UNIVERSITÉ DU QUÉBEC À MONTRÉAL

IMPLÉMENTATION D'UN PLAN DE CONTRÔLE POUR LES RÉSEAUX MULTICOUCHES IP/OPTIQUES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE ÈS SCIENCES (TECHNOLOGIES DE L'INFORMATION)

PAR
PATRICE KUEKEM TCHOUA

JUIN 2016

REMERCIEMENTS

Ce mémoire est dédié au Seigneur Dieu Tout Puissant, le Dieu d'Abraham, Dieu d'Isaac, Dieu de Jacob.

Tous mes remerciements aux professeurs de l'UQAM qui, par leurs enseignements, encadrements et conseils, ont permis l'accomplissement de ce travail de recherche.

Toute ma gratitude au Canada, et en particulier au Québec d'où l'opportunité et les moyens m'ont été offerts afin de poursuivre mes études.

Une très grande pensée à toutes celles et ceux qui, de prêt ou de loin m'ont assisté pendant ce parcours universitaire.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABREVIATIONS, SIGLES ET ACRONYMES

Les traductions portant la mention « ® » proviennent de la banque de données terminologiques et linguistiques du gouvernement du Canada (**TERMIUM Plus®**).

API	Application Programming Interface (Interface de programmation d'application ®)
AST	Application Specific Topologies (Topologies spécifiques à l'application)
ASTB	Application Specific Topology Builder (Constructeur de topologie spécifique à l'application)
ATM	Asynchronous Transfer Mode (Mode de transfert asynchrone ®)
BHP	Burst Header Packet (Entête de paquet à rafale)
CLI	Command Line Interface (Interface de ligne de commande)
CS	Client System (Système client)
CSA	Client System Agent (Agent de système client)
DB	Data Burst (Rafale de données)
DHCP	Dynamic Host Configuration Protocol (Protocole DHCP ®)
DRAGON	Dynamic Resource Allocation in GMPLS Optical Networks (Allocation dynamique des ressources dans les réseaux optiques GMPLS)
DWDM	Dense Wavelength-Division Multiplexing (Multiplexage par répartition en longueur d'onde dense ®)
ERO	Explicit Route Object (Objet de route explicite)
GMPLS	Generalized Multi-Protocol Label Switching (Commutation multi protocole par étiquette généralisée)
GNU	GNU's Not UNIX (GNU n'est pas UNIX)
Go	Giga byte (Giga octet)
GRE	Generic Routing Encapsulation (Encapsulation à routage générique)

IP	Internet Protocol (Protocole Internet)
IPv6	Internet Protocole version 6 (Protocole Internet version 6)
LSA	Link State Announcement (Annonce à état de lien)
LSP	Label Switched Path (Chemin commuté par étiquette)
LSR	Label Switching Router (Routeur à commutation d'étiquette)
MHz	Megahertz (Méga hertz ®)
MIB	Management Information Base (Base d'information de gestion ®)
Mo	Megabyte (Méga octet)
MPLS	Multi-Protocol Label Switching (Commutation multi protocole par étiquette)
NARB	Network Aware Resource Broker (Agent de ressource réseau averti)
NAT	Network Address Translation (Traduction d'adresses de réseau ®)
OBS	Optical Burst Switching (Commutation optique par rafales ®)
ONS	Optical Network Switch (Commutateur réseau optique)
OS	Operating System (Système d'exploitation ®)
OSPF	Open Shortest Path First (Open shortest path first ®)
PC	Personal Computer (Ordinateur personnel)
PCE	Path Computation Element (Élément de calcul de chemin)
QSON	Automatically Switched Optical Network (Réseau optique à commutation automatique)
RAM	Random Access Memory (Mémoire vive ®)
RCE	Resource Computation Element (Élément de calcul de ressource)
RFC	Requests For Comments (Appel de commentaires ®)
RSVP	Resource Reservation Protocol (Protocole de réservation de ressource)
RWA	Routing and Wavelength Assignment (Routage et affection de longueur d'onde)
SDN	Software-Defined Network (Réseau SDN ®)
SNMP	Simple Network Management Protocol (Protocole de gestion de réseau simple ®)
SONET	Synchronous Optical Network (Réseau optique synchrone ®)
SSH	Secure Shell Protocol (Protocole SSH ®)
TDM	Time Division Multiplexing (Multiplexage temporel ®)

TE	Traffic Engineering (Ingénierie de la circulation ®, Technique de la circulation ®)
TELNET	Telecommunication Network (Telnet ®)
TL1	Transaction Language N°1 (Langue de transaction N°1)
TTL	Time to Live (Durée de vie ®)
UCP	Unified Control Plane (Plan de contrôle unifié)
UHD	Ultra High Definition (Très haute définition)
UNI	User-Network Interface (Interface usager-réseau ®)
VLAN	Virtual Local Area Network (Réseau local virtuel ®)
VLSR	Virtual Label Switching Router (Routeur virtuel à commutation par étiquette)
VM	Virtual Machine (Machine virtuelle ®)
VPN	Virtual Private Network (Réseau privé virtuel ®)
WDM	Wavelength-Division Multiplexing (Multiplexage par répartition en longueur d'onde ®)
WSON	Wavelength Switched Optical Network (Réseau optique à commutation de longueur d'onde)

RÉSUMÉ

Un plan de contrôle pour les réseaux IP/Optiques dans une architecture SDN est très important, parce qu'il permet de centraliser de façon logique l'intelligence et l'état du réseau à partir des applications sur des super-ordinateurs appelés contrôleurs ou plans de contrôle. Les protocoles les plus utilisés pour implémenter un plan de contrôle respectant l'architecture SDN pour les réseaux IP/Optiques sont OpenFlow et GMPLS.

Mais, en dépit d'énormes progrès et d'expérimentation sur OpenFlow, il demeure efficace comme plan de contrôle sur les réseaux IP, et reste encore à un stade initial sur les réseaux optiques. Aussi, malgré plus de deux décennies de développement et des travaux de standardisation sur GMPLS, il demeure seulement efficace comme plan de contrôle sur les réseaux optiques.

Le défi que nous avons relevé a été de concevoir et d'implémenter un plan de contrôle basé à la fois sur OpenFlow et GMPLS, et d'introduire une technique d'interfonctionnement entre les protocoles OpenFlow et GMPLS. En plus, nous avons été capables par notre plan de contrôle, de faire fonctionner les équipements compatibles GMPLS ou non dans un réseau GMPLS tout en respectant l'architecture SDN.

La faisabilité globale de notre solution est démontrée, et sa performance est évaluée quantitativement et comparée à celle d'une solution basée sur OpenFlow étendue, et d'une autre basée sur GMPLS. Les résultats de cette comparaison montrent que la solution OpenFlow étendue serait plus rapide que les autres. Mais, les caractéristiques de comparaison des plans de contrôle montrent qu'un plan de contrôle basé à la fois sur OpenFlow et GMPLS pour les réseaux IP/Optiques serait mature et facilement évolutif contrairement aux autres.

MOTS-CLÉS : OpenFlow, GMPLS, plan de contrôle, plan de données, réseau multicouche, réseau IP/Optique.

INTRODUCTION

L'Internet du futur et les applications Internet émergentes sont caractérisés par le transfert global du trafic des paquets commutés, conduit par les applications basées sur des réseaux à haute performance tels que la demande vidéo sur streaming et l'infonuagique (Azodolmolky *et al.*, 2011 p.1 ; Channegowda *et al.*, 2013 p.2). Afin que ce transfert de paquets soit effectué de façon efficiente, les opérateurs et fournisseurs de services et produits réseaux doivent gérer des réseaux multi domaines (domaines IP et optiques). Cette gestion pose d'énormes problèmes. En effet, les domaines IP et optiques sont exploités séparément par deux équipes distinctes (une pour le domaine IP et une pour le domaine optique), sans interaction dynamique ; ce qui entraîne des coûts d'exploitation élevés, une faible efficacité du réseau, et une longue latence de traitement pour acheminer des données (Lei, Tsuritani et Morita, 2012a p.1). Aussi, on assiste à une augmentation très importante des réseaux hautement sécurisés. De ce fait, les opérateurs et fournisseurs de services réseaux ont impérativement besoin d'une nouvelle solution réseau pour s'attaquer de façon efficiente aux demandes croissantes de l'évolution du paysage réseau. Avec les interfaces programmables des équipements réseaux d'aujourd'hui, cela est difficile à accomplir (Sezer *et al.*, 2013 p.1); d'où la nécessité de repenser la conception et le fonctionnement des équipements réseaux actuels. Pour cela, les recherches ont conduit à une nouvelle architecture nommée SDN (*Software-Defined Network*). Cette dernière a la particularité de séparer le plan de contrôle de celui de données, et de centraliser de façon logique l'intelligence et l'état du réseau à partir des applications. Les protocoles couramment utilisés dans la mise en œuvre des plans de contrôle respectant l'architecture SDN sont OpenFlow et GMPLS.

Mais en dépit d'énormes progrès et d'expérimentation sur OpenFlow, il demeure efficace comme plan de contrôle sur la portion IP du réseau (Das *et al.*, 2010 ; Gudla *et al.*, 2010 ; Simeonidou, Nejabati et Azodolmolky, 2011), et reste encore à un stade initial pour

les réseaux optiques. Pour cela, le protocole MPLS devenu aujourd'hui GMPLS a vu le jour pour assurer la gestion efficace de la couche optique. Aussi, malgré plus de deux décennies de développement et des travaux de standardisation sur GMPLS, il demeure seulement efficace comme technique de plan de contrôle pour les réseaux optiques; mais, comme plan de contrôle dans les réseaux multicouches IP/Optiques, il est trop complexe à mettre en œuvre en raison de sa nature distribuée, du nombre de protocoles et des interactions entre les différentes couches (Kumaki, 2008 ; Shiomoto, 2008). Pour assurer efficacement la gestion des réseaux IP/Optiques à partir d'un plan de contrôle, une solution logique à l'heure actuelle serait d'introduire une technique d'interfonctionnement entre les protocoles OpenFlow et GMPLS, capable d'utiliser GMPLS pour contrôler la couche optique (nœud de commutation optique des réseaux de transport optique) et OpenFlow pour coordonner et faire interagir les couches IP et optiques dynamiquement.

Au meilleur de nos connaissances, trois auteurs ont déjà implémenté des plans de contrôle basés à la fois sur OpenFlow et GMPLS; il s'agit de: Azodolmolky *et al.* (2011) ; Channegowda *et al.* (2013) ; Lei, Tsuritani et Morita (2012b, 2012a). Par contre, leurs plans de contrôle ne gèrent pas les équipements non-compatibles GMPLS. Ce qui soulève un problème crucial, qui est le fait qu'un grand nombre d'équipements réseaux en fonctionnement de nos jours ne sont pas compatibles GMPLS.

La première différence fondamentale entre notre plan de contrôle et ceux des auteurs suscités est que, le nôtre est capable de gérer dans une architecture réseau les équipements non-compatibles GMPLS. La deuxième différence fondamentale est que, les auteurs suscités sont partis des protocoles OpenFlow standand et GMPLS standard pour implémenter leurs plans de contrôle, alors que nous sommes partis de OpenFlow standard et de GMPLS-Dragon qui est une version étendu de GMPLS. Notre contribution est donc de bâtir un plan de contrôle basé sur OpenFlow et GMPLS, capable de gérer les équipements réseaux compatibles GMPLS ou non, et pouvant assurer la restauration d'un lien réseau après sa rupture.

Pour bâtir notre plan de contrôle, nous avons fait des extensions des protocoles OpenFlow standard et GMPLS-Dragon. Nous avons ensuite conçu et mis en œuvre une architecture réseau basée sur SDN qui convient à notre solution. Nous avons aussi implémenté, testé et évalué notre plan de contrôle. Ce dernier a également été comparé à un plan de contrôle basé

sur OpenFlow étendu et à un autre basé sur GMPLS. Ce choix de comparaison est expliqué plus loin dans le document. Nous avons enfin décrit le comportement de notre plan de contrôle en cas de rupture de lien dans l'architecture réseau qui a été proposée. Les résultats d'évaluation et de comparaison montrent que notre plan de contrôle basé à la fois sur OpenFlow et GMPLS-Dragon serait mature et facilement évolutif, bien qu'il serait aussi légèrement moins rapide que ceux basés uniquement sur OpenFlow ou uniquement sur GMPLS et qui ne gèrent pas de module de restauration d'un lien réseau après sa rupture.

Notre mémoire est organisé comme suit : le premier chapitre est consacré à la problématique, à l'objectif et à la méthodologie de recherche. Le deuxième chapitre est dédié à la revue de littérature. Les troisième et quatrième chapitres sont réservés à la connaissance des protocoles GMPLS via le projet DRAGON (*Dynamic Resource Allocation in GMPLS Optical Networks*) et OpenFlow via le tutorial de déploiement de OpenFlow de la *OF Foundation*. Les cinquième et sixième chapitres concernent la conception, l'implémentation, le test, et l'évaluation du plan de contrôle OpenFlow-GMPLS.

Après cette introduction, nous passons maintenant à la problématique, à l'objectif et à la méthode de recherche.

CHAPITRE I

PROBLÉMATIQUE, OBJECTIF ET MÉTHODOLOGIE DE RECHERCHE

Dans ce chapitre, nous allons poser le problème de recherche et faire ressortir les concepts clés. L'objectif de recherche et la solution proposée seront ensuite présentés. La méthodologie de recherche viendra clore ce chapitre.

1.1 Contexte et Problématique

Dans cette section, nous présentons le contexte de notre étude et la problématique qui a motivé cette recherche.

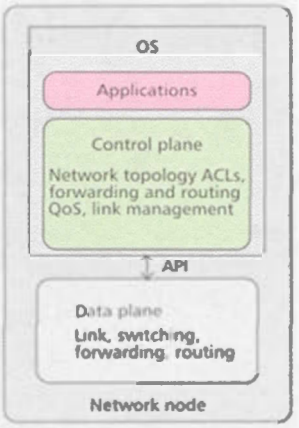
1.1.1 Contexte

L'Internet d'aujourd'hui est caractérisé par le transfert des données extrêmement élevé, et nécessite des applications basées sur des réseaux à haute performance. Ces applications multimédias sont de plus en plus exigeantes en termes de débit de communication et de bande passante. Le fait que les plans de contrôle et de données soient associés dans une même couche de l'architecture réseau traditionnelle, et que les réseaux multicouches IP/Optiques d'aujourd'hui ne possèdent pas de plan de contrôle, viennent compliquer la situation existante. Un plan de contrôle basé sur l'architecture SDN est plus que nécessaire.

L'origine du problème remonte à la technologie client-serveur sur laquelle l'architecture réseau actuelle est basée.

Sezer et al., (2013) présentent clairement les différences entre l'architecture des réseaux traditionnels d'aujourd'hui, et l'architecture SDN. En effet, dans l'architecture des réseaux traditionnels de nos jours, chaque nœud réseau comprend un plan de données et un OS

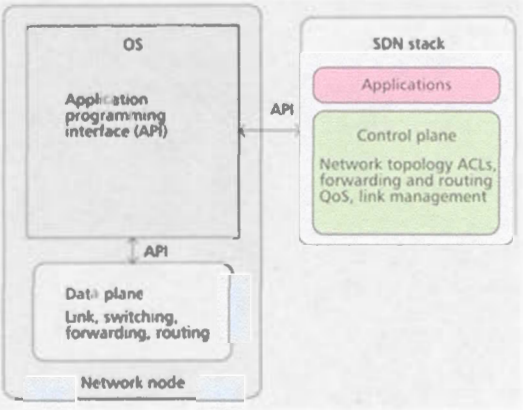
(*Operating System*) incluant un plan de contrôle et une couche application. La figure ci-dessous illustre cette représentation.



Source : (Sezer et al., 2013 38)

Figure 1.1 Architecture de réseau traditionnel

Dans l'architecture SDN, le plan de contrôle et la couche application sont extraits et constituent une couche appelée « pile SDN » qui se situe hors de l'OS. La figure ci-dessous illustre ce concept SDN.



Source : (Sezer et al., 2013 38)

Figure 1.2 Architecture de réseau SDN

Or, l'explosion des appareils mobiles, les types de contenu échangés par les internautes, la virtualisation des serveurs et l'avènement des services en nuage sont des exemples des tendances de l'industrie de la réseautique qui forcent à réexaminer les architectures réseaux traditionnelles d'aujourd'hui. De leurs structures hiérarchiques, les nombreux réseaux conventionnels sont construits avec des niveaux de commutateurs Ethernet disposés dans une structure arborescente. Cette conception était logique au moment où la technologie informatique client-serveur était dominante ; mais une telle architecture statique est inadaptée aux besoins de stockages dynamiques des centres de données, des campus et environnements des prestataires de services réseaux d'aujourd'hui (Foundation, 2012).

Cette non-compatibilité a un impact très large, allant des entreprises fournissant des services réseaux jusqu'aux consommateurs que nous sommes. En effet, le fait que les plans de contrôle et de données soient associés dans une même couche réseau, oblige les praticiens à accéder individuellement sur chaque équipement réseau pour sa configuration ou sa maintenance. Cette façon de faire impacte négativement les performances du réseau, car les temps de configuration et de maintenance des équipements réseaux sont plus longs, contrairement à une gestion des équipements réseaux dont l'intelligence et l'état sont centralisés de façon logique à partir des applications sur des contrôleurs comme dans l'architecture SDN. Le fait d'accéder individuellement sur chaque équipement réseau pour une quelconque intervention pousse aussi les praticiens réseaux à travailler plus, pour un rendement pas toujours à la hauteur de l'effort fourni. Cela pose aussi un problème d'efficacité des entreprises fournisseurs des services réseaux, quand on sait qu'elles ont plusieurs milliers d'équipements réseaux à gérer. Cette inefficacité impacte négativement la qualité de service. Moins le réseau est performant, moins la qualité de service réseau est bonne.

Les impacts et leurs conséquences sont mis en évidence dans le schéma de la page suivante :

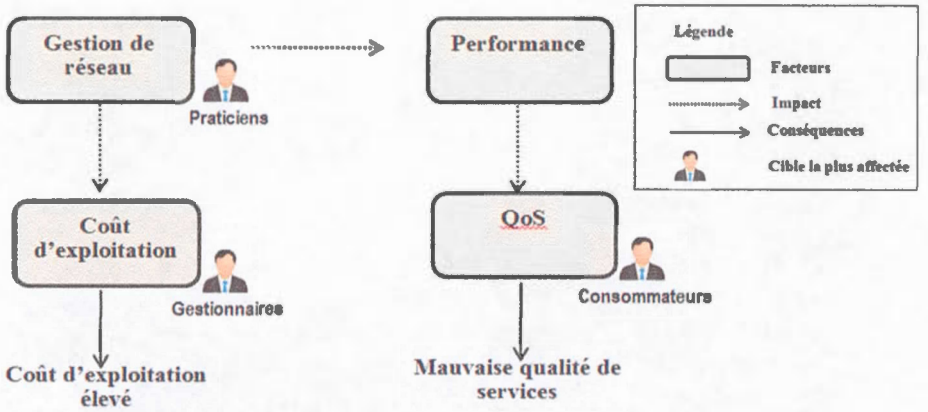


Figure 1.3 Impacts et conséquences de l'architecture réseau traditionnel

La figure 1.1 montre que la gestion des réseaux traditionnels d'aujourd'hui a une influence négative sur une grande partie de la population (praticiens, gestionnaires et consommateurs des services réseaux). Les conséquences qui en découlent sont, la mauvaise qualité de service pour les consommateurs, et les coûts d'exploitation élevés pour les gestionnaires.
Il est alors extrêmement important d'étudier ce problème afin de relever ces défis énormes.

1.1.2 Question centrale

Pour suivre l'évolution des réseaux et maintenir ou améliorer la qualité de service dans les réseaux IP/Optiques, il faudrait séparer le plan de contrôle de celui de données, centraliser de façon logique l'intelligence et l'état du réseau à partir des applications sur des super-ordinateurs appelés contrôleurs ou plans de contrôle, et permettre aux équipements réseaux compatibles GMPLS ou non, de fonctionner dans une architecture réseau GMPLS. Ce grand défi nous interpelle chercheurs et praticiens, et nous amène à nous poser la question suivante : comment assurer la gestion intelligente, dynamique et centralisée des réseaux multicouches IP/Optiques, en permettant aux équipements non-compatibles GMPLS d'être fonctionnels dans un réseau GMPLS tout en utilisant l'architecture SDN ?
Cette question est essentielle pour la réseautique, car elle permet d'obéir à la tendance générale qui est la migration des réseaux actuels vers l'architecture SDN.

1.1.3 Questions sectorielles

1.1.3.1 Pourquoi un plan de contrôle ?

Un plan de contrôle pour le management de l'infrastructure réseau (routeurs, commutateurs, etc.) permet de rendre la gestion très efficace, capable de supporter la nature dynamique des fonctions futures du réseau et des applications intelligentes, tout en réduisant les coûts grâce aux équipements, aux logiciels et à une gestion simplifiée (Sezer *et al.*, 2013).

1.1.3.2 Pourquoi sur les équipements compatibles GMPLS ou non ?

Les plus grands constructeurs d'équipements réseaux (Cisco, HP, 3 Com, Jupiter, etc.) produisent de nos jours des équipements compatibles GMPLS, contrairement aux équipements produits antérieurement. De ce fait, un grand nombre d'équipements non-compatibles GMPLS sont toujours en utilisation. C'est pour faciliter la transition matérielle vers les nouveaux équipements, et laisser le temps aux utilisateurs finaux d'abandonner par eux-mêmes leurs anciens matériels au lieu d'être forcés à le faire, que nous proposons un plan de contrôle permettant également aux équipements non-compatibles GMPLS d'être fonctionnels.

Après avoir présenté le contexte et la problématique de recherche, nous nous focalisons à présent sur l'objectif et la solution proposée.

1.2 Objectif et solution proposée

Dans cette section, nous présentons le but de notre recherche et la solution que nous offrons.

1.2.1 Objectifs et pertinence du sujet

Notre recherche a pour objectif de concevoir et d'implémenter un plan de contrôle pour assurer la gestion dynamique, intelligente et centralisée des réseaux multicouches IP/Optiques respectant l'architecture SDN, tout en permettant aux équipements non-compatibles GMPLS d'être fonctionnels dans des réseaux GMPLS. Aussi, notre plan de contrôle devra être en mesure d'assurer la restauration d'un lien réseau après sa rupture. Ces objectifs ont été atteints via la mise en œuvre d'un plan de contrôle basé à la fois sur

OpenFlow et sur GMPLS-Dragon, et possédant un module de restauration après rupture de lien.

Notre travail de recherche est pertinent, car la gestion des réseaux traditionnels d'aujourd'hui a une influence négative sur ses performances. Ce qui impacte négativement la qualité des services réseaux offerts aux consommateurs qui constituent une très grande partie de la population. En plus, nous avons pensé au grand nombre d'équipements non-compatibles GMPLS toujours en fonctionnement. Cela permet d'assurer une transition aisée entre les équipements d'aujourd'hui et ceux de demain; car une gestion efficace du changement, même matérielle passe par une bonne gestion de la transition entre les anciens et les nouveaux équipements.

1.2.2 Solution proposée

Nous présentons notre solution sur le plan architectural et sur le plan logiciel.
Sur le plan architectural, nous proposons une solution dont l'architecture est basée sur SDN, qui est l'architecture par excellence pour le déploiement des plans de contrôle.
Sur le plan logiciel, notre solution est basée sur l'approche utilisant les protocoles de communication OpenFlow et GMPLS de façon simultanée. Cette approche innovatrice a été proposée pour la première fois en 2011 par Azodolmolky qui declare : « *To the best of our knowledge, this is the first time that integration of OpenFlow and GMPLS control plane for software-defined packet over optical networks has been proposed and experimentally demonstrated* » (Azodolmolky *et al.*, 2011 p.1).

Notre contribution est de bâtir un plan de contrôle OpenFlow-GMPLS pour les réseaux IP/Optiques respectant la technologie SDN, capable de gérer les équipements réseaux compatibles GMPLS ou non, et pouvant restaurer un lien réseau après rupture. Le tableau ci-dessous donne une représentation des composants initiaux clés (OpenFlow et GMPLS) à partir desquels il faut bâtir un plan de contrôle basé à la fois sur OpenFlow et GMPLS. Il en ressort aussi la différence des composants fondamentaux entre notre plan de contrôle et ceux des auteurs (recensés dans le tableau ci-dessous) qui nous ont précédés.

