime* : date et heure de la dernière authentification. Permet de savoir si l'utilisateur est connecté.

Un exemple de la table Authentication est proposé tableau XXVIII.

Tableau XXVIII

Exemple de table Authentication

Login	Password	(suite)
admin	oeIwbxzT0irkJ9BBULdGV5wWlsRUa	
user	kiiKdjdjsksdsfds8K==dlksdejrwrffgfgsaerfekdsjeb	

(suite)	Subset	Administrator	LastTime
	1	True	2006-06-07 10:19:15
	1	False	2006-05-07 10:19:15

Configuration

La table Configuration, étroitement liée à la table *Authentication*, fournit pour un ensemble les différents paramètres nécessaires à QMA, comme les accès SNMP aux routeurs et à la base de données. De fait, les tables Configuration et *Authentication* sont définies dans un emplacement fixe, mais l'emplacement des autres tables est laissé à la discrétion de l'administrateur. Sa structure est présentée au tableau XXIX.

Tableau XXIX

Table Configuration

Nom de la colonne	Type de données
Id	nvarchar(10)
Database_Server	nvarchar(50)
Database_Name	nvarchar(50)
Database_Stats_Name	nvarchar(50)
Database_Login	nvarchar(50)
Database_Password	nvarchar(50)
Snmp_Login	nvarchar(50)
Snmp_Password	nvarchar(50)

Ses colonnes sont les suivantes :

a. *Id* : identifiant de l'ensemble;
b. *Database_Server* : nom du serveur de la base de données;
c. *Database_Name* : nom de la base de données;
d. *Database_Stats_Name* : nom de la base de données de statistiques;
e. *Database_Login* : nom d'utilisateur de la base de données;
f. *Database_Password* : mot de passe de la base de données;
g. *Snmp_Login* : nom d'utilisateur SNMP dans les routeurs;
h. *Snmp_Password* : mot de passe SNMP dans les routeurs.

Un exemple de la table Configuration est fourni au tableau XXX.

Tableau XXX

Exemple de table Configuration

Id	Database_ Server	Database_ Name	Database_ Stats_Name	(suite)
1	lagrit21\ SQLEXPRESS	test_outil_ mpls_try	Mgmt ToolStatsDB	

(suite)	Database_ Login	Database_ Password	Snmp_ Login	Snmp_ Password
	sa	lagrit	etudiant	mplspower2

Fichiers de logs

En plus des tables, QMA stocke de l'information dans des fichiers textes, plus légers et facilement accessibles : deux fichiers sont ainsi utilisés :

a. *logs.txt* : contient les évènements liés au fonctionnement de QMA, permet de savoir quelles opérations sont effectuées et à quel moment;
b. *traps.txt* : contient les *traps* reçues ; utile si l'information consignée dans la table *Event* ne suffit pas, et que des détails techniques sont nécessaires.

BIBLIOGRAPHIE

[1] ISO, *International Standard ISO/IEC 7498-4, Information processing systems -- Open Systems Interconnection ? Basic Reference Model -- Part 4: Management framework.* 1989.

[2] Postel, J., *RFC792 - Internet Control Message Protocol.* 1981.

[3] Hares, S., *RFC1574 - Essential Tools for the OSI Internet.* 1994.

[4] J. Case, M.F., M. Schoffstall, J. Davin, *RFC1157 - A Simple Network Management Protocol (SNMP).* 1990.

[5] J. Case, K.M., M. Rose, S. Waldbusser, *RFC1905 - Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2).* 1996.

[6] U. Blumenthal, J.C., D. Harrington, D. Levi, K. McCloghrie, P. Meyer, R. Presuhn, B. Stewart, B. Wijnen, *RFC3411-3415 Simple Network Management Protocol (SNMPv3).* 2002.

[7] Ash, J., et al., *AT&T's MPLS OAM architecture, experience, and evolution.* Communications Magazine, IEEE, 2004. **42**(10): p. 100-111.

[8] Rosen, E., A. Viswanathan, and R. Callon, *RFC3031 - Multiprotocol Label Switching Architecture.* 2001.

[9] Nadeau, T., M. Morrow, and G. Swallow, *RFC4377 - OAM Requirements for MPLS Networks.* 2006.

[10] Guichard, J., I. Pepelnjak, and J. Apcar, *MPLS and VPN architectures.* 2001, Indianapolis, IN: Cisco Press. 2 v.

[11] MPLS, W.G. *Multi Protocol Label Switching Charter.* 2005 [En ligne] http://www.ietf.org/html.charters/mpls-charter.html.

[12] Cisco, *Internetworking Basics*. 4 ed. 2003. 1110.