Tableau 1.1 Composants initiaux clés pour les plans de contrôle OpenFlow-GMPLS

	Composants initiaux clés
Auteurs qui nous ont précédés *Azodolmolky et al. (2011) ;* *Channegowda et al. (2013) ;* *Lei, Tsuritani et Morita (2012)*	**OpenFlow standard**
	GMPLS standard *- OSPF-TE standard* *- RSVP-TE standard*
Notre recherche	**OpenFlow standard**
	GMPLS-Dragon *- OSPF-TE Dragon* *- RSVP-TE Dragon*

Les composants de GMPLS-Dragon sont OSPF-TE Dragon et RSVP-TE Dragon. OSPF-TE Dragon est une extension du module OSPF de GNU Zebra, qui est elle-même une extension du OSPF standard. RSVP-TE Dragon est une extension de *l'open source KOM RSVP Engine*, qui est elle-même une extension de RSVP-TE standard.

Nous avons choisi GMPLS-Dragon plutôt que GMPLS standard parce que les équipements non-compatibles GMPLS peuvent fonctionner dans un réseau GMPLS à l'aide de GMPLS-Dragon (Bahnasy, Idoudi et Elbiaze, 2015). Aussi, les équipements optiques (Cisco 15 454) dont nous disposons dans notre laboratoire informatique et se trouvant dans notre architecture réseau ne sont pas compatibles GMPLS. En plus, nous avons utilisé dans notre plan de contrôle un composant SNMP/TL1 (Bahnasy, Idoudi et Elbiaze, 2015) issu des recherches précédentes sur GMPLS-Dragon afin de ne pas partir de zéro.

Une fois l'objectif et la solution proposée présentés, nous pouvons aborder la méthode de recherche.

1.3 Approche méthodologique

Cette section présente l'architecture générale d'un plan de contrôle OpenFlow-GMPLS, ainsi que toutes les phases et livrables par lesquels nous sommes passés pour mener à bien notre travail de recherche.

1.3.1 Présentation de l'approche générale d'un contrôleur OpenFlow-GMPLS

L'approche générale pour implémenter un plan de contrôle basé à la fois sur les protocoles OpenFlow et GMPLS, consiste à faire le déploiement de deux serveurs/contrôleurs dans une même architecture réseau, comportant une portion IP et une portion optique. L'installation, le paramétrage et la configuration du protocole OpenFlow sont effectués sur un serveur. Il en est de même sur l'autre serveur concernant le protocole GMPLS. Le contrôleur OpenFlow assure la gestion de la portion IP du réseau, alors que le contrôleur GMPLS assure la gestion de la portion optique. Le plan de contrôle OpenFlow-GMPLS vient assurer la gestion des contrôleurs OpenFlow et GMPLS, établir et gérer la communication entre eux comme le montre la figure ci-dessous.

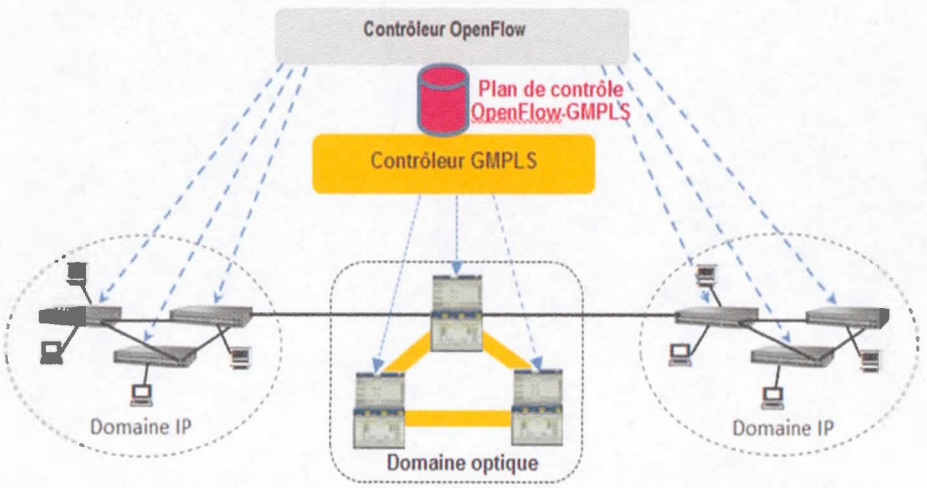


Figure 1.4 Présentation d'une architecture générale OpenFlow-GMPLS

1.3.2 Phases et livrables de l'approche méthodologique

Les phases et livrables par lesquels nous sommes passés pour mener à bien notre travail de recherche sont présentés ci-dessous.

- Phase 1 : appropriation du protocole OpenFlow et de quelques-unes de ses implémentations pour nous familiariser avec cet environnement. La version standard de OpenFlow de la *OF Fondation* constituera notre point de départ pour bâtir un plan de contrôle OpenFlow-GMPLS.
- Phase 2 : appropriation du protocole GMPLS-Dragon et de quelques-unes de ses implémentations pour aussi nous familiariser avec cet environnement. La version de GMPLS du projet Dragon constituera également notre point de départ pour bâtir un plan de contrôle OpenFlow-GMPLS.
- Phase 3 : conception et mise en œuvre physique d'une architecture réseau basée sur SDN, et qui convienne à notre solution.
- Phase 4 : extension du protocole OpenFlow à l'aide de trois éléments que nous avons conçu et développé afin que le contrôleur soit à même de faire une réinitialisation aisée d'un environnement réseau même complexe. Ces éléments permettront l'initialisation et la préparation de l'environnement réseau sans l'intervention de l'utilisateur. Le premier élément (Activation OpenFlow) est chargé d'activer automatiquement le protocole OpenFlow sur tous les commutateurs OpenFlow du réseau. Le second élément (Création GRE) est chargé de créer automatiquement toutes les interfaces GRE (*Generic Routing Encapsulation*) sur tous les équipements intermédiaires dans le domaine GMPLS. Le troisième élément (Démarrage GMPLS) est chargé de démarrer de façon automatique les composants GMPLS-Dragon sur tous les équipements intermédiaires dans le domaine GMPLS.
- Phase 5 : implémentation de l'agent OpenFlow-GMPLS qui intègre la technique d'interfonctionnement et de communication entre les plans de contrôle OpenFlow et GMPLS. Cet agent effectue la création automatiquement de deux LSP (*Label Switched Path*), un principal et un de sauvegarde entre un CSA (*Client System Agent*)

source et un CSA de destination. Le LSP de sauvegarde est chargé de remplacer immédiatement le LSP principal en cas de rupture de ce dernier.

- Phase 6 : description du comportement de notre plan de contrôle en cas de rupture de lien dans l'architecture réseau proposée. C'est-à-dire comment notre plan de contrôle pourra rétablir un nouveau lien après une rupture de lien. Pour cela, nous avons fait l'extension du protocole GMPLS-Dragon. Elle consiste en l'ajout d'un élément de reprise dans le domaine GMPLS-Dragon que nous avons conçu et implémenté.
- Phase 7 : expérimentation, évaluation et test de la solution dans notre laboratoire de réseau informatique.

Cette approche nous a permis de solutionner le problème parce qu'à chaque phase, nous avons obtenu les livrables explicites nous permettant de mener à bien notre recherche. Ces livrables sont représentés dans le tableau qui suit.

Tableau 1.2 Livrables issus de chaque phase du travail de recherche

Phase	Livrable
1	Rapport de : - synthèse des principales architectures des plans de contrôle OpenFlow - synthèse des composants utilisés dans les plans de contrôle basés sur OpenFlow - rôle et interaction entre les différents composants
2	Rapport de : - synthèse des principales architectures des plans de contrôle GMPLS - synthèse des composants utilisés dans les plans de contrôle basés sur GMPLS - rôle et interaction entre les différents composants
3	Architecture réseau basée sur OpenFlow et GMPLS conçue, installée en réel et configurée, respectant la technologie SDN
4	Extension de OpenFlow pour permettre l'initialisation automatique de l'environnement réseau en produisant du code informatique nécessaire (Appendice A1).
5	Implémentation de l'agent OpenFlow-GMPLS en produisant aussi du code informatique nécessaire (Appendice A4).
6	Extension de GMPLS-Dragon pour gérer les cas de reprise après rupture de lien, via du code informatique produit (Appendice A5).
7	Rapport de test et d'évaluation du plan de contrôle via les tableaux présentant les temps de traitement d'information dans chaque nœud de l'environnement réseau.

Cette recherche est menée dans un environnement d'expérimentation constitué des équipements de pointes pour les réseaux IP et optiques.

Il est composé de : cinq micro-ordinateurs, trois commutateurs optiques Cisco 15454, deux commutateurs électriques Cisco 6504 et un analyseur de paquet/données (EXFO FTB-500). Ces équipements sont disposés, installés et configurés pour correspondre à l'architecture réseau mise sur pied pour mener à bien notre recherche.

Et comme le disait si bien le philosophe français Auguste Comte, « Savoir pour prévoir, afin de pouvoir », nous allons faire l'état de l'art des travaux de recherche visant à apporter un éclairage sur la situation actuelle.

CHAPITRE II

REVUE DE LITTÉRATURE

Ce chapitre présente un regroupement des travaux publiés concernant l'implémentation des plans de contrôle pour les réseaux multicouches IP/Optiques. Il permet aussi de nous familiariser avec les recherches effectuées afin de clarifier notre réflexion pour nous engager sur des voix fructueuses.

2.1 Introduction

Cela fait deux décennies que plusieurs travaux de normalisation et de standardisation du protocole GMPLS ont été réalisés dans le but de mettre sur pied des plans de contrôle pour la gestion des réseaux multicouches IP/Optiques, afin d'assurer une gestion dynamique des domaines IP et optiques.

Aussi, en 2007 des travaux de modification et d'extension du protocole OpenFlow ont été effectués dans le même but. En plus, les plus grands constructeurs d'équipements réseaux ont aussi commencé à intégrer des modules compatibles OpenFlow sur leurs modèles, à l'instar de Cisco, HP, 3 Com, Jupiter, etc.

Notre état de l'art comporte deux points majeurs (les approches de recherche et l'architecture adéquate pour les plans de contrôle).

Dans ce chapitre, nous allons définir les concepts clés, présenter la genèse du domaine, aborder les principales approches de recherche et l'architecture nécessaire aux plans de contrôles pour les réseaux IP/Optiques.

Nous définissons dans la section qui suit les concepts clés de notre sujet de recherche afin de permettre une aisance dans la compréhension du sujet abordé.

2.2 Définitions des concepts clés

Dans cette section, les concepts clés que nous dégageons de notre sujet d'étude permettent une meilleure compréhension des termes utilisés. Nous avons ressorti trois concepts clés (plan de contrôle, réseau multicouche et réseau IP/Optique) que nous allons définir.

Plan de contrôle : applications permettant de gérer de l'extérieur via des API les équipements réseaux (routeurs, commutateurs, concentrateurs, convertisseurs, etc.), à partir d'une couche appelée couche application.

Plan de données : partie du réseau liée aux aspects ayant trait au format de trame, au débit de données et au matériel.

Réseau multicouche : réseau comportant plusieurs couches.

Réseau IP/Optique : réseau dont les flux de données sont transformés en paquets dans la portion IP et en lumière (longueur d'onde) dans la portion optique.

La figure ci-dessous illustre les concepts de plan de contrôle et de plan de données.

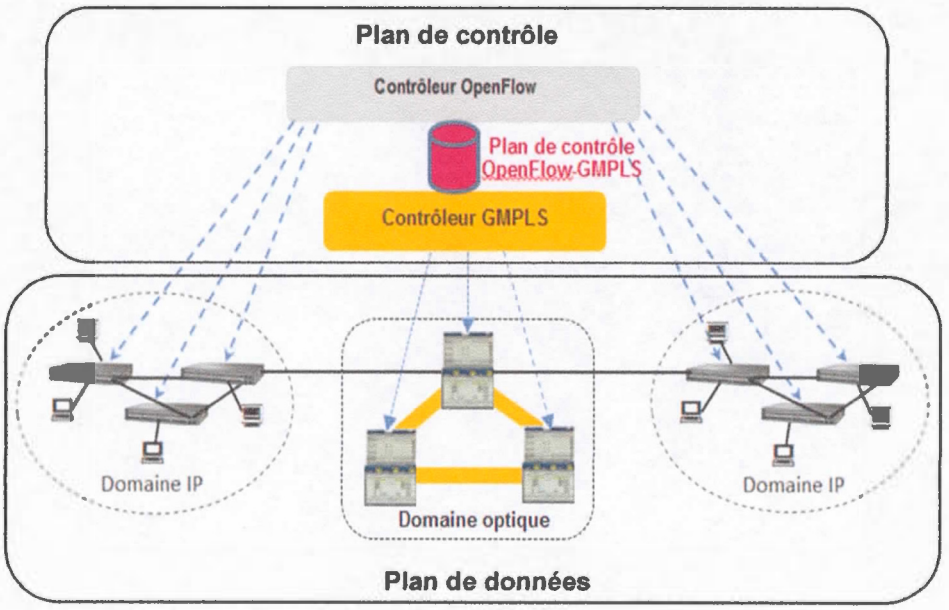


Figure 2.1 Illustration du concept de plan de contrôle et de plan de données

En utilisant une logique booléenne sur ces mots clés, nous avons pu dégager un grand nombre de travaux et de publications précis en rapport avec notre sujet. Nous avons extrait un regroupement axé sur la base des protocoles sous-jacents (OpenFlow, GMPLS) des approches de recherche.

Cette définition des concepts clés nous conduit à une brève présentation de l'historique du domaine.

2.3 Historique du domaine

Dans cette partie, nous présentons une brève évolution des protocoles GMPLS et OpenFlow.

Le souci d'avoir un plan de contrôle pour les réseaux IP/optiques remonte à près de deux décennies avec GMPLS et depuis 2007 avec OpenFlow.

En effet, depuis les années 1990, il y a eu progressivement une prolifération du trafic de données sur Internet via de nombreux services exigeants en bande passante tels que les jeux vidéo avec partenaires distants, la voix sur IP, la vidéo sur streaming, l'infonuagique (*cloud computing*), etc. Il fallait alors élaborer une nouvelle technologie pour augmenter la vitesse de transfert de données, et assurer une évolutivité aisée de l'Internet qui devenait de plus en plus difficile. En conséquence, une nouvelle technique appelée MPLS (*Multiprotocol Label Switching*) a été mis au point pour augmenter la vitesse de commutation des paquets de données. Dans la commutation traditionnelle des paquets IP, un routeur effectue une consultation de la table de routage et utilise la règle du « *longest prefix match* » pour déterminer le prochain saut (nœud) sur le chemin vers la destination en fonction de l'adresse IP de destination du paquet. En revanche avec MPLS, chaque paquet est marqué d'une étiquette, et un LSR (routeur à commutation d'étiquette) maintient également une table de commutation MPLS qui contient les mappages de [interface entrante, étiquette entrante] à [interface sortante, étiquette sortante]. Ainsi, quand un paquet marqué atteint un LSR, ce dernier consultera la table de commutation MPLS pour trouver une entrée qui correspond à l'étiquette et l'interface entrante du paquet. Une fois l'entrée correspondante trouvée, l'étiquette existante du paquet est supprimée. Ensuite, le paquet est étiqueté avec une nouvelle étiquette et expédié vers une interface de sortie selon le mappage. Cette approche augmente la vitesse de la transmission des paquets étant donné que le routeur n'a pas besoin

d'examiner le paquet. En outre, MPLS offre d'autres avantages tels que le TE (Ingénierie/technique de la circulation), le VPN, le transport de couche 2 et l'élimination des couches multiples. Malheureusement, MPLS ne s'applique qu'aux réseaux à commutation par paquets. Or Internet n'est pas seulement composé des réseaux à commutation par paquets, mais aussi de différents types de réseaux de transport qui utilisent des technologies « non-paquets » tels que les réseaux WDM (*Wavelength Division Multiplexing*), et TDM (*Time Division Multiplexing*). Par conséquent, une nouvelle suite de protocoles appelés GMPLS (*Generalized MPLS*) a été mise au point pour que le concept de MPLS puisse être appliqué à tous types de réseaux de transport sur Internet (Domancich et Wannasai, 2009).

L'historique de OpenFlow remonte vers 2006, lorsque Martin Casado, étudiant en doctorat à l'université de Stanford dans la Silicon Valley en Californie, a développé un prototype appelé Éthane. Conçu comme un moyen de gérer de manière centralisée la politique mondiale de réseau, il a utilisé un réseau basé sur les flux et un contrôleur axé sur la sécurité du réseau (OpenFlowNetworks.com, 2013). Cette idée a finalement conduit en 2007 à ce qui est connu comme OpenFlow, grâce à des recherches menées conjointement par des équipes des universités de Stanford et de Californie à Berkeley.

En 2011, la ONF (*Open Networking Foundation*) a été créée dans le but de standardiser les technologies émergentes à la pointe, et de faire la promotion de SDN et de OpenFlow. La version 1.1 qui est la première a été publiée en février 2011, et la seconde (version 1.2) a été supervisée par la ONF qui conserve le contrôle de la spécification. Les membres initiaux de cette fondation ont été les entreprises comme Google, Facebook et Microsoft. D'autres membres ont depuis rejoint la fondation à l'instar de Citrix, Cisco, Dell, HP, IBM, NEC, Huawei, Juniper Networks, Oracle, VMware, etc.

Après ce bref historique sur GMPLS et OpenFlow, nous passons à la présentation des approches de recherche.

2.4 Principales approches de recherche

Dans cette section, nous présenterons les trois principales approches de recherche concernant l'implémentation des plans de contrôle respectant l'architecture SDN.

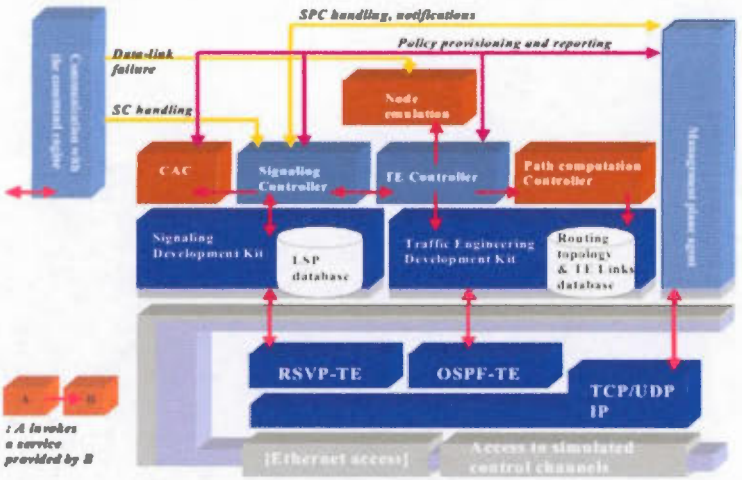
Note : toutes les figures présentées dans ce chapitre nous permettent de nous approprier des architectures conçues, des différents composants utilisés et de l'interrelation entre eux.

2.4.1 Approches basées sur GMPLS

Un UCP (plan de contrôle unifié) basé sur GMPLS consiste à la transformation (extension, réduction, modification, etc.) de ce protocole dans le but de fournir un plan de contrôle pour les réseaux multicouches IP/Optiques. Plusieurs chercheurs se sont orientés vers cette approche, et les techniques utilisées permettent de mettre en évidence la faisabilité. Nous éluciderons et montrerons de façon très brève les architectures des principaux auteurs ayant utilisés cette approche. Cela nous permet de comprendre (dans le cas d'une élucidation), le fonctionnement intrinsèque d'un plan de contrôle pour les réseaux multicouches IP/Optiques.

Comme plan de contrôle, GMPLS est devenu la suite de protocole de choix. Cependant, son adoption est confrontée à des défis majeurs en termes de faisabilité, de performance et de gain lors de la migration des paquets existants sur des réseaux optiques multicouches conduit par les plans de contrôle superposés. Un UCP constitue la base pour la construction d'un plan de contrôle unifié d'interopérabilité (Papadimitriou *et al.*, 2006).

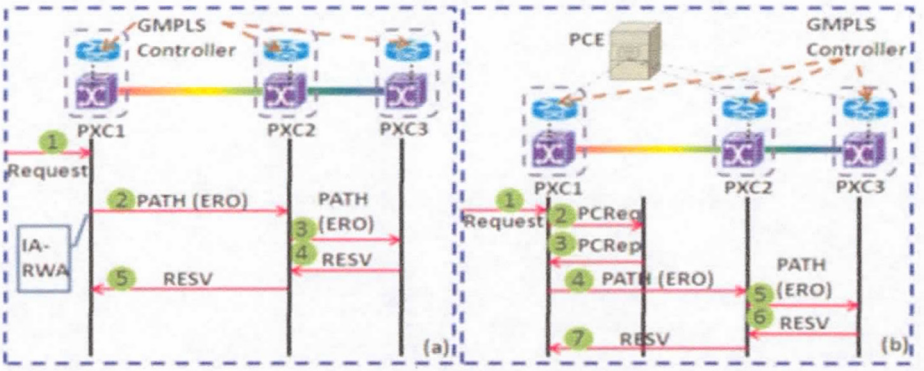
Le projet *ITEA TBONES* visait à s'attaquer au développement d'une plate-forme comprenant tous les éléments constitutifs de réseaux (dimensionnement de réseau, plan de gestion et plan de contrôle GMPLS). En plus, il visait aussi à démontrer la faisabilité de la mise en œuvre d'un plan de contrôle unifié GMPLS conforme pour les environnements multizones et multicouches. Les objectifs du projet comprenaient également la validation de la migration d'un modèle d'interconnexion de plan de contrôle superposé (nécessitant une instance de plan de contrôle séparée du plan de données par une couche de commutation) vers un modèle d'interconnexion de plan de contrôle unifié. Dans ce dernier, une instance de plan de contrôle gère un réseau comprenant plus d'une couche de commutation. Celle-ci constitue la base pour la construction d'un plan de contrôle unifié d'interopérabilité. Le schéma ci-dessous montre l'architecture réseau et logicielle qui a été utilisée dans le projet TBONES pour l'implémentation d'un plan de contrôle des réseaux multicouches basé sur GMPLS.



Source (Papadimitriou et al., 2006 3)

Figure 2.2 Architecture logicielle de TBONES

Dans la mouvance de cette approche de recherche, Lei, Tsuritani et Morita (2012b) disent que les plans de contrôle unifiés, basés sur SDN sont très importants pour les réseaux parce qu'ils peuvent permettre l'approvisionnement dynamique des chemins optiques, la restauration, l'amélioration de l'intelligence du réseau et la réduction considérable du temps de latence de traitement et des dépenses opérationnelles. Ces dernières années, il y a eu de grandes progressions dans ce domaine, allant de l'architecture GMPLS traditionnelle au PCE / GMPLS. Ces auteurs présentent les techniques habilitantes pour chaque plan de contrôle, et nous résume leurs avantages et leurs inconvénients. Le schéma de la page suivante donne les architectures basées sur GMPLS et sur PCE/GMPLS.



Source (Lei, Tsuritani et Morita, 2012b 1)

Figure 2.3 Architectures basées sur GMPLS et sur PCE/GMPLS

Les plans de contrôle des réseaux WSON (réseaux optiques à commutation de longueur d'onde) basés sur GMPLS ou PCE/GMPLS montrent bien qu'une solution basée sur GMPLS ou sur PCE/GMPLS est d'une très grande complexité. Nous n'avons pas pris la peine d'expliquer ces architectures, car nous n'avons pas l'intention d'évoluer vers cette direction. Mais ces solutions sont présentées afin de nous rassurer que nous avons parcouru toutes les solutions envisageables les plus connues, et de discerner les composants GMPLS dont nous avons besoin pour notre plan de contrôle. Le tableau de comparaison ci-dessous nous donne déjà une idée des différents facteurs à prendre en compte dans notre travail de recherche.

Tableau 2.1 Comparaison des plans de contrôle basés sur GMPLS, PCE/GMPLS et OpenFlow

Control Plane	Feature	Complexity	Flexibility Manageability	Technical Maturity	Standardization
GMPLS	Fully distributed	High	Low	High	Architecture framework standardized in 2004 [1]. Remaining issues are being discussed with the IETF CCAMP working group.
PCE GMPLS	Centralized path computation, distributed control	High	Medium	High	Architecture framework standardized in 2011 [2]. Remaining issues are being discussed with the IETF CCAMP and PCE working groups.
OpenFlow	Fully centralized	Low	High	Low	At a starting stage

Source (Lei, Tsuritani et Morita, 2012b 3)

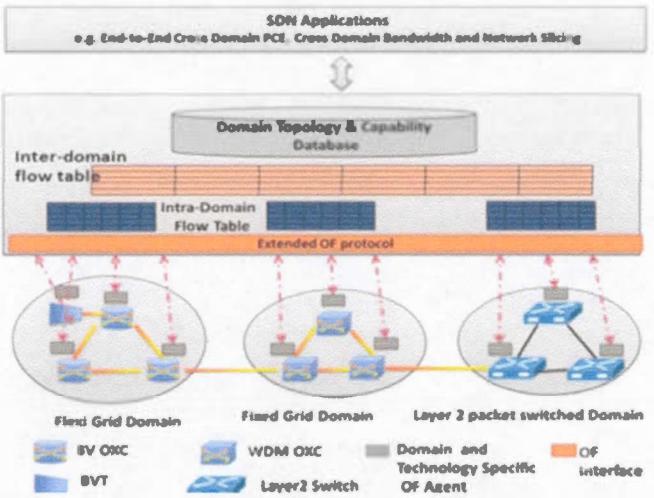
Après ces approches basées sur GMPLS, et le tableau ci-dessus évoquant à la troisième ligne les caractéristiques de OpenFlow, nous enchaînons avec les approches basées sur ce dernier.

2.4.2 Approches basées sur OpenFlow

Des recherches ont été faites sur la conception, la réalisation et l'évaluation expérimentale d'un plan de contrôle basé sur un PCE (élément de calcul de chemin), agissant comme un contrôleur OpenFlow. Un PCE est une composante fonctionnelle du plan de contrôle (physique ou logique), capable d'effectuer un calcul de chemin sur un graphique représentant un réseau ou une partie de réseau fixe et à grille flexible. Pour les recherches, les chercheurs ont fait une extension du protocole OpenFlow, sans traiter des aspects spécifiques tels que la connectivité intra-nœud et de déficiences physiques (Casellas *et al.*, 2013). Nous éluciderons de façon brève, certaines architectures pour nous permettre de comprendre le fonctionnement essentiel d'un plan de contrôle basé sur OpenFlow pour les réseaux multicouches IP/Optiques.

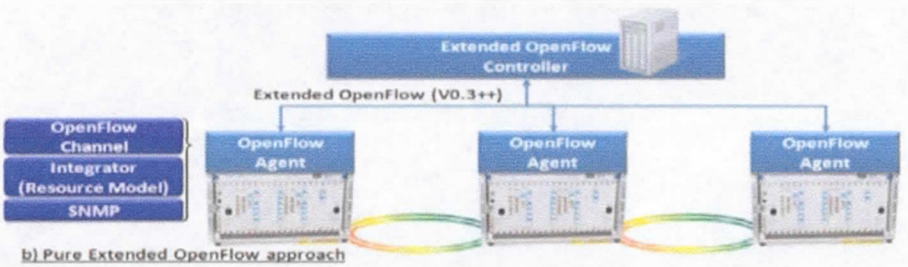
En effet, Channegowda *et al.* (2013) ont défini une approche basée sur une extension de OpenFlow (OF) en 2012 et 2013. Ils proposent une approche étendue OF pure, qui comprend un agent OF sur chaque NE (élément de réseau). L'agent développé utilise une API interne pour communiquer avec les éléments réseaux, et l'extension du protocole pour communiquer avec le contrôleur OF développé. Dans cette approche, le contrôleur utilise les demandes de fonctionnalités de commutation / réponses (ports, longueur d'onde, etc.) pour construire la topologie du réseau, et les messages *CFlow_MOD* (spécification de flux) pour contrôler les connexions croisées dans les commutateurs optiques. Une application réseau spécialisée dans le calcul de chemin (OF_PCE) est mise en œuvre et intégrée dans le contrôleur étendu. Il est chargé de calculer un trajet optique à l'intérieur du domaine optique en tenant dûment compte des contraintes de commutation.

Les schémas de la page suivante donnent l'architecture d'un plan de contrôle basé sur OpenFlow pour les réseaux multi-technologies et multi-domaines.



Source : (Channegowda et al., 2013 4)

Figure 2.4 Architecture UCP multi-domaines et multi-technologies

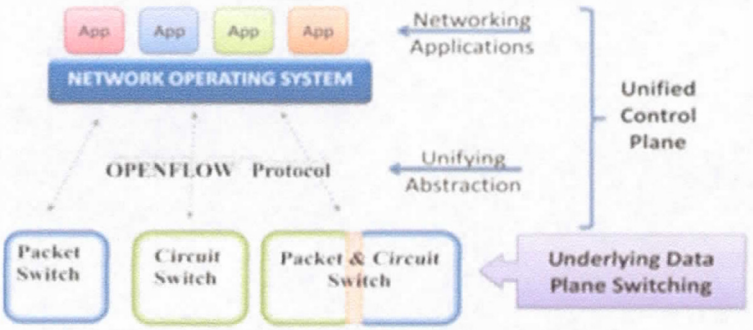


Source : (Channegowda et al., 2013 7)

Figure 2.5 Approche OpenFlow étendue

Aussi, Das *et al.* (2010) nous proposent un plan de contrôle basé uniquement sur OpenFlow. Ils soutiennent que OpenFlow est un exemple de mise en réseau défini par logiciel (SDN), où le matériel de commutation sous-jacent (commutateurs de paquets et de circuits) est contrôlé par l'intermédiaire d'un logiciel qui s'exécute sur un plan de contrôle externe découplé et automatisé. Les auteurs nous proposent un plan de contrôle composé de trois parties : un système d'exploitation réseau situé dans le NOX (contrôleur OpenFlow ou plan de contrôle OpenFlow) logé dans plusieurs contrôleurs logiciels, les applications réseaux qui fonctionnent sur le système d'exploitation réseau, et le protocole OpenFlow qui permet

d'accéder aux plans de données des commutateurs, à leurs tables de flux internes, à la configuration et aux statistiques. Le schéma ci-dessous montre l'architecture réseau proposée pour les réseaux multicouches IP/Optiques.

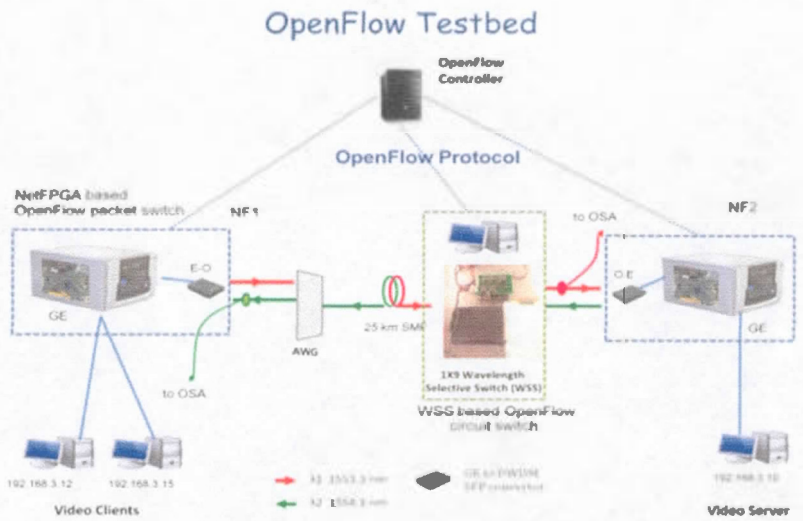


Source (Das et al., 2010 2) et (Gudla et al., 2010 2)

Figure 2.6 Architecture unifiée OpenFlow

De même, Gudla *et al.* (2010) nous présentent OpenFlow comme une architecture et un plan de contrôle unifié pour les réseaux multicouches IP/Optiques (paquet/circuit réseaux commutés). Ils démontrent sur un banc d'essai de validation simple, où un circuit de longueur d'onde (optique) bidirectionnelle est créé de façon dynamique pour transporter un flux TCP (paquet). En plus, ils ont mesuré la latence dans la création du circuit et croient qu'avec l'optimisation, le temps de liaison peut être réduit à moins d'une seconde. Leurs futurs travaux impliqueront l'élargissement du banc d'essai pour intégrer d'autres types de réseau comme TDM (multiplexage temporel), et les commutateurs à fibres. Ils visent également à démontrer la virtualisation unifiée du plan de données.

L'environnement d'expérimentation est présenté dans la figure ci-dessous :

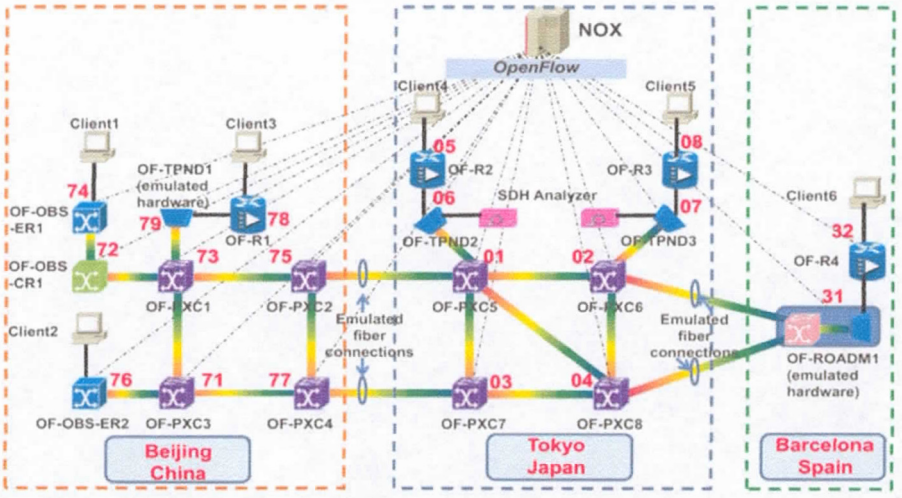


Source (Gudla et al., 2010 2)

Figure 2.7 Banc d'essai OpenFlow

De plus, Lei *et al.* (2013) proposent une technologie basée uniquement sur OpenFlow. Ils montrent que l'architecture SDN basée sur OpenFlow permettra aux opérateurs réseaux de contrôler le réseau à l'aide de logiciels s'exécutant sur un système d'exploitation réseau au sein d'un contrôleur externe (plan de contrôle externe). Cette architecture assurera aussi une flexibilité maximale aux opérateurs réseaux pour contrôler un réseau, étant donné son architecture centralisée permettant de simplifier et faciliter la gestion. Lei et al. (2013) montrent sur un champ de test un plan de contrôle unifié basé sur OpenFlow pour les réseaux multicouches à commutation optique, et vérifient la faisabilité et l'efficacité globale en évaluant quantitativement les latences pour la création du chemin de bout en bout et la restauration. Ces auteurs affirment aussi qu'au meilleur de leurs connaissances, un champ de test UCP axé sur OpenFlow pour les réseaux optiques est une première mondiale (Lei et al., 2013).

Le schéma d'expérimentation et le tableau présentant les temps de latence des différents chemins sont présentés dans les pages suivantes.



Source (Lei et al., 2013 6)

Figure 2.8 Procédure de création des chemins de bout en bout

Une fois qu'un nouveau flux IP arrive sur le routeur d'entrée (OF-R ou OF-OBS-ER), il transmet le premier paquet de ce flux au NOX (contrôleur exécutant un système d'exploitation réseau). Le NOX calcule l'itinéraire, attribue la longueur d'onde dans le domaine optique et ajoute ensuite une entrée de flux dans tous les nœuds le long de la route calculée dans un ordre séquentiel, à partir du routeur d'entrée comme illustré dans le schéma ci-dessus. Après l'entrée de flux, chaque VOFS (commutateur OpenFlow virtuel) dans les nœuds OBS (commutation de rafales optique)/WSON (réseau optique à commutation de longueur d'onde) contrôle automatiquement le matériel sous-jacent pour traverser les ports correspondants, grâce à la même vue virtuelle de la structure physique de la VOFS. Notons que si les paires BHP (paquet de tête à rafale) /DB (rafale de données) sont essentielles pour transmettre via l'intermédiaire du WSON, le NOX doit insérer une paire d'entrée de flux pour établir un groupe de chemins de lumière (chemin optique) comprenant au moins un chemin de lumière BHP et un DB pour le transfert de bout en bout BHP/DB.

Le sommaire des chemins est présenté dans le tableau ci-dessous.

Tableau 2.2 Résumé des chemins et temps de latence

No	Src	Dest	Calculated Path	Wavelength	SL(CP)	SL(Overall)	RL(CP)	RL(Overall)
(1)	Client1	Client2	74→72→73→71→76	W1,W2	~149 ms	~432 ms	~141 ms	~298 ms
(2)	Client3	Client4	78→79→73→75→01→06→05	W1	~138 ms	~388 ms	~132 ms	~257 ms
(3)	Client4	Client5	05→06→01→02→07→08	W2	~13 ms	~257 ms	~11 ms	~131 ms
(4)	Client4	Client6	05→06→01→02→31→32	W3	~181 ms	~424 ms	~170 ms	~290 ms
(5)	Client6	Client3	32→31→02→01→75→73→79→78	W4	~332 ms	~578 ms	~321 ms	~445 ms

Source (Lei et al., 2013 6)

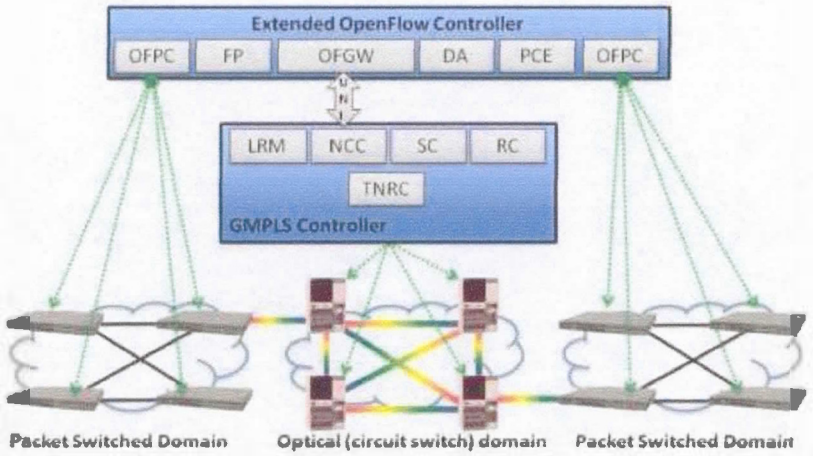
Après ces approches basées sur OpenFlow, passons maintenant aux approches basées à la fois sur OpenFlow et GMPLS.

2.4.3 Approches basées sur OpenFlow et GMPLS

Les auteurs ayant abordé cette approche fournissent une démonstration de la faisabilité de leurs solutions et proposent des architectures réseaux pour les plans de contrôle des réseaux IP/Optiques.

En effet, Azodolmolky et al., (2011) disent qu'ils ont été les premiers à proposer et démontrer expérimentalement l'intégration du plan de contrôle OpenFlow-GMPLS pour le *Software-Defined Packet* sur des réseaux optiques (Azodolmolky *et al.*, 2011 p.1). Auteur de plus de deux articles dans le domaine, Azodolmolky et ses co-auteurs proposent une architecture, un montage expérimental et le temps moyen d'installation de flux pour les différents flux optiques. La solution proposée est une architecture basée sur SDN qui intègre GMPLS et OpenFlow. Cette architecture utilise un plan de contrôle (contrôleur) OpenFlow étendu pour la commutation (distribution) de paquets, intégré à un plan de contrôle GMPLS dans un modèle de superposition.

Cette solution dite de superposition est représentée dans le schéma ci-dessous :



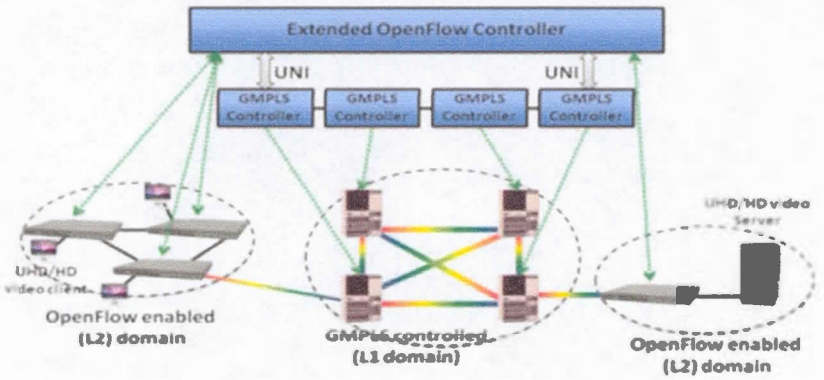
Source : (Azodolmolky et al., 2011 2)

Figure 2.9 Architecture du plan de contrôle OpenFlow-GMPLS intégré

D'une part, le plan de contrôle GMPLS suit le modèle standard de ASON (réseau optique à commutation automatique) et ses blocs de construction comprennent : le contrôleur de connexion réseau (NCC) chargé de gérer et de traiter les demandes de connexion; le contrôleur de signalisation (SC) qui implémente le protocole RSVP-TE et est chargé de gérer la signalisation GMPLS; le contrôleur de routage (RC) comprenant le protocole OSPF-TE et un algorithme de calcul de trajectoire pour calculer le chemin de bout en bout; le gestionnaire de ressources de lien (LRM) qui est responsable de la collection de suivi et d'information concernant le statut des éléments du réseau et le contrôleur de ressources de réseau de transport (TNRC) qui assure l'interface entre GMPLS et les éléments du réseau pour leurs configurations et surveillances.

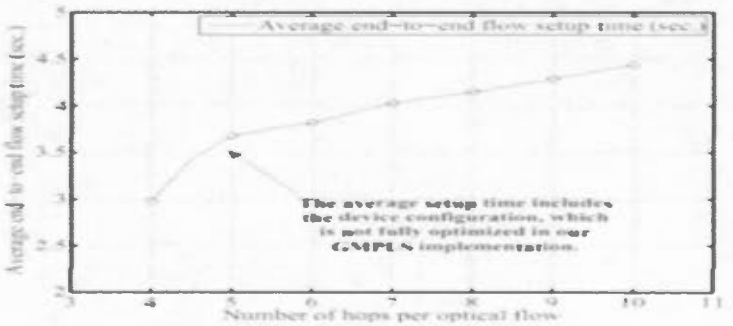
D'autre part, le contrôleur OpenFlow étendu comprend : le processeur de flux (FP) qui est responsable du traitement des nouveaux flux et crée des règles de flux et de mises à jour des tableaux de flux dans les commutateurs; l'élément de calcul de chemin (PCE) responsable du calcul de chemin d'accès pour chaque flux au sein de chaque domaine commuté par paquets; l'agent de découverte (DA) qui est responsable de la découverte de la topologie du réseau et

de la connectivité, y compris les points de terminaison dans chaque paquet de domaine commuté ; la passerelle OpenFlow (OFGW) qui fournit l'interface entre le contrôleur OpenFlow et le contrôleur GMPLS en passant par une interface utilisateur réseau (interface de signalisation UNI). Le contrôleur de protocole OpenFlow (OFPC) est responsable de l'interfaçage entre le contrôleur OpenFlow et les commutateurs où OpenFlow est activé. Les figures ci-dessous présentent une configuration expérimentale dans un banc d'essai, et les temps moyens d'établissement des flux de bout en bout en fonction du nombre de sauts par flux optique.



Source : (Azodolmolky et al., 2011 3)

Figure 2.10 Configuration expérimentale d'un banc d'essai

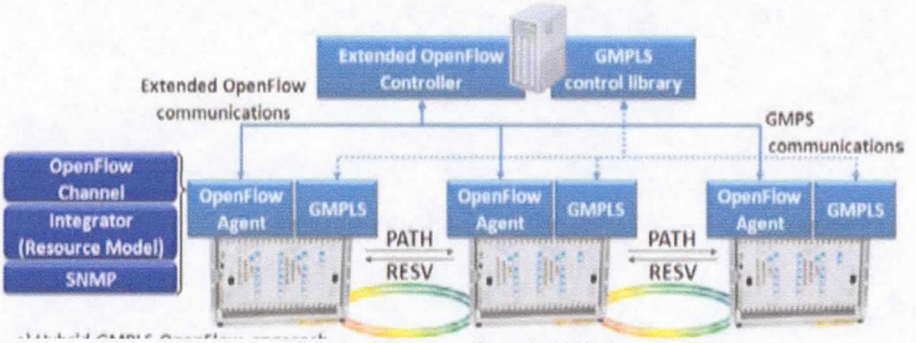


Source : (Azodolmolky et al., 2011 3)

Figure 2.11 Temps d'établissement des flux en fonction du nombre de sauts

Aussi, Channegowda *et al.* (2013) définissent une approche hybride dans laquelle chaque élément réseau (NE) fonctionne comme un commutateur où OpenFlow (OF) est

activé. Le contrôleur OF reçoit des informations concernant la topologie et les ressources à l'aide du protocole et de l'agent OF étendu. Par contre, le calcul de chemin d'accès, la création de chemin optique et le démontage sont effectués en utilisant le plan de contrôle GMPLS. L'agent OF étendu, le contrôleur et les applications de contrôle d'échantillon de réseau sont élaborés compte tenu de la création des chemins de lumière explicites. Dans le premier cas, seuls les ports et les éléments réseaux d'entrées et de sorties sont spécifiés et le plan de contrôle gère le calcul de chemin d'accès et sa mise en place. Dans le deuxième cas, le contrôleur est en mesure de préciser les détails complets du chemin optique; c'est-à-dire tous les commutateurs et ports situés le long du chemin optique. Le contrôleur vérifie aussi la faisabilité du chemin optique et effectue la mise en place. L'idée principale est de fournir une interface étendue au contrôleur pour réutiliser les fonctionnalités GMPLS. Le contrôleur OF étendu exploite la bibliothèque GMPLS pour calculer, établir et vérifier les chemins optiques à l'aide d'éléments de calcul des chemins intégrés, des contraintes de commutation et des fonctionnalités de péréquation puissantes. Ci-dessous la figure concernant l'approche GMPLS-OpenFlow hybride.



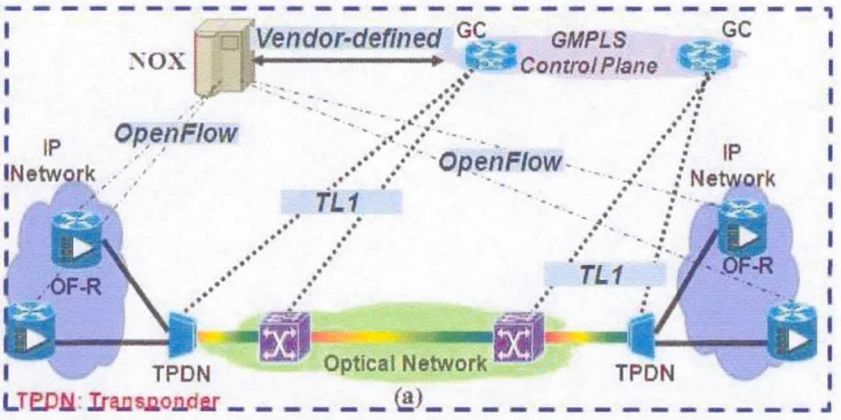
Source (Channegowda et al., 2013 7)

Figure 2.12 Approche GMPLS-OpenFlow hybride

En plus, Lei, Tsuritani et Morita en 2012, ont affirmé qu'un UCP pour les réseaux optiques multicouches IP/Optiques est très important pour les compagnies de réseaux. En effet, ils présentent une solution à l'heure actuelle qu'ils qualifient de logique, qui est d'introduire un plan de contrôle OpenFlow/GMPLS qui est capable d'utiliser GMPLS pour contrôler la couche optique, et OpenFlow pour coordonner et faire interagir les couches IP et optiques dynamiquement. Pour cela, les auteurs présentent expérimentalement trois solutions

d'interfonctionnement (parallèle, superposition et intégrée) entre OpenFlow et GMPLS pour le contrôle intelligent des chemins de longueur d'onde dans les réseaux multicouches IP/Optiques.