[13] Rosen, E., et al., *RFC3032 - MPLS Label Stack Encoding*. 2001.

[14] Stallings, W., *MPLS*, in *The Internet Protocol Journal (IPJ)*. 2001.

[15] Osborne, E. and A. Simha, *Traffic Engineering with MPLS*. 2003, Indianapolis, IN: Cisco Press.

[16] Andersson, L., et al., *RFC3036 - LDP Specification*. 2001.

[17] Bidau, A. *MPLS : introduction, fonctionnement, applications*. in *Renater*.

[18] Cisco, *Building MPLS-Based VPNs for Service Provider IP Networks*. 2001.

[19] Farinacci, D., et al., *RFC2784 - Generic Routing Encapsulation (GRE)*. 2000.

[20] Kent, S. and R. Atkinson, *RFC2401 - Security Architecture for the Internet Protocol*. 1998.

[21] Rosen, E. and Y. Rekther, *RFC4364 - BGP/MPLS IP VPNs*. 2006.

[22] Arcome, *Etude technique, économique et réglementaire de l'évolution vers les réseaux de nouvelle génération (NGN, Next Generation Networks)*. 2002.

[23] Mannie, E., *RFC3945 - Generalized Multi-Protocol Label Switching (GMPLS) Architecture*. 2004.

[24] Gao Xianrui, Z.Y., *The NGN age has arrived.* Huawei technologies, 2005(20).

[25] Allan, D. and T. Nadeau, *RFC4378 - A Framework for Multi-Protocol Label Switching (MPLS) Operations and Management (OAM)*. 2006.

[26] Kompella, K. and G. Swallow, *RFC4379 - Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures*. 2006.

[27] Nadeau, T., C. Srinivasan, and A. Farrel, *RFC4221 - Multiprotocol Label Switching (MPLS) Management Overview*. 2005.

[28] Srinivasan, C., A. Viswanathan, and T. Nadeau, *RFC3812 - Multiprotocol Label Switching (MPLS) Traffic Engineering (TE) Management Information Base (MIB)*. 2004.

[29] Srinivasan, C., A. Viswanathan, and T. Nadeau, *RFC3813 - Multiprotocol Label Switching (MPLS) Label Switching Router (LSR) Management Information Base (MIB)*. 2004.

[30] Nadeau, T., C. Srinivasan, and A. Viswanathan, *RFC3814 - Multiprotocol Label Switching (MPLS) Forwarding Equivalence Class To Next Hop Label Forwarding Entry (FEC-To-NHLFE) Management Information Base (MIB)*. 2004.

[31] Luciani, J., H. Sjostrand, and J. Cucchiara, *RFC3815 - Definitions of Managed Objects for the Multiprotocol Label Switching (MPLS), Label Distribution Protocol (LDP)*. 2004.

[32] Cavendish, D., H. Ohta, and H. Rakotoranto, *Operation, administration, and maintenance in MPLS networks.* Communications Magazine, IEEE, 2004. **42**(10): p. 91-99.

[33] Fang, L., et al., *LDP failure detection and recovery.* Communications Magazine, IEEE, 2004. **42**(10): p. 117-123.

[34] Akyildiz, I.F., et al., *A new traffic engineering manager for DiffServ/MPLS networks: design and implementation on an IP QoS Testbed.* Computer Communications, 2003. **26**(4): p. 388-403.

[35] Scoglio, C., et al., *TEAM: A traffic engineering automated manager for DiffServ-based MPLS networks.* Communications Magazine, IEEE, 2004. **42**(10): p. 134-145.

[36] Anjali, T., C. Scoglio, and G. Uhl, *A new scheme for traffic estimation and resource allocation for bandwidth brokers.* Computer Networks, 2003. **41**(6): p. 761-777.

[37] Aukia, P., et al., *RATES: A Server for MPLS Traffic Engineering.* IEEE Network, 2000. **vol. 14**(Mar./Apr. 2000): p. pp. 34–41.

[38] Elwalid, A., et al. *MATE: MPLS adaptive traffic engineering.* in *INFOCOM.* 2001.

[39] Ipswitch. *What'up.* 2006 [En ligne] http://www.ipswitch.com/products/whatsup/.

[40] Hewlett-Packard. *HP Open View.* 2006 [En ligne] http://www.managementsoftware.hp.com/.

[41] Hewlett-Packard, *HP OpenView Network Services Management Solution for MPLS networks - Technical Blueprint.* 2005.

[42] Cisco, *Cisco IP Solution Center.* 2006.

[43] Cisco. *Cisco IOS Technologies*. 2006 [En ligne] http://www.cisco.com/en/US/products/ps6537/products_ios_sub_category_home.html.