La solution parallèle se résume dans l'architecture ci-dessous :



Source (Lei, Tsuritani et Morita, 2012a 2)

Figure 2.13 Architecture de la solution parallèle d'un plan de contrôle OpenFlow-GMPLS

Dans la figure ci-dessus, le transfert de flux se déroule comme suit :

1- Lorsqu'un flux arrive à l'entrée OF-R1 (routeur IP compatible OpenFlow), son premier paquet est transmis au NOX (contrôleur OpenFlow ou plan de contrôle OpenFlow) s'il n y a aucune correspondance avec les entrées existantes dans la table de flux du OF-R1.

2- Le NOX exécute le calcul RWA (routage et affectation de longueur d'onde), et envoie un ERO (Objet de Route Explicite) au plan de contrôle GMPLS qui établit un LSP (*Label Switched Path*) dans le domaine optique via le signalement RSVP-TE (protocole de réservation de ressource avec l'ingénierie de trafic) et au même moment contrôle les TPNDs (brasseurs optiques).

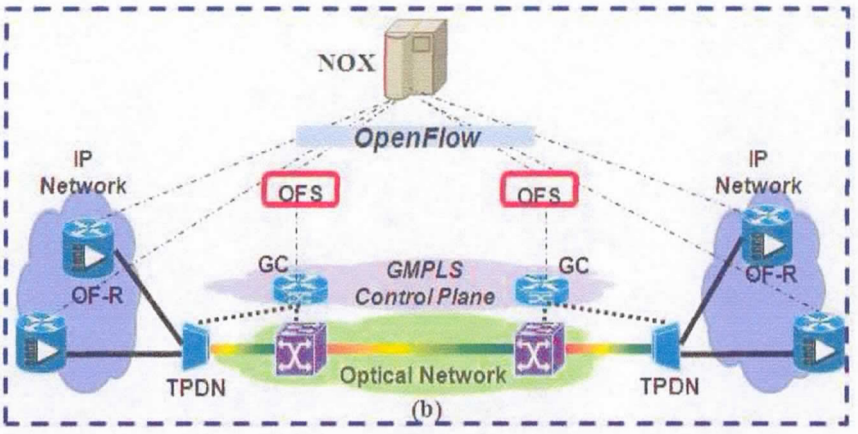
3- Un chemin optique est établi dans le domaine optique. Après que ce chemin soit configuré, une entrée de flux est insérée dans l'OF-R1 par le NOX. Le flux est alors transmis via le chemin optique et arrive à la sortie OF-R2.

La solution parallèle est le moyen le plus simple pour l'interfonctionnement OpenFlow/GMPLS. Mais dans ce cas, l'interface entre le NOX et le plan de contrôle GMPLS

n'est pas basée sur un protocole standard. Ce type d'interface non-standard peut poser des problèmes sérieux d'interopérabilité multifournisseurs puisque les vendeurs pourraient faire des interfaces différentes pour satisfaire leurs propres besoins. Par ailleurs, les nouveaux messages et les protocoles doivent être introduits dans le NOX, ce qui augmente la complexité de la conception. Pour résoudre ces problèmes, la solution de superposition est proposée.

C'est une approche de virtualisation qui introduit l'OFS (commutateur OpenFlow). Le NOX est capable de contrôler les couches optiques et IP via le protocole OpenFlow standard, ce qui est bénéfique pour aborder les questions d'interopérabilité multifournisseurs mentionnées ci-dessus et simplifie la conception du NOX. Il est mentionné comme solution de superposition parce que le plan de contrôle basé sur OpenFlow existe en tant que couche supérieure du plan de contrôle GMPLS.
Le fonctionnement diffère de la solution parallèle au niveau de l'étape N° 2. En effet, cette phase se déroule comme ci-dessous :
2 - Le NOX exécute le calcul RWA, et insère un flux d'entrée spécifié dans chaque OFS (Switch OpenFlow). Une extension de OpenFlow est nécessaire pour envoyer un ERO (objet de route explicite) au plan de contrôle GMPLS. Par ailleurs, si le RWA est mené par le plan de contrôle GMPLS, le NOX a seulement besoin d'insérer une nouvelle entrée de flux dans l'OFS d'entrée; ensuite ce dernier informe le plan de contrôle GMPLS pour terminer le calcul RWA et fournir des canaux optiques selon les résultats de calcul RWA. Selon le «*flow entry/ERO translation table*», l'OFS envoie un ERO au plan de contrôle GMPLS qui établit un LSP (*Label Switched Path*) dans le domaine optique via le signalement RSVP-TE (protocole de réservation de ressource avec l'ingénierie de trafic) et au même moment contrôle les TPNDs (brasseurs optiques).

Le schéma suivant donne l'architecture de la solution superposée d'un plan de contrôle OpenFlow-GMPLS.



Source (Lei, Tsuritani et Morita, 2012a 2)

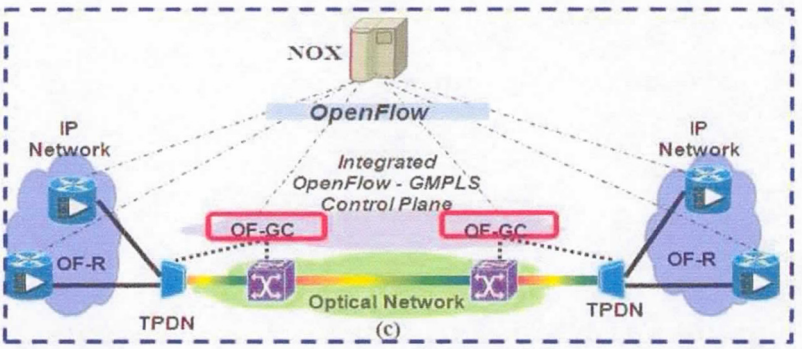
Figure 2.14 Architecture de la solution superposée d'un plan de contrôle OpenFlow-GMPLS

Avec l'introduction du commutateur OpenFlow, le contrôleur OpenFlow est capable de contrôler les deux couches optiques et IP, via le protocole OpenFlow standard qui est un protocole de communication réseau ouvert.

Concernant la solution dite intégrée, le fonctionnement diffère de la précédente au niveau de l'étape N° 2. En effet, cette phase se déroule comme ci-dessous :

2- Le NOX exécute le calcul RWA et insère un flux d'entrée spécifié (le long de la trajectoire calculée) dans chaque OF-GC (contrôleur GMPLS compatible OpenFlow) qui traduit ce flux selon le «*flow entry/ERO translation table*», et configure le LSP (*Label Switched Path*) dans le domaine optique via le signalement RSVP-TE et au même moment contrôle les TPNDs (brasseurs optiques).

Le schéma suivant nous donne l'architecture de la solution intégrée d'un plan de contrôle OpenFlow-GMPLS.



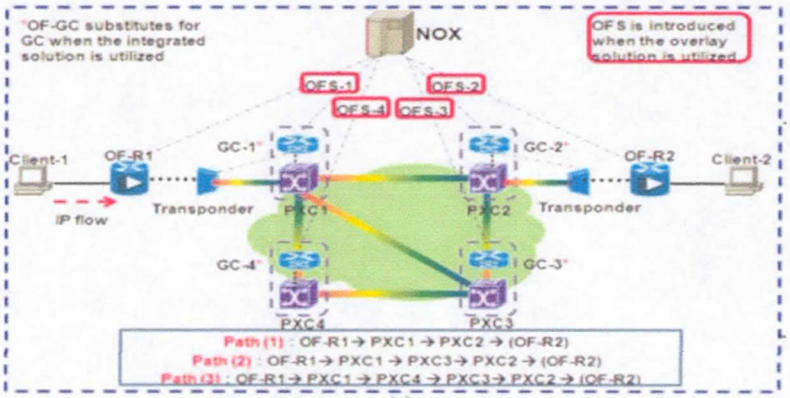
Source (Lei, Tsuritani et Morita, 2012a 2)

Figure 2.15 Architecture de la solution intégrée d'un plan de contrôle OpenFlow-GMPLS

Cette solution dite « intégrée » est presque similaire à la solution de superposition. La légère différence est que le contrôleur GMPLS et le commutateur OpenFlow sont fusionnés et intégrés dans un même contrôleur. Toutes les opérations échangées entre eux sont réalisées à l'intérieur de ce super contrôleur, ce qui est utile pour réduire la latence de traitement global, au détriment de la charge de traitement élevée.

Les résultats expérimentaux montrent que la solution intégrée surpasse les solutions de superposition et parallèle en termes de latence d'approvisionnement de chemin général, au détriment de la complexité de conception plus élevée et la charge de traitement dans un seul super contrôleur.

L'environnement d'expérimentation et les résultats des trois solutions suivent.



Source (Lei, Tsuritani et Morita, 2012a p. 3)

Figure 2.16 Dispositif expérimental

Tableau 2.3 Résumé des latences de provisionnement des chemins.

Path	Parallel solution			Overlay solution			Integrated solution		
(ms)	NOX Waiting	RSVP-TE	Total	NOX Waiting	RSVP-TE	Total	NOX Waiting	RSVP-TE	Total
(1)	1500	48.0	1503.3	1500	41.5	1504.8	500	39.7	503.8
(2)	1600	126.5	1603.6	1600	122.7	1605.9	600	128.0	604.6
(3)	1700	209.8	1703.3	1700	211.3	1707.2	700	206.2	705.5

Source (Lei, Tsuritani et Morita, 2012a p. 3)

Après avoir présenté les trois principales approches de recherche concernant l'implémentation des plans de contrôle respectant l'architecture SDN, nous présentons maintenant ladite architecture.

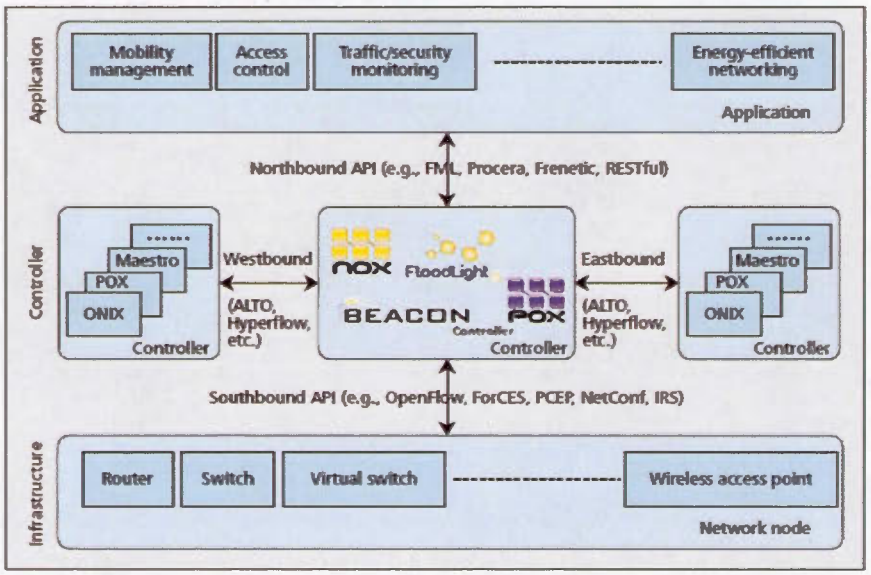
2.5 Architecture nécessaire pour un plan de contrôle dans les réseaux multicouches

Cette section donne plusieurs représentations de l'architecture adéquate pour la mise en œuvre des plans de contrôle dans les réseaux multicouches.

Le terme SDN (réseau défini par logiciel) a été inventé au cours des récentes années. Toutefois, le concept derrière SDN a évolué depuis 1996, animé par le désir de fournir la gestion contrôlée par l'utilisateur de la transmission dans les nœuds de réseau. Les études menées à ce jour dans le but de trouver une architecture adéquate pour un plan de contrôle dans les réseaux en général, et dans les réseaux IP/Optiques en particulier ont montré que l'architecture SDN reste la plus prometteuse (Gringeri, Bitar et Xia, 2013 ; Ji, 2012 ; Jiayuan, Ying et Dittmann, 2013 ; Patel, Ji et Ting, 2013 ; Simeonidou, Nejabati et Channegowda, 2013). Les architectures Éthane en 2007 et OpenFlow en 2008 ont amené la mise en œuvre de SDN plus proche de la réalité (Sezer *et al.*, 2013). Le fait d'adopter l'architecture SDN peut améliorer la commodité de gestion de réseau, son évolutivité et son dynamisme dans un centre de données d'entreprise (Azodolmolky, Wieder et Yahyapour, 2013). De même, cette architecture aussi appelée architecture divisée, permettra aux opérateurs réseaux de gérer de façon flexible les équipements réseaux à l'aide de logiciels s'exécutant sur des serveurs externes. En plus, cette architecture peut être exploitée pour offrir de façon efficace des services innovants, en améliorant la qualité de service et l'expérience des utilisateurs finaux. Si elle est appliquée convenablement, elle peut aussi entraîner des économies *CAPEX* (coûts d'investissement), et *OPEX* (les coûts récurrents) pour les opérateurs à long terme. Dans les

réseaux à paquets, la nécessité d'adopter SDN vient principalement de nouvelles exigences réseaux imposées par de nouveaux services (services mobiles haut débit, contenus vidéo déployés dans les réseaux d'accès, services d'infonuagique et de centre de données) (Shirazipour *et al.*, 2012).

Sezer et al., (2013) présentent clairement les différences entre l'architecture des réseaux traditionnels d'aujourd'hui (figure 1.1), et l'architecture SDN (figure 1.2). Vu sous un autre angle, on obtient la figure suivante où ressort clairement la séparation entre le plan de données (couche infrastructure) et le plan de contrôle (couche application).

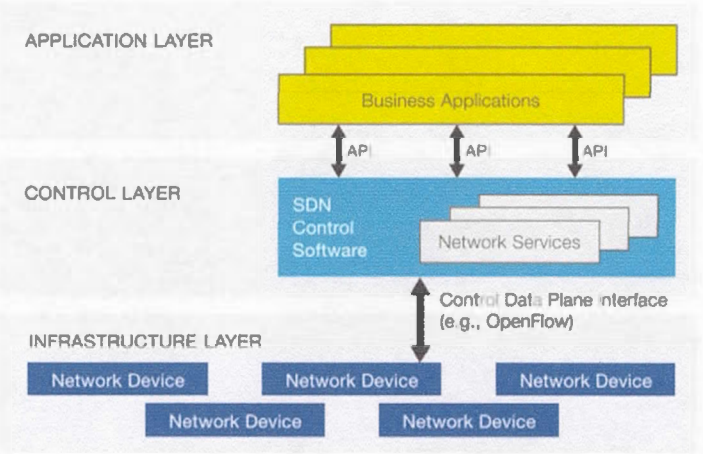


Source : (Sezer et al., 2013 37)

Figure 2.17 Architecture fonctionnelle de SDN

Aussi, la « *Open Networking Foundation* » du fait des tendances de l'industrie de la réseautique (explosion des appareils mobiles et des types de contenu échangés par les internautes, virtualisation des serveurs, avènement des services en nuage, etc.), a réexaminé l'architecture réseau traditionnelle d'aujourd'hui pour nous proposer une architecture SDN.

La vue logique de cette architecture se présente comme suit :



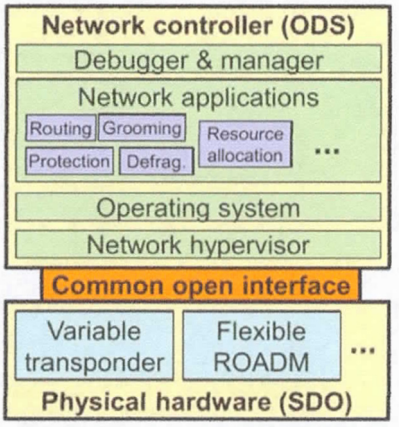
Source : (Foundation, 2012 7)

Figure 2.18 Architecture SDN

Dans cette architecture, l'intelligence réseau est (logiquement) centralisée dans les contrôleurs logiciels basés sur SDN, qui maintiennent une vue globale du réseau. En conséquence, le réseau apparaît à l'application et aux moteurs de règles comme un seul commutateur logique. Avec SDN, on peut prendre le contrôle du réseau entier à partir d'un point logique unique, ce qui simplifie grandement la conception et le fonctionnement réseau. SDN simplifie aussi fortement les périphériques réseaux, car on n'a plus besoin de comprendre et d'exécuter des milliers de processus standards des protocoles, mais simplement d'accepter les instructions du contrôleur SDN.

En plus, Ji (2012) présente une architecture SDN personnalisée au domaine optique. En effet, cette structure appelée SDON (*Software Defined Optical Network*), réseau optique défini par logiciel contient trois éléments principaux. L'élément le plus bas est la couche physique matérielle ou couche de données (transmetteurs, receveurs, nœuds de commutation, amplificateurs, etc.). L'élément supérieur ou couche d'application est un contrôleur intelligent qui utilise la flexibilité du matériel physique pour gérer le réseau et effectuer une optimisation et une personnalisation de réseau. Outre les applications, le contrôleur réseau

comprend également l'hyperviseur de réseau, le système d'exploitation, le débogueur et le gestionnaire. Faisant écho à la terminologie de SDO (*Software Defined Optics*), ce contrôleur peut être appelé optique défini par logiciel (ODS) *Optics Defining Software*. Le schéma ci-dessous montre la structure de cette architecture.



Source : (Ji, 2012 2)

Figure 2.19 Architecture SDON

Après avoir défini les concepts clés, parcouru brièvement l'historique du domaine, évoqué les principales approches de recherche et l'architecture nécessaire pour l'implémentation d'un plan de contrôle, quelle conclusion pouvons-nous en tirer ?

2.6 Conclusion

Les solutions proposées et les plus avancées à ce jour pour centraliser de façon logique l'intelligence et l'état du réseau à partir des applications dans des réseaux multicouches IP/Optiques sont celles utilisant les protocoles GMPLS et/ou OpenFlow dans une architecture SDN. Trois principales approches de recherche concernant l'implémentation des plans de contrôle respectant l'architecture SDN ont vu le jour.

Nous avons l'approche basée sur GMPLS uniquement, où les recherches se focalisent sur des modifications ou des extensions de GMPLS pour bâtir des plans de contrôle pour les réseaux IP/Optiques.

Nous avons aussi l'approche basée sur OpenFlow uniquement, où les recherches sont centrées sur des modifications ou des extensions de OpenFlow pour bâtir des plans de contrôle pour les réseaux IP/Optiques.
Nous avons enfin l'approche basée à la fois sur OpenFlow et GMPLS, où les recherches sont axées sur des modifications ou des extensions de OpenFlow et de GMPLS pour implémenter des plans de contrôle pour les réseaux IP/Optiques, en utilisant OpenFlow pour la gestion de la portion IP et GMPLS pour la gestion de la portion optique.

L'architecture adéquate des solutions utilisant toutes ces approches est SDN. Elle fait la quasi-unanimité pour l'Internet du futur si les opérateurs et fournisseurs de services réseaux veulent respecter les exigences sans cesse en croissance exponentielle des applications utilisées via le réseau, et qui demandent de plus en plus de bande passante, de flexibilité d'administration et d'évolution. Quel que soit sa spécification ou sa personnalisation, cette architecture est appropriée pour bâtir des plans de contrôle pour les réseaux IP/Optiques.

Malgré le fait que les solutions proposées et implémentées à ce jour restent encore au stade d'essai en laboratoire, les chercheurs et praticiens avancent de plus en plus vers une solution pour une exploitation réelle. En effet, les plus grands constructeurs des nœuds réseaux (routeurs, commutateurs, hubs, etc.), à l'instar de Cisco System, HP, 3Com, etc., intègrent déjà progressivement sur certains modèles des compatibilités avec le protocole OpenFlow, ce qui permettra de faire des tests en réel sans perturbation du réseau informatique en exploitation. Mais plusieurs équipements réseaux sont encore non-compatibles GMPLS, et les solutions basées sur OpenFlow et GMPLS n'intègrent pas ce genre d'équipements.
Après cette recension des écrits concernant les approches de recherche et l'architecture adéquate pour l'implémentation des plans de contrôle, nous passons à l'étude de GMPLS.

CHAPITRE III

ÉTUDE DE GMPLS

Pour cette étude, nous nous sommes intéressés au projet DRAGON (*Dynamic Resource Allocation in GMPLS Optical Networks*) dont le déploiement dans notre architecture réseau de certains de ces composants logiciels et configurations matérielles sont un préalable au fonctionnement du contrôleur OpenFlow-GMPLS que nous avons implémenté.

Il était très important pour nous d'apprivoiser les composants et les configurations matérielles dudit projet afin de pouvoir les intégrer à notre architecture réseau et les adapter à nos besoins.

3.1 Introduction

Le projet DRAGON, financé par la NSF (*National Science Foundation*) s'occupe de la recherche et du développement des services dynamiques de transport des réseaux déterministes de bout à bout et gérable pour les applications haut de gamme (vidéo haute définition, grand flux de données, etc.). Pour sa mise en œuvre, DRAGON déploie l'infrastructure de réseau IP et crée un GMPLS (*Generalized Multi-Protocol Label Switching*) compatible avec le cœur du réseau optique pour permettre un approvisionnement dynamique de chemins d'accès des réseaux déterministes en réponse directe aux demandes des utilisateurs finaux, s'étendant sur plusieurs domaines administratifs.

Les informations contenues dans ce chapitre sont tirées du guide nommé « *Virtual Label Switching Router Implementation Guide* », écrit par l'USC (*University of Southern California*), l'ISI (*Information Sciences Institute*), l'UMD (*Université du Maryland*), la MAX

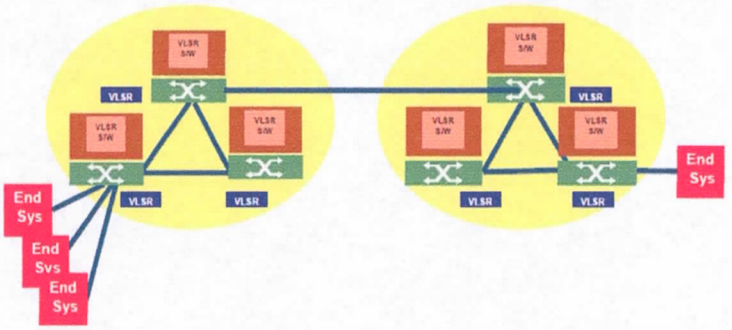
(*Mid-Atlantic Crossroads*) et le GMU (*George Mason University*). Ce guide fournit une description de la mise en œuvre d'une partie de la suite logicielle de DRAGON. Dans ce chapitre, nous allons étudier, installer et configurer certains composants du plan de contrôle DRAGON pour une prise en main totale.

Après cette brève introduction, nous passons aux éléments qui composent le plan de contrôle DRAGON.

3.2 Composants du plan de contrôle DRAGON

Cette section présente les éléments fonctionnels clés identifiés dans l'architecture du plan de contrôle DRAGON.

En effet, l'utilisation de certains logiciels et composants du projet DRAGON permet de mettre en place les possibilités d'approvisionnement dynamique des chemins dédiés à travers le réseau pour les applications haut de gamme. Ceci est accompli via un VLSR (routeur virtuel à commutation par étiquettes). Généralement, sur une topologie basée sur GMPLS, le chemin approvisionné dynamiquement se fait d'un nœud d'extrémité vers un nœud d'extrémité. Toutefois, si le logiciel DRAGON s'exécute sur les systèmes d'extrémités, le chemin d'accès peut être établi de bout en bout. Nous avons ci-dessous un exemple de déploiement des VLSR.



Source : ((USC) et al., 2008 2)

Figure 3.1 Diagramme général pour le déploiement des VLSR

L'architecture de DRAGON utilise GMPLS comme composante fondamentale pour le contrôle et la mise en service d'éléments réseaux. Les éléments fonctionnels clés se trouvant dans l'architecture du plan de contrôle DRAGON sont :

- VLSR (*Virtual Label Switch Router*)
- NARB (*Network Aware Resource Broker*)
- CSA (*Client System Agent*)
- ASTB (*Application Specific Topology Builder*)

Il existe deux principaux modes de fourniture des services dynamiques (c'est-à-dire les circuits) qui sont :

- CSA-to-CSA
- VLSR-to-VLSR

Commençons par définir les rôles de ces éléments fonctionnels clés.

3.2.1 VLSR

En tant que composant clé de notre plan de contrôle OpenFlow-GMPLS, le VLSR est l'ensemble composé d'un PC (où DRAGON est installé) et d'une structure de commutateur. Ce PC doit exécuter un logiciel de plan de contrôle basé sur GMPLS. Son utilisation principale dans le projet DRAGON est de contrôler les commutateurs Ethernet via un contrôleur GMPLS. Ce routeur virtuel permet de mettre en place l'approvisionnement dynamique de routes dédiées, et fournit un mécanisme pour intégrer les équipements non-compatibles GMPLS dans un réseau qui offre des services GMPLS de bout en bout. Le VLSR transforme la suite de protocoles standards GMPLS par des protocoles d'équipements spécifiques, afin de permettre la reconfiguration dynamique des équipements non-GMPLS.
Le PC VLSR se compose d'une pile de protocoles qui comprend OSPF-TE (Kompella et Rekhter, 2005) et RSVP-TE (Berger, 2003) et agit comme un proxy pour les périphériques non-GMPLS. Ceci permet à ces derniers d'être inclus dans les instanciations des chemins de bout à bout. Toutefois, le VLSR a également été adapté pour contrôler certains commutateurs TDM et optiques.
L'application de VLSR aux environnements Ethernet a inclus la modification de OSPF-TE et RSVP-TE pour permettre la fourniture des circuits Ethernet basés sur les configurations VLAN. À chaque commutateur Ethernet, le VLSR traduit les messages de signalisation RSVP-TE en commandes de commutateur local et crée les associations VLAN-ports désirées ainsi que les bandes passantes requises garanties. Chaque fois qu'un circuit Ethernet (LSP) est mis en place ou coupé, la bande passante et les informations d'étiquetage de VLAN sont mis

à jour par la distribution d'annonces à état de lien OSPF-TE (LSAs) afin de maintenir les états de liens appropriés sur le réseau. Le PC VLSR utilise une combinaison de SNMP RFC 2674 et CLI (Interface de Ligne de Commandes) pour le contrôle local du Switch Ethernet.

Il est à noter que la version actuelle du code de VLSR prend en charge un nombre très limité de commutateurs.

Tableau 3.1 Commutateurs prises en charge par le code VLSR de DRAGON

Nom du commutateur	Modèle	Méthode de contrôle
Dell PowerConnect 5224/5324	RFC2674	SNMP
Dell PowerConnect 6024/6024F	RFC2674	SNMP
Extreme Summit 1i/5i	RFC2674	SNMP
Intel Express 530T	IntelES530	SNMP
Raptor ER1010	RaptorER1010	SNMP
Cisco Catalyst3750	Catalyst3750	SNMP
Cisco Catalyst6550	Catalyst6550	SNMP
HP 5406	HP5406	SNMP
SMC10G8708	SMC 10G 8708	SNMP
Force10 E300/E600	Force10E600	CLI (SSH/TELNET)
Linux Software Switch	LinuxSwitch	Shell/SSH/TELNET
EoS Subnet	N/A	TL1

Ce tableau est présenté pour marquer le fait que les commutateurs de marque Cisco, de modèle ONS 15454 avec lesquels nous avons travaillé ne sont pas pris en charge dans le code VLSR du projet DRAGON. Nous allons par conséquent adapter notre plan de contrôle OpenFlow-GMPLS Dragon, afin que le code VLSR-Dragon soit en mesure de fonctionner avec les commutateurs Cisco 15454.

3.2.1.1 DRAGON OSPF-TE

Le logiciel DRAGON OSPF-TE est une extension du module OSPF de GNU Zebra (package de logiciel de routage *open source*).

OSPF est un protocole de routage à état de lien développé pour les réseaux IP. Il envoie les LSAs (Annonce à État de Lien) à tous les autres routeurs dans la même zone. Chaque routeur OSPF maintient une base de données identique décrivant la topologie du réseau. À partir de cette base de données, une table de routage est calculée en construisant un arbre de chemin d'accès plus court vers chaque nœud. OSPF recalcule les routes rapidement face aux changements topologiques en utilisant un minimum de trafic.

OSPF porte également les informations d'état de lien pour GMPLS en possédant l'OSPF-TE. Le module CSPF (*Constraint Shortest Path First*) est fourni comme un module séparé qui inclut une API sous forme d'appel fonction et offre la possibilité de calculer les chemins d'ingénierie de trafics basés sur le LSDB (*Link State Database*) OSPF-TE dérivé.

3.2.1.2 DRAGON RSVP-TE

Le logiciel DRAGON RSVP-TE est une extension de l'*open source KOM RSVP Engine* de l'Université Technique de Darmstadt pour inclure la fonctionnalité requise de GMPLS TE.

RSVP est un protocole de signalisation qui permet à l'expéditeur et au destinataire dans une communication, de mettre en place une large bande réservée pour la transmission de données avec une qualité de Service (QoS) spécifiée.

3.2.2 CSA

Le CSA (*Client System Agent*) est l'un des composants essentiels de notre plan de contrôle OpenFlow-GMPLS. C'est un logiciel qui s'exécute sur tout système qui termine un lien de plan de données du service fourni. C'est le logiciel qui permet l'approvisionnement de la demande de bout en bout entre les systèmes clients dans le domaine GMPLS. Dans ce contexte, un client est un système qui demande des services réseaux. Le CSA s'exécute généralement en mode *peer-to-peer*, mode de superposition via un protocole d'interface usager-réseau (UNI) ou en mode de service web.

L'architecture DRAGON identifie trois types distincts de modes opératoires CSA.

- *Peer-to-peer* CSA
  - CS *(Client System)* qui termine le lien du plan de données,
  - CS qui termine le lien du plan de contrôle,
  - CS qui possède à la fois RSVP et OSPF.

Note : dans ce mode de fonctionnement, le CSA ressemble beaucoup à un VLSR.

- UNI CSA
  - CS qui termine le lien du plan de données,
  - CS qui termine le lien du plan de contrôle,
  - CS qui utilise le signalement UNI (basé sur RSVP).

- Service Web CSA
  - CS qui termine le lien du plan de données,
  - CS qui termine le lien du plan de contrôle,
  - CS qui utilise XML pour demander la mise en service.

3.2.3 DRAGON NARB and RCE

NARB (*Network Aware Resource Broker*) est utilisé pour le calcul sophistiqué des chemins d'accès et l'ingénierie de trafic, où les systèmes d'extrémités ou autres dispositifs peuvent faire des interrogations pour connaître la disponibilité des voies de trafic conçues entre les paires sources et destinations spécifiées. C'est une entité qui représente le système autonome local (AS) ou le domaine. L'abstraction de domaine fournit des mécanismes pour un domaine administratif afin d'annoncer au monde extérieur une vue très simplifiée de sa topologie. Cela permet aux domaines de cacher leurs véritables topologies, mais aussi de réduire la quantité de mises à jour externes requises. Chaque domaine administratif peut utiliser des paramètres de configuration pour personnaliser son abstraction du domaine au niveau désiré.

Un sous-composant autonome du NARB appelé RCE (*Resource Computation Element*) effectue les tâches de calcul de chemin d'accès inter-domaine et la mise en service du LSP (*Label Switched Path*). Le NARB est également responsable du routage inter-domaine.

3.2.4 ASTB

L'architecture de DRAGON inclut l'idée de créer des AST (*Application Specific Topologies*). Celles-ci sont requises par un utilisateur final et sont généralement un ensemble de LSP qu'un domaine d'application désire mettre en place en tant que groupe. L'élément DRAGON, connu comme ASTB (*Application Specific Topology Builder*) est chargé de coordonner cela en réponse aux demandes de l'application. L'ASTB utilise les fonctionnalités des autres éléments du plan de contrôle DRAGON (NARB, VLSR, CSA) pour demander une instanciation individuelle des LSP, maintenir la cartographie des différents groupements LSP aux topologies spécifiques et interagir avec les demandes des utilisateurs.

Après avoir parcouru les quatre composants du plan de contrôle DRAGON, passons maintenant à son installation.

3.3 Installation du logiciel DRAGON

Nous présentons dans cette section l'installation de l'environnement DRAGON et les conditions nécessaires pour cela.

3.3.1 Préparation avant l'installation

Les configurations requises se situent au niveau du matériel et du système d'exploitation.

a) Le matériel

Au moins 500 MHz de vitesse du processeur (Pentium III) et 256 Mo de RAM sont nécessaires pour exécuter le système d'exploitation et le logiciel DRAGON. Après l'installation du système d'exploitation, il faut au moins 1 Go d'espace libre sur le disque dur. Chaque machine doit avoir au moins deux cartes réseau Ethernet (*FastEthernet*) ; *Gigabit Ethernet* est préférable mais pas indispensable.

b) Le système d'exploitation

Les hôtes et les ordinateurs de contrôle VLSR peuvent fonctionner avec Linux (version du noyau 2.4.20 ou supérieur), FreeBSD (version du noyau 4.11), ou une version hybride de Linux et FreeBSD.

3.3.2 Prérequis avant l'installation

Le logiciel « *DRAGON dependency* » est requis avant l'installation du logiciel DRAGON. Il comprend les outils SSH, GNU, Net-SNMP, SVN, libxml2 et zlib-1.2.3. Le rôle de chaque outil est expliqué ci-dessous.

3.3.2.1 SSH

SSH (*Secure Shell*) est à la fois un programme informatique et un protocole de communication sécurisé qui permet la connexion sécurisée à un ordinateur distant. On peut déplacer des fichiers d'une machine à l'autre avec des communications sécurisées via des canaux non sécurisés. Sans SSH, quelqu'un qui a un accès « *root* » sur les ordinateurs du

réseau ou un accès physique au câble réseau, peut accéder sans autorisation aux fichiers de plusieurs façons.

3.3.2.2 Les compilateurs GNU

a) gcc/g++

Les logiciels de DRAGON ont été compilés sous le nom de GNU GCC/G++, versions 2.95.x, 3.2.x, 3.4.x, et 4.0.x.

b) bison

C'est le générateur d'analyseur syntaxique de GNU pour interpréter les fichiers d'extension « .y ».

c) flex

C'est le générateur d'analyseur lexical rapide pour interpréter les fichiers d'extension « .l ».

3.3.2.3 Net-SNMP

Net-SNMP (*Net-Simple Network Management Protocol*) est une suite d'applications utilisées pour implémenter le SNMP v1, SNMP v2c et SNMP v3, et qui utilise à la fois IPv4 et IPv6. Le logiciel DRAGON nécessite Net-SNMP pour son installation. Cette suite d'applications comprend :

- une application en ligne de commande,
- un navigateur graphique MIB,
- l'application *daemon* pour la réception des notifications SNMP,
- un agent extensible pour la gestion de l'information et qui permet de répondre aux requêtes SNMP,
- une bibliothèque pour le développement de nouvelles applications SNMP avec les APIs C et Perl.

SNMP (*Simple Network Management Protocol*) est le protocole le plus utilisé pour la gestion des éléments actifs du réseau (commutateurs, routeurs, serveurs, postes de travail, imprimantes, etc.) et/ou des logiciels. Chaque élément du réseau dispose d'une entité dite agent qui répond aux requêtes de la station de supervision (ordinateur exécutant les applications de gestion qui contrôlent les éléments réseaux). Les agents sont des modules qui

résident dans les éléments réseaux. SNMP permet à la station de supervision d'aller chercher des informations sur les éléments réseaux et de recevoir des alertes provenant de ces derniers. Il est aussi utilisé pour la gestion à distance des applications comme les bases de données et des équipements comme les serveurs.

3.3.2.4 SVN

SVN (*SubVersioN*) est un système de contrôle de version *open source* utilisé pour le contrôle des sous-versions. Cela signifie que si une modification incorrecte a été apportée aux données, les données précédentes ne sont pas perdues parce que tout le travail est sauvegardé. Ainsi, peu importe le nombre de modifications qui sont effectuées dans un fichier, le fichier original ainsi que toutes les autres versions du fichier peuvent toujours être référencés à tout moment.

3.3.2.5 Libxml2

La bibliothèque libxml2 est utilisée pour analyser les fichiers XML. Le logiciel DRAGON en a besoin pour supporter la mise en service de la topologie spécifique des applications XML.

3.3.2.6 zlib-1.2.3

Zlib est utilisé pour compresser et décompresser des données qui figurent dans certains protocoles de routage et de signalisation.

3.3.3 Installation de Dragon VLSR

Pour installer les logiciels DRAGON VLSR, KOM-RSVP (autre implémentation de RSVP) et GNU ZEBRA, il faut se référer à l'annexe A.1.
Après avoir fait ces installations, parcourons par la suite les étapes de configuration des commutateurs.

3.4 Etapes de configuration du commutateur

Cette section présente les étapes à parcourir pour configurer les commutateurs (qui prennent en charge Net-SNMP). Ces étapes ont aussi été utilisées dans notre plan de contrôle. Pour cette configuration, nous devons contrôler les éléments ci-dessous :

- Activation du serveur SNMP

Le serveur SNMP doit être activé sur le commutateur contrôlé par le VLSR. Pour cela, il faut ajouter une communauté (option de commande du commutateur) nommée « dragon » sur le serveur SNMP. La communauté devrait avoir deux privilèges (lecture et écriture).

- Affectation et administration d'adresses IP

Il faut attribuer une adresse IP à l'interface du commutateur reliée au VLSR. Il est important de conserver cette adresse, puisque nous l'ajouterons dans le fichier de configuration des commutateurs par la suite.

- Création des VLANs vides

Pour les commutateurs qui utilisent la méthode de contrôle SNMP, mais ne supportent pas la création et la suppression dynamique des VLANs, nous devons créer des VLANs vides portant des balises qui seront éventuellement utilisées dans les mises en service futures.

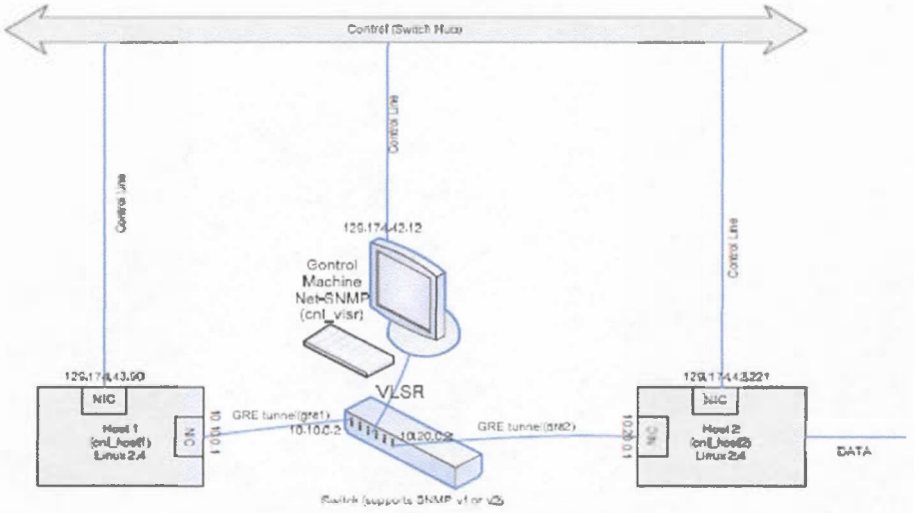
Après la présentation des étapes de configuration, voici la configuration des éléments de DRAGON.

3.5 Configuration des éléments de DRAGON

Cette partie présente toutes les configurations des éléments de DRAGON (nécessaires à notre plan de contrôle) qui seront également effectuées dans notre plan de contrôle à l'instar des tunnels GRE, de DRAGON OSPF-TE, DRAGON RSVP-TE, DRAGON daemon et ZEBRA.

En effet, l'exemple de configuration établit les LSPs (chemins commutés par étiquettes) entre les hôtes exécutant GMPLS via un hôte spécial sans compatibilité GMPLS, fournissant des services de base GMPLS de bout en bout. Cet hôte spécial est désigné comme VLSR, routeur virtuel à commutation par étiquettes. Ici, le logiciel DRAGON est installé sur trois machines où l'une d'elles sera la machine de contrôle (VLSR) qui prendra en charge le commutateur Ethernet.

La configuration qui sera faite plus loin dans cette section est celle de l'architecture réseau suivante :



Source : ((USC) et al., 2008 7)

Figure 3.2 Prototype de configuration du VLSR

Il s'agit d'une topologie simple composée de deux systèmes d'extrémités (hôte 1 et hôte 2) connectés à un commutateur Ethernet lié au VLSR. Le logiciel DRAGON s'exécute sur ces hôtes. Un Switch hub permet de relier le VLSR aux hôtes. Le VLSR contrôle le Switch Ethernet (qui prend en charge SNMP v1 ou v2) et la rend compatible avec la commutation par étiquettes, alors que ce genre de commutateur est par défaut compatible avec la commutation par paquets. Ceci est rendu possible avec l'aide de Net-SNMP et du logiciel DRAGON installé sur le VLSR. Pendant ce temps, les tunnels GRE (*Generic Routing Encapsulation*) sont mis en place entre les hôtes et le VLSR et vice versa. De cette manière, les plans de données et de contrôle sont séparés. Les LSPs peuvent ensuite être mis en place entre les deux hôtes par l'intermédiaire du VLSR.

3.5.1 Configuration des tunnels GRE sur les hôtes

L'encapsulation à routage générique (GRE) prend des paquets ou des trames d'un système réseau et les place à l'intérieur d'une trame d'un autre système réseau. Cette méthode est parfois appelée *tunneling*, et fournit un moyen d'encapsuler les paquets à l'intérieur d'un protocole routable via des interfaces virtuelles. Le tunnel permet à n'importe quel protocole

du plan de contrôle d'être transmis d'un réseau virtuel vers un réseau physique qui s'exécute sur un autre protocole.

Dans l'exemple représenté à la figure 3.2, nous avons besoin de mettre en place deux tunnels GRE. Le premier appelé gre1, de l'hôte 1 vers la machine de contrôle VLSR, et le second appelé gre2, de l'hôte 2 vers la machine de contrôle VLSR. Les messages de signalisation et de routage peuvent ensuite circuler dans les tunnels GRE de l'hôte 1 à l'hôte 2.

Nous commençons la configuration en chargeant d'abord le module de tunnel GRE dans le noyau Linux à l'aide de la commande « ***# /sbin/modprobe ip_gre*** ».

Pour expliquer la configuration du tunnel GRE entre l'hôte 1 et la machine de contrôle, nous avons considéré les adresses IP suivantes :

Host 1 (cnl_host1):
Adresse réseau: 129.174.43.90
Masque de sous-réseau: 255.255.255.0
Adresse locale de l'interface gre1: 10.10.0.1
Host 2 (cnl_host2):
Adresse réseau: 129.174.42.221
Masque de sous-réseau: 255.255.255.0
Adresse locale de l'interface gre2: 10.20.0.1
Machine de contrôle (cnl_vlsr):
Adresse réseau: 129.174.42.12
Masque de sous-réseau: 255.255.255.0
Adresse locale de l'interface gre1: 10.10.0.2
Adresse locale de l'interface gre2:10.20.0.2

Pour configurer les tunnels GRE sur les hôtes 1, 2 et sur le VLSR, il faut se référer à l'annexe A.2.

3.5.2 Configuration de DRAGON OSPF-TE, DRAGON RSVP-TE, DRAGON daemon et ZEBRA

L'exemple des configurations présentées sont celles du cas illustré à la figure 3.2, c'est-à-dire pour les équipements cnl_host1 (hôte 1), cnl_host2 (hôte 2) et cnl_vlsr (VLSR). Le contenu des fichiers de configuration et les processus d'installation sont présentés à l'annexe A.3.

Après avoir configuré les éléments de DRAGON, nous pouvons passer à l'étude des commandes de mise en service du chemin commuté par étiquettes (LSP).

3.6 Commandes de mise en service du LSP via CLI

Cette section présente les commandes qui seront automatisées dans le code de notre plan de contrôle OpenFlow-GMPLS pour mettre en marche les LSP.

En effet, la mise en service du LSP permet d'établir un chemin à commutation par étiquettes entre deux ordinateurs (source et destination).

Pour cela, il faut taper les commandes ci-dessous :

1. $ telnet 129.74.43.90 2611 *(se connecte sur la machine d'adresse IP 129.74.43.90 via le port 2611)*
2. Afficher les modules et les configurer un par un
   a. *show module* *(affiche les modules que nous avons)*
   b. *configure [MODULE_NAME]* *(configure le module)*
3. Créer un LSP sur l'ordinateur source
   a. *edit lsp [LSP_NAME]* *(donne un nom au lsp)*
   b. *set ip_src A.B.C.D port X lsp-id XX ip_dest A.B.C.D port Y tunnel-id YY (définit l'adresse IP et le numéro de port source et de destination)*
   c. *set bandwidth gige swap lsc encoding Ethernet gpid Ethernet (définit la valeur de la capacité de commutation à lsc (Label Switch Capability)}*
   d. *exit*
4. Initier le chemin d'accès de l'expéditeur
   a. *commit lsp [LSP_NAME]* *(valide et active le LSP)*
5. Vérifier l'état LSP
   a. *show lsp [LSP_NAME]* *(montre le statut du lsp)*

   Le statut LSP est comme suit:

   Lsp status = In service: *le chemin d'accès a été établi*
   = Commit: *en attente des réponses du narb*
   = Edit: *n'a pas été encore validé*
   = Listening: *écoute les prochaines demandes de chemin*
   = Delete: *en attente de confirmation de suppression du narb*

Après avoir effectué l'étude des composants et logiciels de GMPLS via le projet DRAGON, quelle conclusion pouvons-nous en tirer ?

3.7 Conclusion

Le guide de mise en œuvre de DRAGON vise à fournir les informations nécessaires aux utilisateurs et aux développeurs pour installer et mettre en place un réseau VLSR. Ce guide nous a permis de prendre en main les composants du projet DRAGON (GMPLS), via l'identification des composants du plan de contrôle DRAGON, l'installation des logiciels nécessaires, la configuration des tunnels GRE et des éléments de DRAGON OSPF-TE,

CHAPITRE IV

ÉTUDE DE OPENFLOW

Pour cette étude, nous nous sommes familiarisés avec l'environnement OpenFlow via le tutorial de déploiement de OpenFlow de la *OF Foundation.* Le déploiement dans notre architecture réseau des composants de OpenFlow est un préalable au fonctionnement de l'agent OpenFlow-GMPLS que nous avons implémenté.

Il était aussi important pour nous d'apprivoiser ces composants et les configurations nécessaires pour être capable de les intégrer à notre architecture réseau et les adapter à nos besoins.

4.1 Introduction

OpenFlow est une interface ouverte pour contrôler à distance les tables de transfert dans les commutateurs, les routeurs et les points d'accès réseaux. Avec la version actuelle de OpenFlow, les chercheurs peuvent construire des réseaux avec de nouvelles propriétés de haut niveau. Par exemple, OpenFlow facilite la gestion des réseaux sécurisés, des réseaux sans fil avec des transferts fluides, des réseaux des centres de données évolutifs, de la mobilité de l'hôte, des réseaux plus efficients en termes d'énergie et des nouveaux réseaux étendus, pour ne nommer que ces quelques fonctionnalités.

Dans ce chapitre, nous allons dans un premier temps énumérer les composants nécessaires à l'environnement virtuel OpenFlow. Dans un second temps, nous allons installer ces composants et les configurer pour la création de l'environnement virtuel. Dans un troisième temps, nous allons effectuer la mise en service de réseau virtuel. Puisque le tutorial de déploiement de OpenFlow de la *OF Foundation* se déroule en environnement virtuel, nous

allons étudier cet environnement. Pour cela, nous pouvons nous poser la question suivante : quels sont les éléments qui constituent cet environnement ?

4.2 Composants/logiciels de l'environnement virtuel de OpenFlow

Cette portion permet de nommer les composants ou logiciels nécessaires au déploiement de l'environnement virtuel OpenFlow.

Les composants dont l'environnement virtuel OpenFlow a besoin sont : un système d'exploitation, un logiciel de virtualisation, un serveur X et un terminal. Ces composants se révéleront utiles pour faire tourner le hub prévu dans un commutateur d'apprentissage. Nous nommerons ces logiciels et donnerons leurs rôles plus bas dans la sous-section prérequis avant l'installation.

Après avoir nommé les composants de l'environnement virtuel, nous passons à leurs installations.

4.3 Installation des logiciels/composants

Nous présentons dans cette partie l'installation de l'environnement virtuel OpenFlow et les conditions nécessaires pour cela.

Dans le cadre du déploiement virtuel de OpenFlow, il nous a été fourni l'image (mininet-2.0.0-113012-amd64-ovf) préconfigurée de la machine virtuelle, qui comprend les logiciels et les composants dont OpenFlow a besoin.