[44] Breton, M.-A., *Développement d'un système de surveillance des mécanismes de qualité de service dans le contexte des réseaux de prochaine génération*. 2006, École de Technologie Supérieure: Montréal.

[45] Postel, J. and J. Reynolds, *RFC854 - Telnet Protocol Specification*. 1983.

[46] Ylonen, T. and E. C. Lonvick, *RFC4251 - The Secure Shell (SSH) Protocol Architecture*. 2006.

[47] Postel, J., *RFC768 - User Datagram Protocol*. 1980.

[48] Postel, J., *RFC793 - Transmission Control Protocol*. 1981.

[49] Fielding, R., et al., *RFC2616 - Hypertext Transfer Protocol -- HTTP/1.1*. 1999.

[50] Rescorla, E., *RFC2818 - HTTP Over TLS*. 2000.

[51] Sollins, K., *RFC1350 - The TFTP protocol (revision 2)*. 1992.

[52] Ohta, H., *RFC3429 - Assignment of the 'OAM Alert Label' for Multiprotocol Label Switching Architecture (MPLS) Operation and Maintenance (OAM) Functions*. 2002.

[53] Katz, D., *RFC2113 - IP Router Alert Option*. 1997.

[54] Katz, D. and D. Ward, *Bidirectional Forwarding Detection draft-ietf-bfd-base-04.txt*. 2005.

[55] Aggarwal, R., et al., *BFD For MPLS LSPs*. 2005.

[56] Swallow, G., K. Kompella, and D. Tappan, *Label Switching Router Self-Test draft-ietf-mpls-lsr-self-test-06.txt*. 2005.

[57] Chatfield, C., *The analysis of time series : an introduction*. 6th ed. Texts in statistical science. 2004, Boca Raton, Flor.: Chapman & Hall/CRC. xiii, 333.

[58] Kolarov, A., A. Atai, and J. Hui. *Application of Kalman filter in high-speed networks*. in *Global Telecommunications Conference, 1994. GLOBECOM '94. 'Communications: The Global Bridge'., IEEE*. 1994.

[59] Haro, F., M. Chhaparia, and L. Cottrell, *Detecting loss of performance in Dynamic Bottleneck Capacity (DBCap) measurements using the Holt-Winters Algorithm*. 2004.

[60] NIST, *Engineering Statistics Handbook*, ed. NIST. 2005.

[61] CCM, *Réseaux - Architecture client/serveur à 3 niveaux*. 2006.

[62] Sun. *Java Platform, Entreprise Edition (Java EE)*. 2006 [En ligne] http://java.sun.com/javaee/index.jsp.

[63] Microsoft. *Microsoft .Net*. 2006 [En ligne] http://www.microsoft.com/net/default.mspx.

[64] Simonnet, J., *Rapport de stage de fin d'études - « Outil de surveillance dans un réseau MPLS »*. 2005, LAGRIT - École de Technologie Supérieure.

[65] Oracle. *Oracle Corporation*. 2006 [En ligne] http://www.oracle.com/.

[66] MySQL. *MySQL AB*. 2006 [En ligne] http://www.mysql.com/.

[67] PostGreSQL. *PostgreSQL*. 2006 [En ligne] http://www.postgresql.org/.

[68] Microsoft. *Microsoft SQL Server*. 2006 [En ligne] http://www.microsoft.com/sql/.

[69] Microsoft. *Visual Studio*. 2006 [En ligne] http://msdn.microsoft.com/vstudio/.

[70] OpenSource. *ZedGraph*. 2006 [En ligne] http://zedgraph.org.

[71] Obviex. *How To: Hash Data with Salt (C#/VB.NET)*. 2006 [En ligne] http://www.obviex.com/samples/hash.aspx.

[72] *Net-SNMP*. 2006 [En ligne] http://net-snmp.sourceforge.net/.

[73] Nadeau, T. and H. Van der Linde, *RFC4382 - MPLS/BGP Layer 3 Virtual Private Network Management Information Base*. 2006.

[74] Nadeau, T., *MPLS Network Management*. 2003: Morgan Kaufmann Publishers.

[75] Nadeau, T.D. and Z. Ali, *Bidirectional Forwarding Detection Management Information Base - draft-ietf-bfd-mib-02.txt* 2005.

[76] *Bugzilla*. 2006 [En ligne] http://www.bugzilla.org/.

[77] Microsoft. *.Net Developer Center*. 2006 [En ligne] http://msdn.microsoft.com/netframework/.

[78] McCloghrie, K. and M. Rose, *RFC1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. 1991.