4.3.1 Préparation avant l'installation

Cette préparation se situe au niveau du matériel et du système d'exploitation.

a) Le matériel

Nous avons besoin d'un ordinateur avec au moins 1 Go de RAM (2 Go et plus est préféré) et au moins 5 Go d'espace libre sur le disque dur (plus préféré). Un processeur plus rapide peut accélérer le temps de démarrage de l'ordinateur virtuel, et un plus grand écran peut aider à gérer plusieurs fenêtres.

b) Système d'exploitation

Le système d'exploitation que nous utiliserons pour abriter l'environnement virtuel OpenFlow est Windows 7, car il est déjà installé sur l'ordinateur requis à cet effet.

4.3.2 Prérequis avant l'installation

Plusieurs logiciels sont requis avant l'installation de OpenFlow. Nous avons besoin d'un logiciel de virtualisation qui est Virtual Box. Il sera utilisé pour faire tourner le système d'exploitation Linux au-dessus de Windows 7.

Nous avons aussi besoin d'un serveur X, qui est le logiciel Xming. Léger et rapide, il permet de rediriger l'affichage des applications graphiques qui tournent sur des systèmes distants Unix ou Linux vers l'écran du PC qui fait tourner le serveur Xming.

Nous avons enfin besoin d'un terminal virtuel qui est puTTY. Compatible SSH (protocole de communication sécurisé permettant l'accès distant à des machines Linux ou Unix), il permet de se connecter sur un ordinateur distant.

4.3.3 Installation de l'environnement virtuel

Pour installer l'environnement virtuel OpenFlow, il faut importer dans Virtual Box l'image préconfigurée de la machine virtuelle (mininet-2.0.0-113012-amd64-ovf), et installer le server Xming. Une fois l'image importée et la machine virtuelle démarrée, nous devons fournir le nom d'utilisateur « mininet » et le mot de passe « mininet » afin d'obtenir l'environnement Mininet ci-dessous :

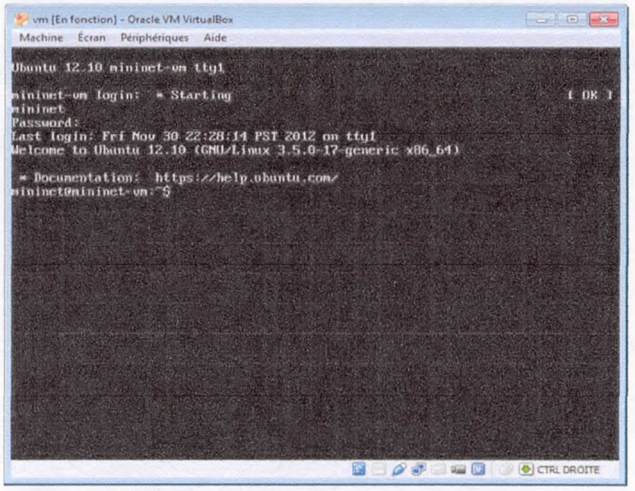


Figure 4.1 Environnement virtuel Mininet

C'est environnement Mininet est la plate-forme d'émulation du réseau qui permet de créer un réseau virtuel OpenFlow (un contrôleur, des commutateurs, des hôtes et des liens). Cet environnement possède plusieurs utilitaires qui sont :

- Dpctl : c'est l'utilitaire de ligne de commande qui permet l'envoie rapide des messages OpenFlow entre un contrôleur OpenFlow et des commutateurs OpenFlow. Il est utile pour la visualisation des ports de commutation et pour l'insertion manuelle des entrées de flux. Il active la visibilité et le contrôle dans la table de flux d'un commutateur OpenFlow. Il est également utile pour le débogage, via la visualisation des compteurs de flux et l'état de flux. La plupart des commutateurs OpenFlow peuvent démarrer avec un port d'écoute passif (dans notre configuration actuelle c'est le numéro de port 6634), à partir duquel nous pouvons interroger le commutateur, sans avoir à ajouter du code de débogage au contrôleur.
- Wireshark : c'est l'utilitaire graphique qui permet de visualiser des paquets. La distribution de référence de OpenFlow comprend un dissecteur Wireshark qui analyse les messages OpenFlow envoyés vers le port OpenFlow par défaut (port 6633) d'une manière pratique et lisible.
- Iperf : c'est l'utilitaire général de ligne de commande qui permet de tester la vitesse d'une connexion TCP.
- Cbench : c'est l'utilitaire qui permet de tester la vitesse de configuration de flux des contrôleurs OpenFlow.

Une fois les logiciels et composants nécessaires à OpenFlow installés via l'image préconfigurée de la machine virtuelle (mininet-2.0.0-113012-amd64-ovf), nous passons à la création d'un réseau virtuel OpenFlow.

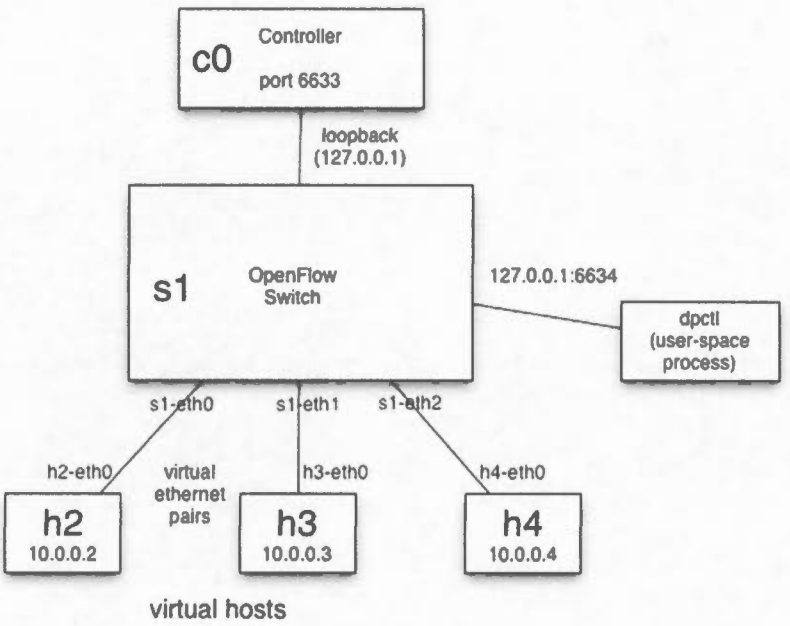
4.4 Création d'un environnement virtuel OpenFlow

Cette section permet de créer un réseau virtuel OpenFlow préconfiguré, et d'obtenir les configurations nécessaires à la portion OpenFlow de notre plan de contrôle OpenFlow-GMPLS.

Toutes les commandes à exécuter sont effectuées via le « *X forwarding* » (option de connexion utilisée par l'émulateur puTTY), où les programmes graphiques d'affichage se feront via un « X server » tournant sur le système d'exploitation hôte. Pour démarrer le « *X forwarding* », nous devons d'abord trouver l'adresse IP des clients. Pour ce déploiement, nous

devons nous assurer que notre VM (Machine Virtuelle) dispose de deux interfaces réseaux : une interface NAT que peut utiliser la VM pour accéder à Internet, et une interface de type hôte uniquement pour permettre à la VM de communiquer avec la machine hôte. Notre interface NAT est eth1 avec l'adresse IP 10.X.X.X, et notre interface de type hôte est eth0 avec l'adresse IP 192.168.X.X. Nous devons nous connecter via l'interface hôte uniquement (eth0). Les deux interfaces doivent être configurées à l'aide de DHCP.

Le réseau virtuel OpenFlow qui sera créé sur la VM se composera d'un contrôleur OpenFlow (cO), d'un Switch OpenFlow (s1), et de trois hôtes (h2, h3 et h4). Ce réseau dispose aussi d'un espace de travail utilisateur (dpctl) où des commandes peuvent être exécutées pour interroger le Switch OpenFlow. Le schéma ci-dessous donne un aperçu du réseau virtuel.



Source (OF-Foundation, 2011)

Figure 4.2 Architecture du réseau virtuel OpenFlow avec un Switch (s1)

Dans ce réseau, le contrôleur et le Switch OpenFlow agissent comme un Switch Ethernet d'apprentissage. Pour créer ce réseau dans la machine virtuelle, il faut :

Premièrement démarrer la machine virtuelle, se connecter et taper la commande *« ifconfig –a »* qui permet d'afficher la configuration réseau sur un système d'exploitation Linux. La fenêtre ci-dessous montre le résultat de cette commande.

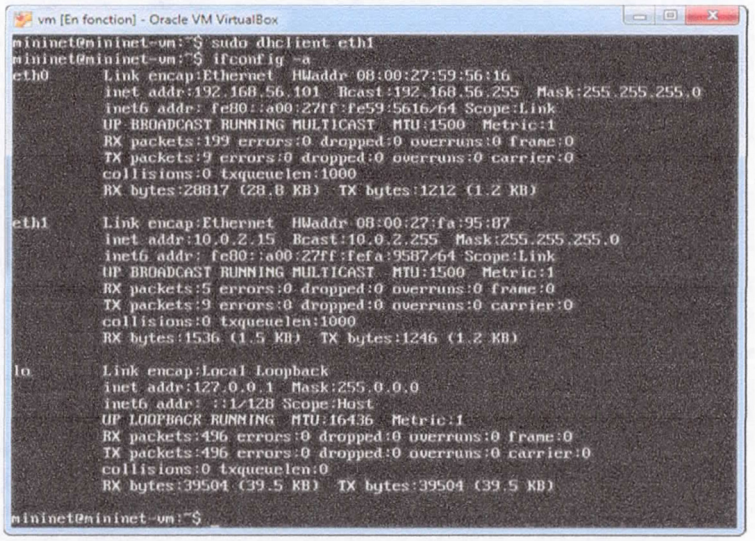


Figure 4.3 Résultat de la commande « ifconfig –a »

Trois interfaces sont affichées (eth0, eth1, lo); deux de ces interfaces (eth0 et eth1) ont une adresse IP attribuée. Cette adresse se trouve à la deuxième ligne de chaque interface après l'étiquette « inet addr :».

Deuxièmement lancer puTTY et établir une connexion SSH à la machine virtuelle via l'interface eth0 (IP : *192.168.56.101*) en activant *X11*. Se connecter et créer le réseau virtuel à l'aide de la commande « **sudo mn --topo single, 3 --mac --switch ovsk --controller remote** ».

Cette commande exécute les actions ci-dessous :

- créer 3 hôtes virtuels, chacun avec une adresse IP distincte.
- créer un commutateur OpenFlow avec 3 ports.
- connecter chaque hôte virtuel au commutateur avec un câble Ethernet virtuel.
- définir l'adresse MAC de chaque hôte égal à son adresse IP.
- configurer le commutateur OpenFlow pour se connecter à un contrôleur distant.

La fenêtre ci-dessous donne le résultat de cette commande.

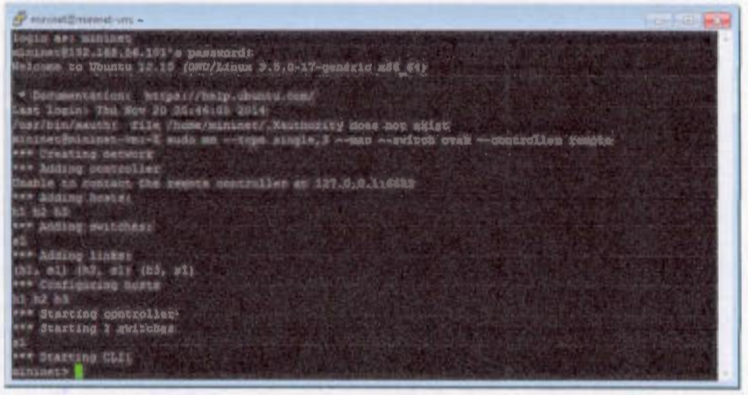


Figure 4.4 Création du réseau virtuel Mininet

Après la création de l'environnement pour le déploiement de OpenFlow, nous le mettons en service.

4.5 Mise en service du réseau virtuel OpenFlow

Cette section permet de mettre en service le réseau virtuel OpenFlow via le démarrage du contrôleur OF qui gèrera tous les équipements de l'environnement réseau.

Pour mettre en service le réseau virtuel, nous devons dans un premier temps tester la connexion entre deux hôtes (h1 et h2). Pour cela, il faut exécuter la commande « h1 ping -c3 h2 ». La figure ci-dessous donne le résultat de ce test.

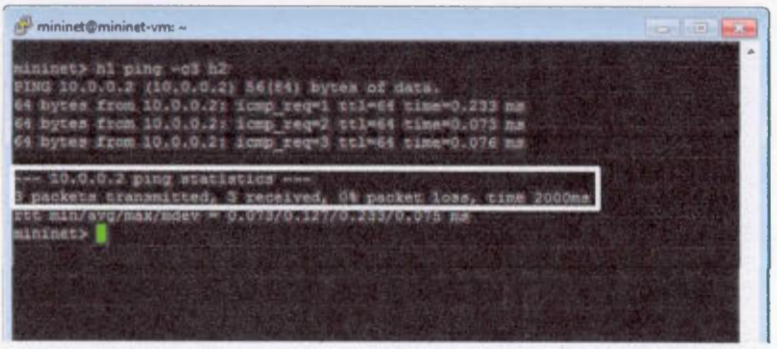


Figure 4.5 Test du réseau virtuel

Nous observons dans le petit rectangle blanc de la figure 4.5, 3 paquets transmis, 3 reçus, 0% de paquet perdu. Ce qui montre que le test a réussi.

Dans un second temps, nous devons faire un test de transmission des paquets OpenFlow. Pour cela, nous devons lancer le serveur Xming en passant par les menus « Démarrer - Tous les programmes - Xming » sur le bureau du PC hôte.

Dans un troisième temps, nous devons lancer Wireshark à l'aide de la commande « sudo wireshark & » Via puTTY. Par la suite, nous devons démarrer le contrôleur OF à l'aide de la commande « controller ptcp: » dans le terminal SSH de puTTY. La capture des échanges entre le contrôleur OF et le Switch OF est représentée dans la figure ci-dessous :

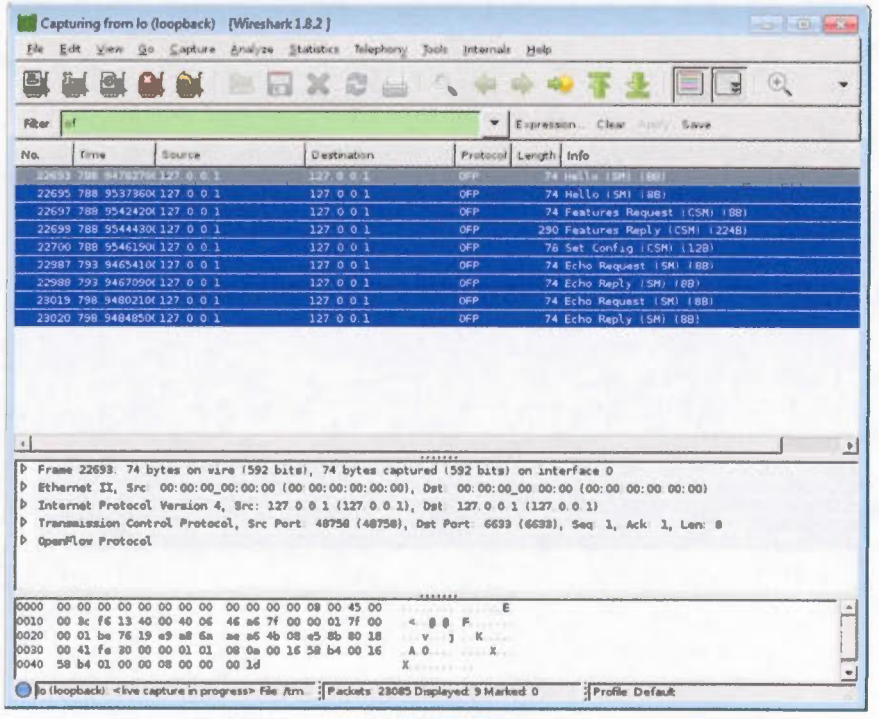


Figure 4.6 Échange des messages entre le contrôleur OpenFlow et le Switch OpenFlow

Les différents types de messages OpenFlow sont représentés dans les tableaux ci-dessous :

Tableau 4.1 Types de messages OpenFlow

Message	Type	Description
Hello	Contrôleur → Switch	Suivant l'échange TCP, le contrôleur envoie son numéro de version au Switch
Hello	Switch → Contrôleur	Le Switch répond avec son numéro de version supporté
Features Request	Contrôleur → Switch	Le contrôleur demande les ports disponibles
Set Config	Contrôleur → Switch	Dans ce cas, le contrôleur demande au Switch d'envoyer les flux d'expiration
Features Reply	Contrôleur → Switch	Le Switch répond avec une liste de ports, les vitesses de ports, et les tables et actions supportées
Port Status	Switch → Contrôleur	Permet au Switch d'informer le contrôleur sur les vitesses et les connectivités.

Tableau 4.2 Nouveaux types de messages OpenFlow

Message	Type	Description
Packet-In	Switch → Contrôleur (port 6633)	Un paquet est reçu et ne correspond à aucune entrée dans la table de flux du Switch. Ce qui pousse le Switch à expédier le paquet au contrôleur
Packet-Out	Contrôleur → Switch	Le contrôleur envoie un paquet à un ou plusieurs ports du Switch.
Flow-Mod	Contrôleur → Switch	Ordonne au Switch d'ajouter un flux particulier à sa table de flux.
Flow-Expired	Switch → Contrôleur	Donne l'information sur un flux expiré après une période d'inactivité

Après s'être approprié de OpenFlow en nommant et en installant les composants de l'environnement virtuel, en créant et en mettant en service cet environnement, quelle conclusion pouvons-nous en tirer ?

4.6 Conclusion

L'étude du déploiement de OpenFlow a été pour nous l'occasion d'acquérir une expérience pratique avec la plate-forme et les outils de débogage très utiles pour le

développement des plans de contrôle réseau utilisant OpenFlow. Au cours de cette étude, nous nous sommes appropriés des commandes, des composants logiciels et matériels, de l'installation et de la mise en service de OpenFlow. Cela a été une étape inéluctable et cruciale dans la suite de notre recherche qui concerne à la fois OpenFlow et GMPLS.

Après la prise en main de GMPLS grâce au projet DRAGON, et de OpenFlow à travers son déploiement, il est temps pour nous de passer à notre plan de contrôle (contrôleur) OpenFlow-GMPLS. Pour ce faire, nous devons d'abord concevoir une architecture SDN qui nous permettra d'implémenter notre plan de contrôle. Le chapitre suivant porte sur la conception de notre architecture réseau OpenFlow-GMPLS.

CHAPITRE V

CONCEPTION DE L'ARCHITECTURE OPENFLOW-GMPLS

La conception d'une architecture réseau a pour objectif essentiel de créer un schéma de réseau capable de répondre à des contraintes fonctionnelles et organisationnelles. Pour cela, nous nous sommes inspirés des architectures OpenFlow et GMPLS qui sont présentées dans ce chapitre.

5.1 Introduction

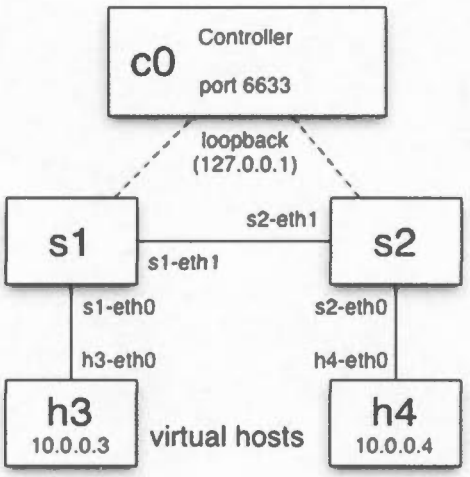
Pour implémenter un plan de contrôle OpenFlow-GMPLS, il faut d'abord concevoir une architecture réseau intégrant d'une part une architecture OpenFlow, et d'autre part une architecture GMPLS. Afin que cette architecture puisse répondre au réseau du futur, il faut en plus ajouter la contrainte majeure de la séparation du plan de contrôle de celui des données.
L'évolutivité, la mobilité et la sécurité des réseaux sont aujourd'hui les composantes centrales des architectures réseaux. La technologie SDN intègre toutes ces composantes. Dans ce chapitre, nous allons présenter dans les détails la démarche qui nous a permis de faire la conception de l'architecture OpenFlow-GMPLS.

5.2 Conception de l'architecture OpenFlow-GMPLS

Cette section permet à partir des modifications et des extensions des architectures OpenFlow d'une part et GMPLS d'autre part, d'obtenir une architecture OpenFlow-GMPLS qui convienne à notre solution.
Pour concevoir cette architecture, nous sommes partis de l'architecture du réseau virtuel OpenFlow avec 2 Switchs et du prototype d'architecture GMPLS de DRAGON se trouvant dans ce chapitre.

5.2.1 Architecture OpenFlow

L'architecture du réseau virtuel OpenFlow proposé par la *OF Fondation* est représentée comme ci-dessous :



Source (OF-Foundation, 2011)

Figure 5.1 Architecture du réseau virtuel OpenFlow avec 2 Switchs (s1 et s2)

Cette architecture possède un contrôleur OpenFlow (c0), deux commutateurs ou Switch OpenFlow (s1 et s2) et deux hôtes (h3 et h4). Le contrôleur OpenFlow est relié aux deux commutateurs qui sont à leurs tours reliés entre eux, et chaque commutateur est relié à un hôte.

Pour obtenir la portion OpenFlow de notre architecture OpenFlow-GMPLS, nous avons fait une transposition et une extension de l'architecture ci-dessus en y intégrant beaucoup plus de commutateurs et d'hôtes.

Nous avons alors obtenu l'architecture de la page suivante :

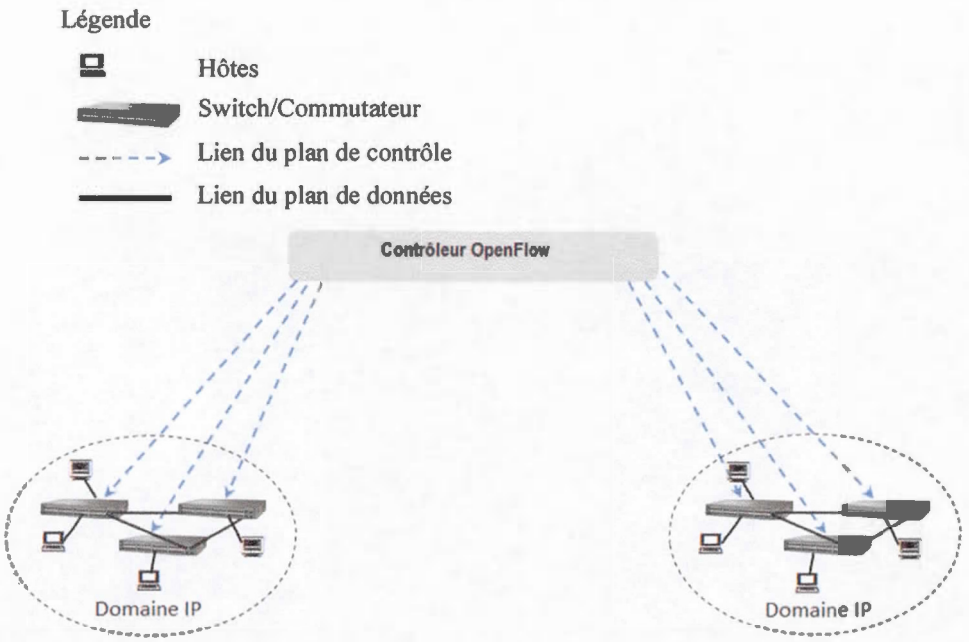


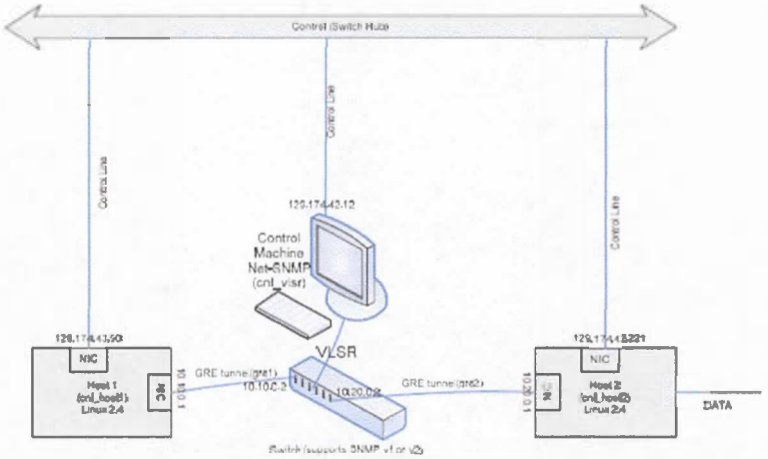
Figure 5.2 Portion OpenFlow de l'architecture OpenFlow-GMPLS

Dans cette architecture, nous avons deux domaines IP reliés par des liens Ethernet (liens du plan de contrôle) au contrôleur OpenFlow. Chaque domaine IP possède trois commutateurs électriques et quatre hôtes reliés entre eux par des liens Ethernet (liens du plan de données).

5.2.2 Architecture GMPLS

Le projet DRAGON (*Dynamic Resource Allocation in GMPLS Optical Networks*) nous propose un prototype d'architecture.

Cette architecture possède un ordinateur de contrôle (Net-SNMP) appelé VLSR, un commutateur électrique supportant SNMP version 1 ou 2 et deux hôtes (host 1 et host 2). Tous ces équipements réseaux sont reliés entre eux via un Switch hub. Cette architecture est représentée comme suit :



Source : ((USC) et al., 2008 7)

Figure 5.3 Prototype d'architecture GMPLS de DRAGON

En faisant une extension de cette architecture, c'est-à-dire en utilisant deux ordinateurs de contrôle appelés VLSR plutôt qu'un, deux ordinateurs clients appelés CSA (*Client System Agent*), et en remplaçant le commutateur électrique par deux commutateurs optiques, nous arrivons à l'architecture ci-dessous :

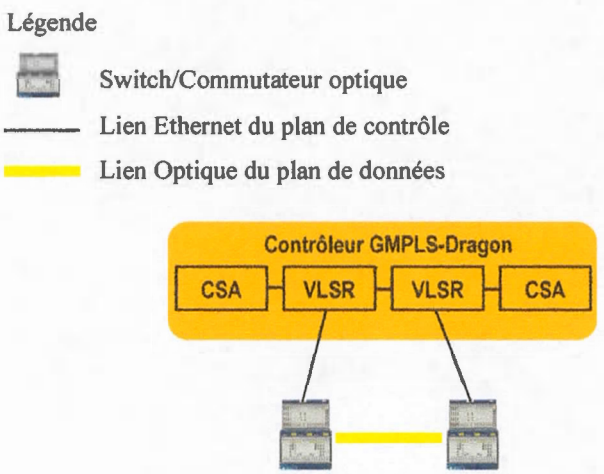


Figure 5.4 Portion GMPLS de l'architecture OpenFlow-GMPLS

Cette architecture possède deux ordinateurs de contrôle VLSR, un contrôleur GMPLS-Dragon, deux hôtes clients CSA et deux commutateurs optiques. Les VLSR et les CSA (équipements du plan de contrôle) sont reliés par des liens Ethernet, et les commutateurs optiques (équipements du plan de données) par un lien optique. Les plans de contrôle et de données sont reliés entre eux par deux liens Ethernet. Chaque VLSR a pour rôle d'assurer la gestion et le contrôle de chaque commutateur optique.

Après avoir fait des extensions et des modifications des architectures OpenFlow d'une part et GMPLS-Dragon d'autre part, nous passons à la conception de l'architecture OpenFlow-GMPLS.

5.3 Architecture OpenFlow-GMPLS

Cette section permet d'intégrer les architectures OpenFlow (figure 5.2) et GMPLS (figure 5.4) dans une seule architecture réseau respectant le principe de la séparation du plan de données de celui de contrôle, afin d'obtenir notre architecture OpenFlow-GMPLS.

Pour cela, nous sommes partis de l'architecture GMPLS en ajoutant un commutateur optique, et nous avons octroyé les fonctions de commutation OpenFlow aux deux postes clients (CSA). Nous avons lié chaque domaine IP à un routeur optique via un transpondeur. Nous avons enfin fusionné les contrôleurs OpenFlow et GMPLS-Dragon avec une passerelle OpenFlow-GMPLS.

En combinant de façon conceptuelle les architectures OpenFlow et GMPLS-Dragon, nous obtenons le schéma suivant incluant une délimitation du plan de contrôle de celui des données.

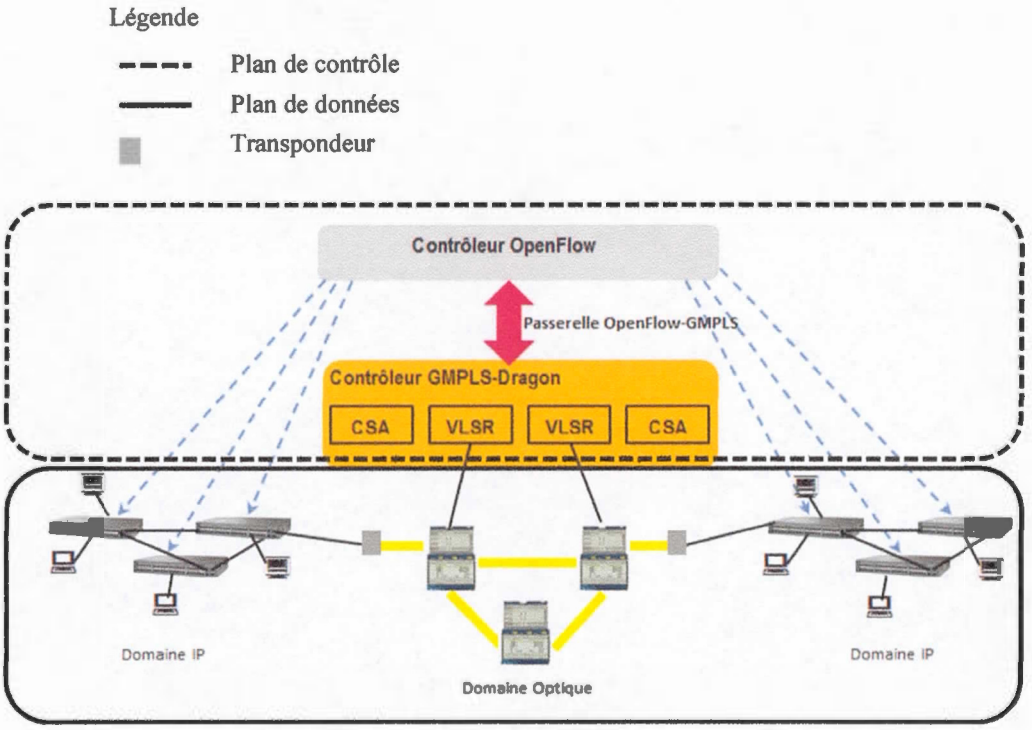


Figure 5.5 Architecture OpenFlow-GMPLS

Cette architecture possède un plan de contrôle et un plan de données. Le plan de contrôle comporte un contrôleur OpenFlow et un contrôleur GMPLS-Dragon (possédant quatre ordinateurs). Ces deux contrôleurs sont reliés par un agent ou passerelle OpenFlow-GMPLS. Le plan de données comporte deux domaines IP à commutation par paquets et un domaine optique. Chaque domaine IP possède des commutateurs électriques et des ordinateurs hôtes, et le domaine optique possède trois commutateurs optiques. Deux transpondeurs permettant de convertir le signal électrique en signal optique et vice-versa relient le domaine optique des domaines IP.

Après la conception d'une architecture OpenFlow-GMPLS adéquate à notre solution, quelle conclusion pouvons-nous en tirer ?

5.4 Conclusion

Après s'être inspiré des architectures des plans de contrôle OpenFlow d'une part, et GMPLS-Dragon d'autre part, nous avons bâti une architecture OpenFlow-GMPLS.

En effet, nous avons transposé et étendu l'architecture OpenFlow en y intégrant beaucoup plus de commutateurs et d'hôtes. Ensuite, nous avons aussi étendu l'architecture GMPLS-Dragon en utilisant deux ordinateurs de contrôle appelés VLSR, deux ordinateurs clients appelés CSA, et en remplaçant le commutateur électrique par des commutateurs optiques. Enfin, nous avons intégré ces deux architectures dans une seule, tout en respectant le principe de la séparation du plan de données de celui de contrôle pour obtenir notre architecture OpenFlow-GMPLS. Cette dernière a été installée physiquement dans notre laboratoire réseau.

La configuration préalable de l'architecture OpenFlow-GMPLS, les tâches à accomplir et l'implémentation proprement dite du plan de contrôle OpenFlow-GMPLS seront décrites dans le chapitre suivant.

CHAPITRE VI

IMPLÉMENTATION DU PLAN DE CONTRÔLE OPENFLOW-GMPLS

L'implémentation d'un plan de contrôle OpenFlow-GMPLS dans une architecture réseau conçue et installée, consiste en une modification et/ou extension des protocoles OpenFlow et GMPLS, pour centraliser de façon logique l'intelligence et l'état du réseau à partir des applications sur un super-ordinateur.

6.1 Introduction

Tout environnement réseau a besoin de configurations préalables pour fonctionner, parce que chaque équipement réseau doit connaître les équipements voisins (ceux avec qui il est directement connecté) et leurs modes de fonctionnement. La configuration de notre plan de contrôle OpenFlow-GMPLS passe par la configuration de chaque équipement présent dans l'architecture réseau.

Dans ce chapitre, nous allons configurer tous les équipements réseaux, tester et évaluer le plan de contrôle implémenté et décrire son comportement en cas de rupture de lien réseau. Nous abordons alors la configuration de notre environnement réseau.

6.2 Configuration de l'environnement OpenFlow-GMPLS

Dans cette section, nous allons détailler toutes les configurations qui ont été effectuées dans l'architecture de l'environnement réel de test OpenFlow-GMPLS. Nous allons présenter les configurations liées au contrôleur OpenFlow-GMPLS, aux commutateurs OpenFlow, aux interfaces GRE et aux VLSR.

Il est à noter que les installations de certains composants et outils de OpenFlow et de GMPLS-Dragon ont été préalablement faites. Il s'agit de l'installation des composants de

GMPLS (section 3.3) et de l'installation des logiciels et composants de OpenFlow (section 4.3).

L'architecture de l'environnement de test OpenFlow-GMPLS (représentant le réseau physique réel) que nous allons configurer est représentée par le schéma ci-dessous :

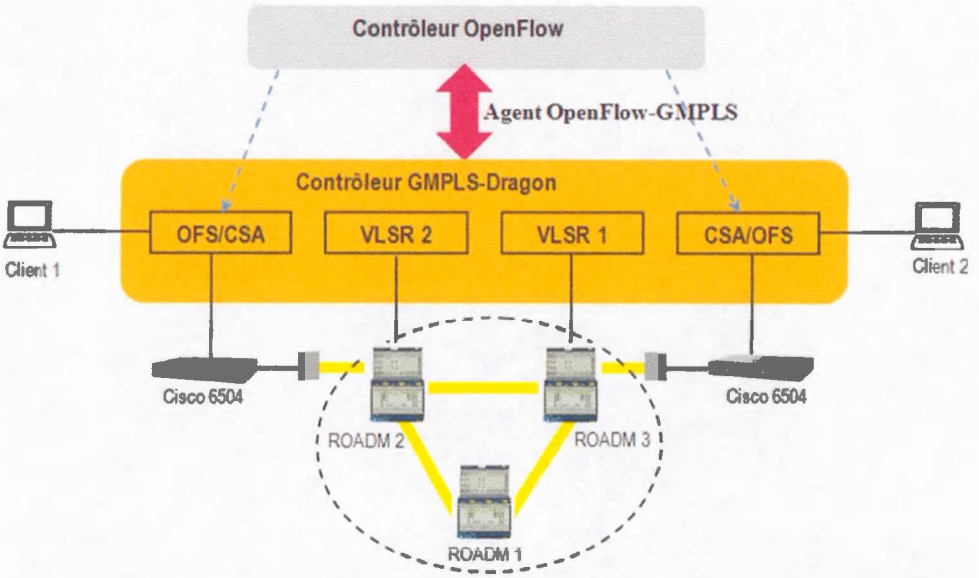


Figure 6.1 Architecture de l'environnement de test du contrôleur OpenFlow-GMPLS

Cet environnement de test, issu de l'architecture OpenFlow-GMPLS de la figure 5.5 a été déployé physiquement dans notre laboratoire informatique. Or, les Switch OpenFlow utilisés dans le tutorial du déploiement de OpenFlow de la *OF Foundation* sont des PC transformés en Switch OpenFlow via l'installation des composants de OpenFlow. Il a fallu réajuster notre architecture pour son déploiement en réel. En effet dans la figure 6.1, les PC nommés OFS/CSA ou CSA/OFS jouent les rôles de CSA pour la portion GMPLS du contrôleur OpenFlow-GMPLS et de Switch OpenFlow pour la portion OpenFlow du contrôleur OpenFlow-GMPLS. De ce fait, chaque domaine IP est donc constitué d'un hôte (Client 1 ou Client 2), d'un Switch Cisco 6504 et d'une partie logique du PC OFS/CSA ou CSA/OFS.

Notre environnement de test est alors composé de deux clients (Client 1 et 2) qui sont reliés aux commutateurs OFS/CSA et CSA/OFS. Chacun d'eux est relié à un commutateur électrique Cisco 6504, connecté au domaine optique via un convertisseur électrique / optique. Le domaine optique est composé de trois commutateurs optiques Cisco (ROADM 1, 2 et 3). Chaque ROADM est contrôlé par un agent SNMP-TL1. Un agent OpenFlow-GMPLS relie le contrôleur OpenFlow au contrôleur GMPLS-Dragon.

6.2.1 Configuration des postes clients

Les postes clients ici sont les ordinateurs d'extrémité à l'architecture, c'est-à-dire les équipements des utilisateurs finaux qui doivent communiquer entre eux. Dans notre cas, ces équipements doivent appartenir à un même sous-réseau. Pour cela ils doivent avoir la même adresse réseau et le même masque de sous-réseau. Dans notre exemple, l'adresse réseau est 172.16.0.0 et le masque 255.255.0.0; donc toutes les adresses de la forme 172.16.X.X appartiennent au même sous réseau de masque 255.255.0.0.

Configuration du client 1 :

- Adresse IP : 172.16.98.101
- Masque de sous réseau : 255.255.0.0 ; équivaut au préfixe « /16 »

Configuration du client 2:

- Adresse IP : 172.16.98.102
- Masque de sous réseau : 255.255.0.0 ; équivaut au préfixe « /16 »

6.2.2 Configuration du contrôleur OpenFlow-GMPLS

Le contrôleur OpenFlow-GMPLS possède deux cartes réseaux Ethernet ; une servant de connexion avec le plan de contrôle et l'autre avec le plan de données.

Configuration de la 1ère carte réseau (connexion avec le plan de contrôle) :

- Adresse IP : 10.10.23.101
- Masque de sous réseau : 255.255.240.0 ; équivaut au préfixe « /20 »

Configuration de la 2ème carte réseau (connexion avec le plan de données)

- Adresse IP : 192.1.0.101
- Masque de sous réseau : 255.255.255.0 ; équivaut au préfixe « /24 »

L'ordinateur jouant le rôle de contrôleur OpenFlow-GMPLS possède deux ordinateurs virtuels pour contrôler les deux commutateurs OpenFlow (OFS/CSA et CSA/OFS). La figure suivante donne une représentation de ce phénomène de virtualisation.

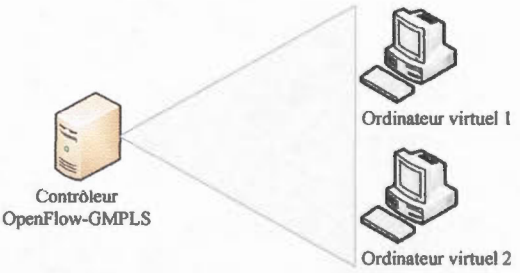


Figure 6.2 Ordinateurs virtuels du contrôleur OpenFlow-GMPLS

L'ordinateur virtuel 1 doit être configuré comme suit :

Configuration de la 1ère carte réseau (connexion avec le plan de contrôle)

- Adresse IP : 10.10.23.100
- Masque de sous réseau : 255.255.240.0 ; équivaut au préfixe « /20 »

Configuration de la 2ème carte réseau (connexion avec le plan de données)

- Adresse IP : 192.1.0.100
- Masque de sous réseau : 255.255.255.0 ; équivaut au préfixe « /24 »

L'ordinateur virtuel 2 doit être configuré comme suit :

Configuration de la 1ère carte réseau (connexion avec le plan de contrôle) :

- Adresse IP : 10.10.23.102
- Masque de sous réseau : 255.255.240.0 ; équivaut au préfixe « /20 »

Configuration de la 2ème carte réseau (connexion avec le plan de données)

- Adresse IP : 192.1.0.102
- Masque de sous réseau : 255.255.255.0 ; équivaut au préfixe « /24 »

6.2.3 Configuration des commutateurs/Switch OpenFlow

Chaque ordinateur converti en commutateur OpenFlow doit avoir trois cartes réseaux afin de jouer le rôle de Switch OpenFlow, et respecter le principe de la séparation du plan de données de celui de contrôle. La première carte réseau sert à relier l'ordinateur (commutateur OpenFlow) à un client, la seconde au contrôleur (plan de contrôle), et la troisième au plan de données. Pour activer la commutation OpenFlow sur chaque commutateur, il faut effectuer les configurations des sous-sections qui suivent.

6.2.3.1 Configuration du Switch (OFS/CSA)

Pour configurer ce Switch afin qu'il agisse comme Switch OpenFlow, il faut ouvrir l'environnement de développement (Eclipse) sur une machine virtuelle du contrôleur OpenFlow-GMPLS, exécuter les classes Java suivantes : SimpleController.java, SimpleONS1SW.java, SimpleONS2SW.java, et SimpleONS3SW.java. Ces classes contiennent du code Java permettant de gérer les commutateurs OpenFlow et les commutateurs optiques. Une partie du code contenu dans ces classes Java provient du logiciel libre OpenFlow de la *OF Foundation*, et l'autre partie provient de l'extension que nous avons effectué dans le cadre de notre recherche. L'appendice A présente uniquement cet ajout, car nous n'avons pas jugé opportun de publier des milliers de lignes de code de la *OF Foundation* dans notre mémoire.

Après l'exécution de ces classes Java, il faut aller à l'invite de commande du système Linux et exécuter les commandes suivantes :

sudo -s *(connexion via la console du système Linux en tant qu'administrateur)*
cd openflow *(entre dans le répertoire nommé « openflow »)*
./udatapath/ofdatapath –detach punix:/var/run/dp0 -d 004E46324301 -i eth1,eth0
./secchan/ofprotocol unix:/var/run/dp0 tcp:10.10.23.100:6633

La ligne 3 permet de faire la correspondance entre l'ID du Switch OF (004E46324301) et les cartes Ethernet Eth1 (où le client est connecté) et Eth0 (connecté au Switch Cisco 6504).

La ligne 4 permet de faire une connexion TCP à la machine virtuelle 1 dont l'adresse IP est 10.10.23.100, via le port 6633.

L'écran suivant donne le résultat de la configuration du Switch OpenFlow. Remarquer à la fin de la dernière ligne le mot « *connected* » qui nous indique que le Switch OpenFlow est connecté au contrôleur OpenFlow-GMPLS via la machine virtuelle d'adresse IP 10.10.23.100 à travers le port 6633.

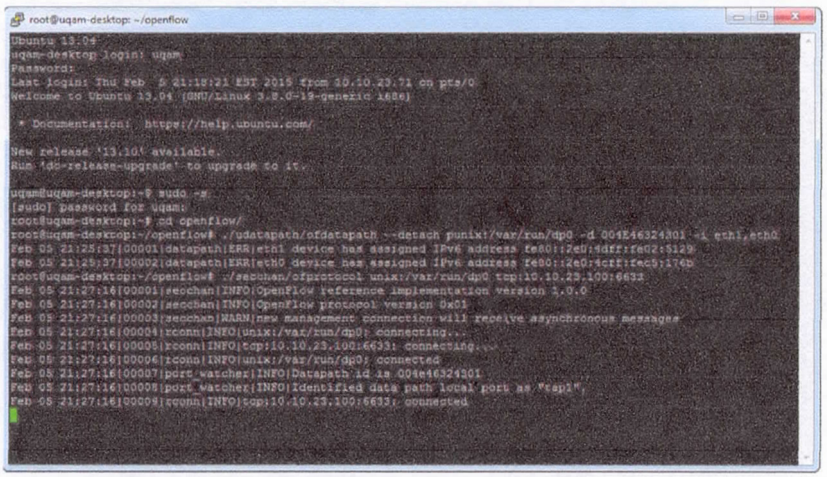


Figure 6.3 Écran de connexion du Switch OpenFlow (OFS/CSA)

6.2.3.2 Configuration du switch (CSA/OFS)

Pour configurer ce Switch afin qu'il agisse aussi comme Switch OpenFlow, il faut suivre les mêmes étapes que précédemment concernant l'exécution des classes citées plus haut. Ensuite, il faut aller à l'invite de commande du système Linux et exécuter les commandes suivantes :

sudo –s *(connexion via la console du système Linux en tant qu'administrateur)*
cd openflow *(entre dans le répertoire nommé « openflow »)*
./udatapath/ofdatapath --detach punix:/var/run/dp0 -d 004E46324302 -i eth1,eth0
./secchan/ofprotocol unix:/var/run/dp0 tcp:10.10.23.102:6633

La ligne 3 permet de faire la correspondance entre l'ID du Switch OF (004E46324302) et les cartes Ethernet Eth1 (où le client est connecté) et Eth0 (connecté au Switch Cisco 6504).
La ligne 4 permet de faire une connexion TCP à la machine virtuelle 2 dont l'adresse IP est 10.10.23.102, via le port 6633.
L'écran suivant donne le résultat de la configuration du Switch OpenFlow. Remarquer à la fin de la dernière ligne le mot « *connected* » qui nous indique que le Switch OpenFlow est connecté au contrôleur OpenFlow-GMPLS via la machine virtuelle d'adresse IP 10.10.23.102 à travers le port 6633.

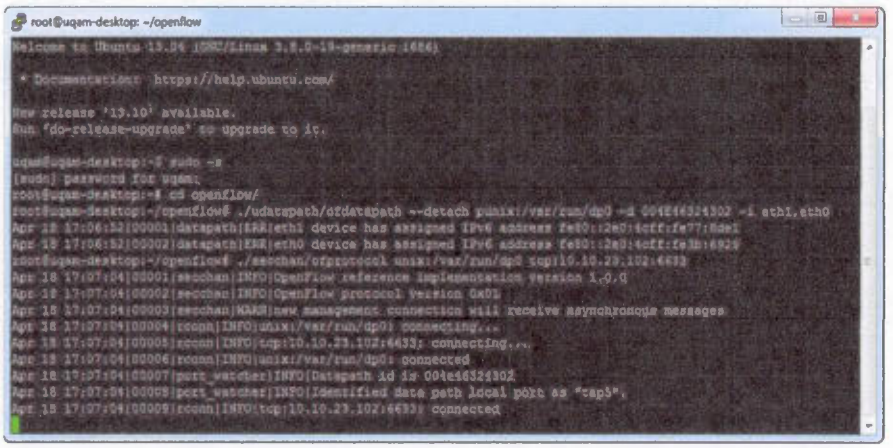


Figure 6.4 Écran de connexion du Switch OpenFlow (CSA/OFS)

6.2.4 Configuration des interfaces GRE

Dans le schéma ci-dessous, les interfaces GRE 1, 2 et 3 sont représentées en violet. Les adresses IP de ces interfaces sont les suivantes :

Gre1 (côté OFS/CSA): 10.10.0.1/30 et Gre1 (côté VLSR2): 10.10.0.2/30

Gre2 (côté VLSR2): 10.20.0.2/30 et Gre2 (côté VLSR1): 10.20.0.1/30

Gre3 (côté VLSR1): 10.30.0.2/30 et Gre3 (côté CSA/OFS): 10.30.0.1/30

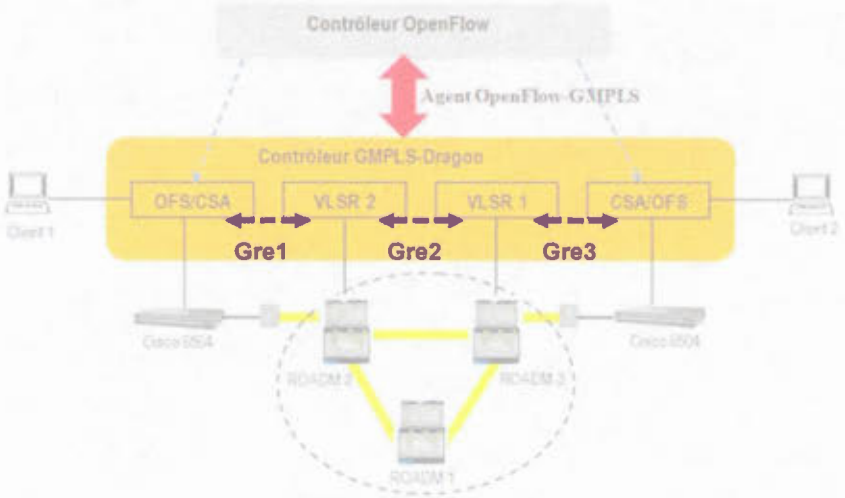


Figure 6.5 Représentation des interfaces GRE dans l'architecture OpenFlow-GMPLS

Les interfaces GRE devraient être configurées sur tous les appareils qui sont dans le domaine GMPLS-Dragon. Pour cela, il faudrait utiliser les fichiers greScriptHost1, greScriptVLSR2, greScriptVLSR1, greScriptHost2 qui contiennent les commandes de configuration de ces interfaces. Le contenu de ces fichiers se trouve dans l'appendice B.

Pour configurer l'interface Gre1 sur OFS/CSA, nous avons effectué les opérations suivantes :

Se connecter en mode racine : ***sudo – s*** et entrer le mot de passe

Entrer dans le répertoire « Desktop » à l'aide de la commande « ***cd Desktop*** »

Exécuter le fichier « greScriptHost1 » à l'aide de la commande « ***./greScriptHost1*** »

L'extrait de la capture d'écran ci-dessous montre le tunnel « gre1 » créé.

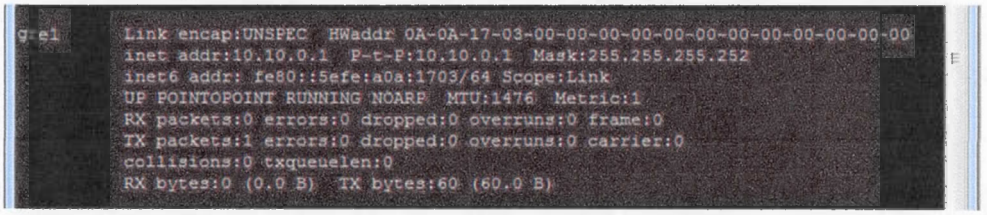


Figure 6.6 Création manuelle du tunnel gre1

Dans la figure ci-dessus, nous avons à la deuxième ligne après l'étiquette « inet addr : » l'adresse IP du tunnel « gre1 » qui est 10.10.0.1 et le masque de sous réseau qui est 255.255.255.252 correspondant à « /30 ».

Pour configurer les interfaces Gre1 et Gre2 sur le VLSR2, nous avons effectué les opérations précédentes en remplaçant le fichier « greScriptHost1» par le fichier « greScriptVLSR2 ». L'extrait de la capture d'écran suivant montre les tunnels « gre1 » et « gre2 » créés.

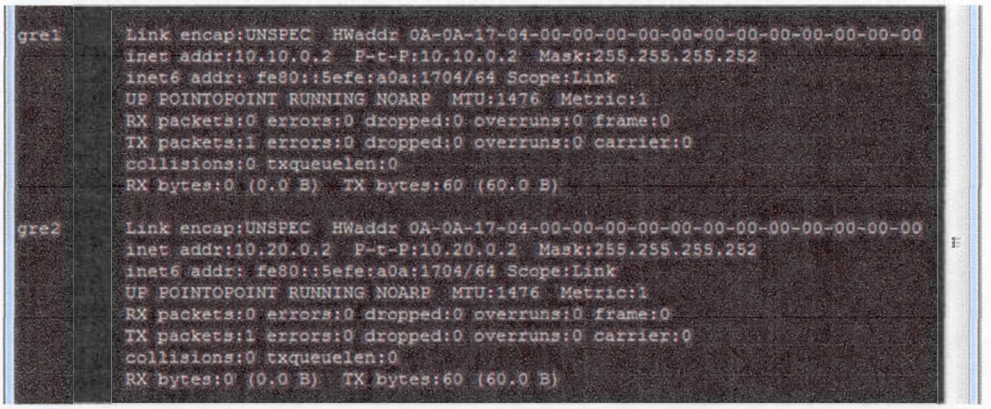


Figure 6.7 Création manuelle des tunnels gre1 et gre2

Dans la figure ci-dessus, nous avons à la deuxième ligne après l'étiquette « inet addr : » du tunnel gre1 l'adresse IP 10.10.0.2 et le masque de sous réseau qui est 255.255.255.252 correspondant à « /30 ». Il en est de même de la deuxième ligne concernant le tunnel « gre2 » où l'adresse IP est 10.20.0.2.

Pour configurer les interfaces Gre 2 et 3 sur le VLSR1, nous avons effectué les opérations précédentes en remplaçant le fichier « greScriptVLSR2» par le fichier «greScriptVLSR1». Un extrait de la capture d'écran ci-dessous montre les tunnels « gre2 » et « gre3 » créés.

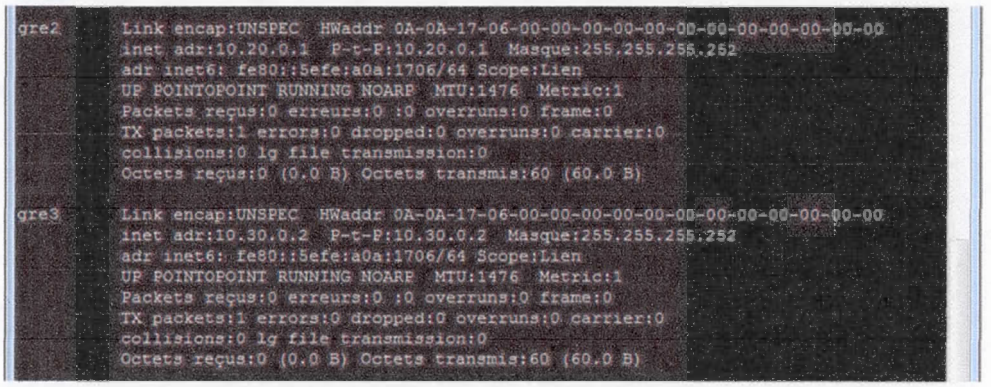


Figure 6.8 Création manuelle des tunnels gre2 et gre3

Dans la figure précédente, nous avons à la deuxième ligne après l'étiquette « inet addr : » du tunnel gre2 l'adresse IP 10.20.0.1 et le masque de sous réseau qui est 255.255.255.252 correspondant à « /30 ». Il en est de même à la deuxième ligne concernant le tunnel « gre3 » où l'adresse IP est 10.30.0.2.

Pour configurer l'interface Gre3 sur le CSA/OFS, nous avons effectué les opérations précédentes en remplaçant le fichier « greScriptVLSR1» par le fichier «greScriptHost2». Un extrait de la capture d'écran montre le tunnel « gre3 » créé.

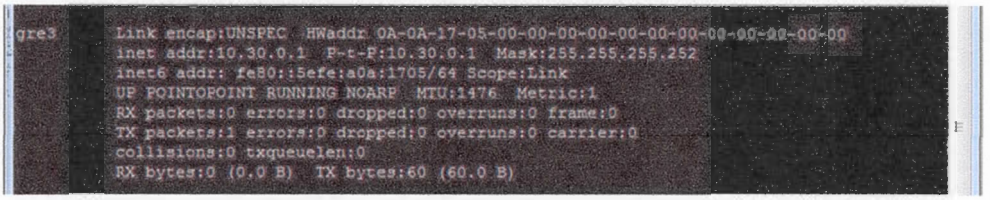


Figure 6.9 Création manuelle du tunnel gre3

A la deuxième ligne après l'étiquette « inet addr : », nous avons l'adresse IP du tunnel « gre3 » qui est 10.30.0.1 et le masque de sous réseau qui est 255.255.255.252 correspondant à « /30 ».

6.2.5 Configuration du VLSR

Le VLSR est l'ensemble composé d'un PC (exécutant un logiciel de plan de contrôle basé sur GMPLS où DRAGON est installé) et d'une structure de commutateur. Il traduit les messages de signalisation RSVP-TE en des commandes de commutateur local, et l'agent SNMP-TL1 traduit ces commandes en TL1 qui sont comprises par les ROADM Cisco 15 454. Il permet en outre de créer un LSP entre le dispositif source et le dispositif de destination grâce au contrôleur OpenFlow-GMPLS via la communication interne entre OpenFlow et GMPLS.

Pour la configuration des VLSR, outre les tunnels GRE configurés, il faut aussi démarrer les composants du domaine GMPLS-Dragon. Pour cela, il faut effectuer les commandes suivantes :

Se connecter en mode racine : ***sudo – s*** et entrer le mot de passe

Entrer dans le répertoire « usr/local/dragon/bin » à l'aide de la commande « ***cd /usr/local/dragon/bin*** »

Démarrer le VLSR à l'aide de la commande « ***./dragon.sh start-vlsr*** »

Un extrait de la capture d'écran montre les composants zebra, ospf, rsvp et dragon démarrés et qui sont mis en surbrillance pour une bonne visibilité.

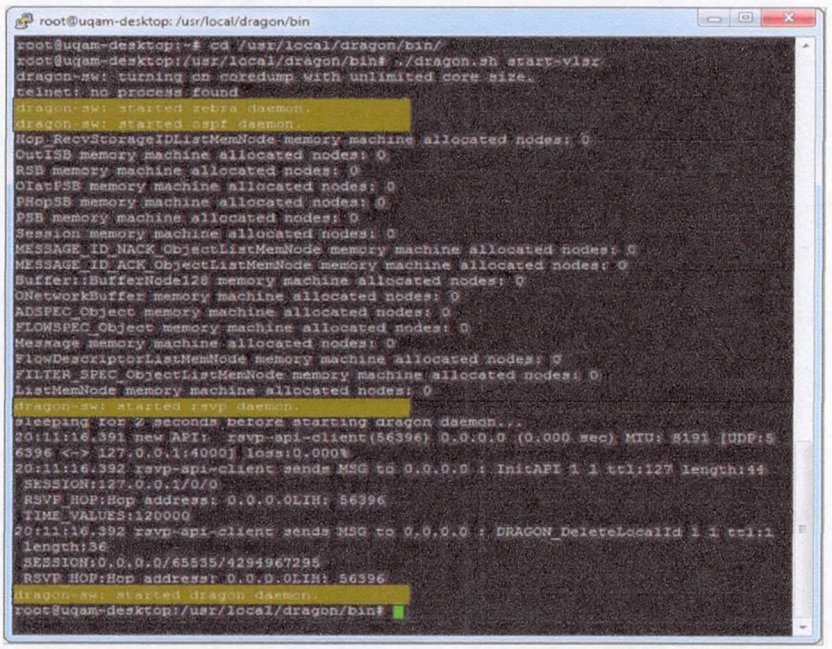


Figure 6.10 Démarrage du VLSR

Après la configuration de l'environnement OpenFlow-GMPLS, décrivons l'implémentation de notre plan de contrôle.

6.3 Description de l'implémentation du plan de contrôle OpenFlow-GMPLS

Cette section décrit tous les modules des éléments qui ont été implémentés dans le cadre de notre recherche. Elle décrit aussi la passerelle OpenFlow-GMPLS et son fonctionnement via la séquence de circulation des données dans l'architecture OpenFlow-GMPLS.

Cette description se fera à partir de l'architecture de test OpenFlow-GMPLS proposé ci-dessous.

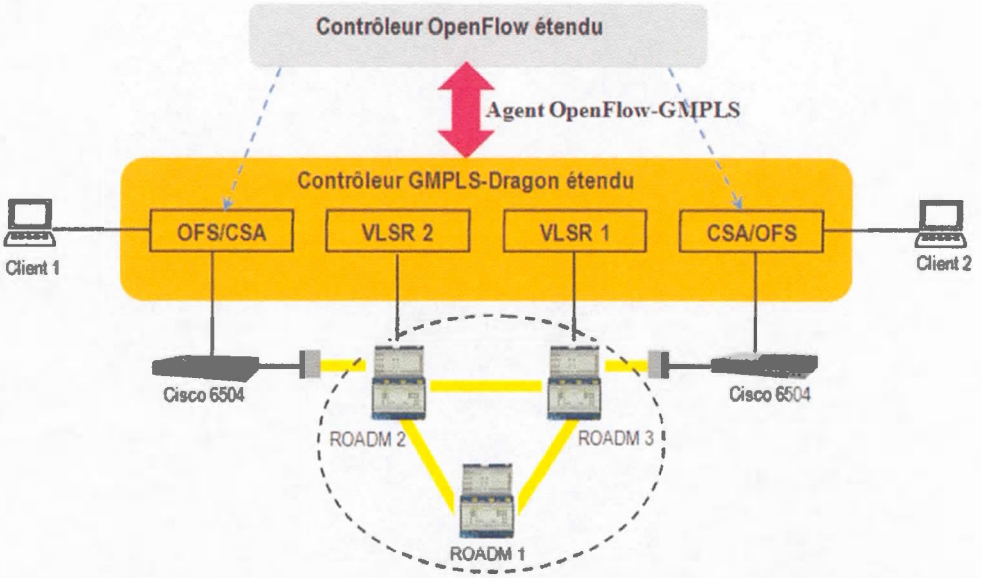


Figure 6.11 Architecture OpenFlow-GMPLS proposée

La figure ci-dessus montre l'architecture proposée du plan de contrôle OpenFlow-GMPLS. Sa description est presque similaire à celle de la figure 6.12, sauf qu'elle possède un contrôleur OpenFlow étendu et un contrôleur GMPLS-Dragon étendu.

Pour étendre le contrôleur OpenFlow, nous avons conçu, implémenté et ajouté trois éléments. Ces éléments permettent l'initialisation et la préparation de l'environnement réseau sans l'intervention de l'utilisateur. Traditionnellement, il faut une exécution des commandes de façon manuelle par l'utilisateur sur chaque commutateur OpenFlow pour activer le protocole OpenFlow, comme le montre les sous-sections 6.2.3.1 et 6.2.3.2 (configuration des Switch OFS/CSA et CSA/OFS). Aussi, lorsqu'il faut créer des interfaces GRE et démarrer les VLSR sur toutes les machines du domaine GMPLS, l'exécution des commandes de façon manuelle est requise, comme le montre la sous-section 6.2.4 (configuration des interfaces GRE). Nous avons solutionné cet état de chose. Chaque élément est décrit à partir de son fonctionnement.

6.3.1 1er élément (Activation automatique du protocole OpenFlow)

Le premier élément est chargé d'activer automatiquement le protocole OpenFlow sur tous les commutateurs OpenFlow du réseau. En effet, les commandes sont passées par le contrôleur OpenFlow-GMPLS aux commutateurs OpenFlow présents dans le réseau ; ces derniers utilisent le protocole OpenFlow standard pour s'auto-activer. Par exemple, lorsque le protocole OpenFlow est activé sur le OFS/CSA d'adresse IP 10.10.23.3, nous avons la console de l'environnement de programmation qui se présente comme ci-dessous :

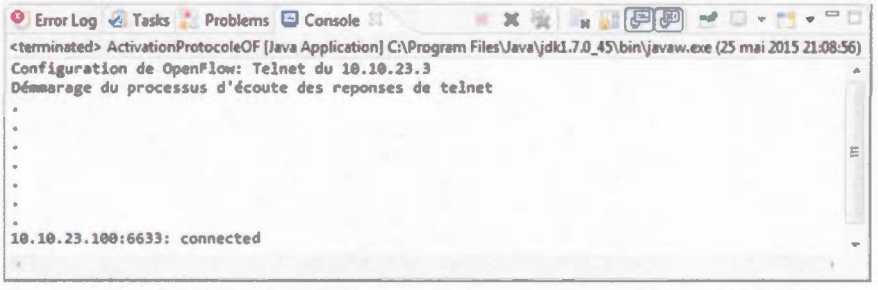


Figure 6.13 Activation du protocole OpenFlow sur OFS/CSA

La dernière ligne dans la console indique que le commutateur OpenFlow est connecté au contrôleur OpenFlow-GMPLS (d'adresse IP : 10.10.23.100) via le port 6633. Cela signifie que le protocole OpenFlow est activé sur ce commutateur. La capture Wireshark lors de cette phase se présente comme ci-dessous :

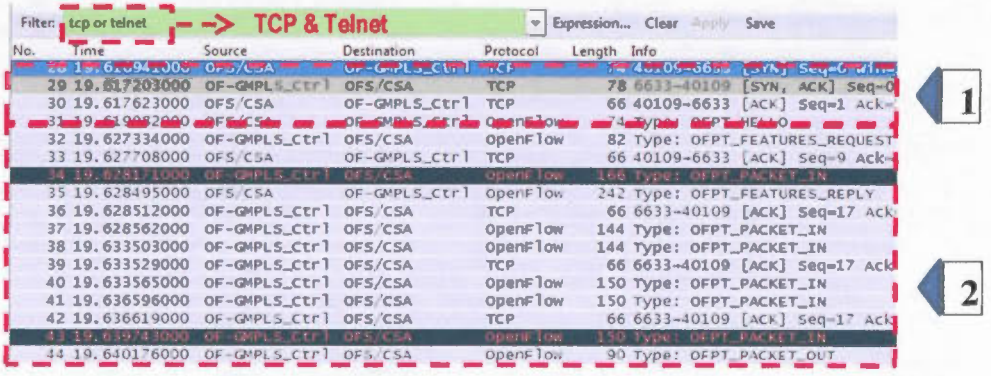


Figure 6.14 Échange des messages du processus d'activation du protocole OpenFlow sur OFS/CSA

A partir de la capture précédente, nous observons en (1) l'établissement d'une connexion TCP en 3 étapes entre le OFS/CSA (Switch OpenFlow / CSA) et le contrôleur OpenFlow-GMPLS (OF-GMPLS_Ctrl).

1 - Le OFS/CSA envoie un segment [SYN] au contrôleur OF-GMPLS_Ctrl

2 - Le contrôleur OF-GMPLS_Ctrl répond par un segment [SYN, ACK] à OFS/CSA

3 - Le OFS/CSA confirme l'établissement de la connexion par un segment [ACK]

En (2), l'échange de données d'ouverture de session se fait avec accusé de réception. Cela concerne les informations de session du OFS/CSA (login de connexion et mot de passe, nom de l'utilisateur racine et mot de passe). Dans la colonne « Info » de la figure précédente, nous avons plusieurs messages dont nous donnons la description.

Les messages « OFPT_HELLO » permettent de synchroniser la version supportée de OpenFlow.

Les messages « OFPT_FEATURES_REQUEST » permettent de connaitre les ports disponibles.

Les messages « OFPT_PACKET_IN », « OFPT_PACKET_OUT» permettent au Switch OpenFlow et au contrôleur de communiquer pendant le transfert des commandes afin que le processus d'auto-activation soit effectif.

Aussi, lorsque le protocole OpenFlow est activé sur le CSA/OFS (Switch OpenFlow servant aussi de CSA) d'adresse IP 10.10.23.5, nous avons la console de l'environnement de programmation qui se présente comme ci-dessous :

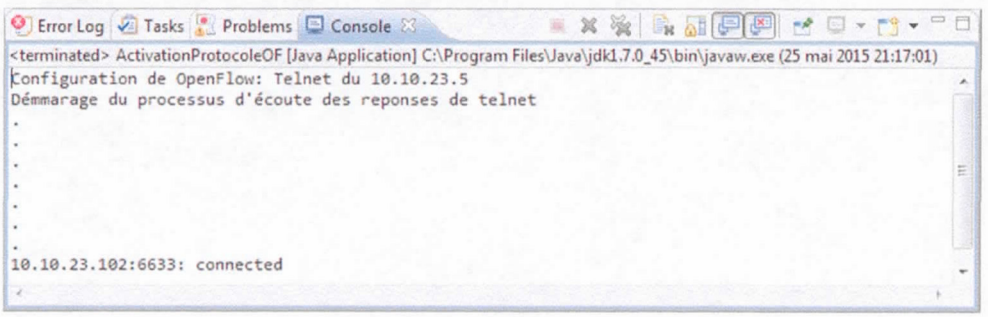


Figure 6.15 Activation du protocole OpenFlow sur CSA/OFS

La capture Wireshark lors de cette phase se présente comme ci-dessous :

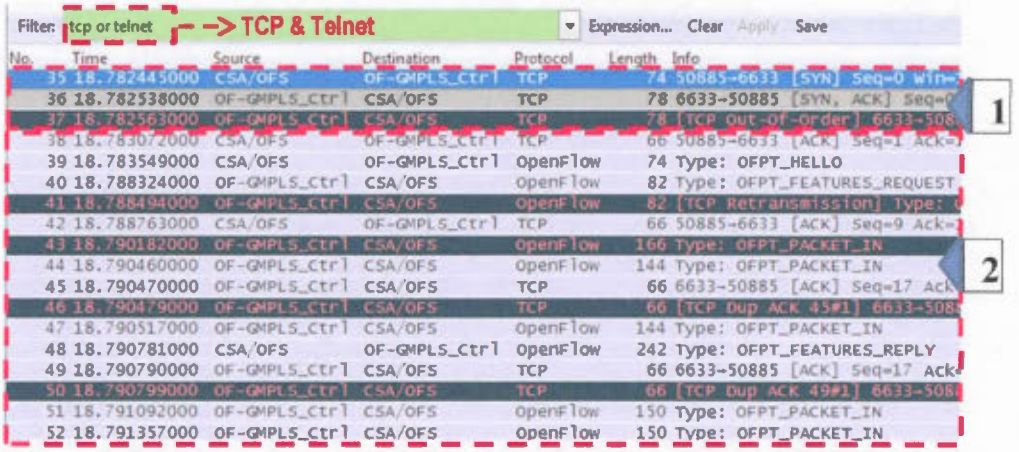


Figure 6.16 Échange des messages du processus d'activation du protocole OpenFlow sur CSA/OFS

A partir de cette capture, nous observons à la partie (1) l'établissement d'une connexion TCP en 3 étapes entre le CSA/OFS (Switch OpenFlow / CSA) et le contrôleur OpenFlow-GMPLS (OF-GMPLS_Ctrl). A la partie (2), l'échange des données d'ouverture de session se fait avec accusé de réception comme précédemment.

Cet élément d'activation automatique du protocole OpenFlow fonctionnerait aussi bien si l'on possédait beaucoup plus de commutateurs OpenFlow dans l'environnement réseau.

Le code que nous avons développé pour ce module se trouve à l'appendice A1, et a permis d'étendre le code standard de OpenFlow.

6.3.2 2ième élément (Création automatique des interfaces GRE)

Le second élément est chargé de créer automatiquement toutes les interfaces GRE sur tous les équipements intermédiaires dans le domaine GMPLS-Dragon. Dans notre cas, les interfaces Gre1, Gre2 et Gre3 sont créées.

Une fois ces interfaces créées, nous avons la console de l'environnement de programmation pour le VLSR d'adresse IP 10.10.23.4 qui se présente comme suit :

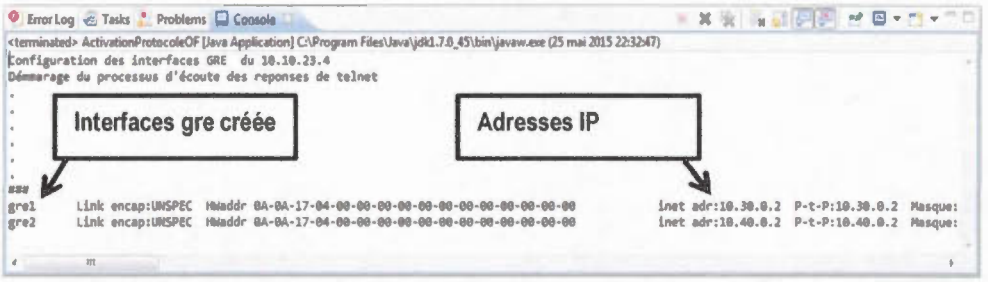


Figure 6.17 Création automatique des tunnels gre1 et gre2

La capture Wireshark lors de cette phase se présente comme ci-dessous :

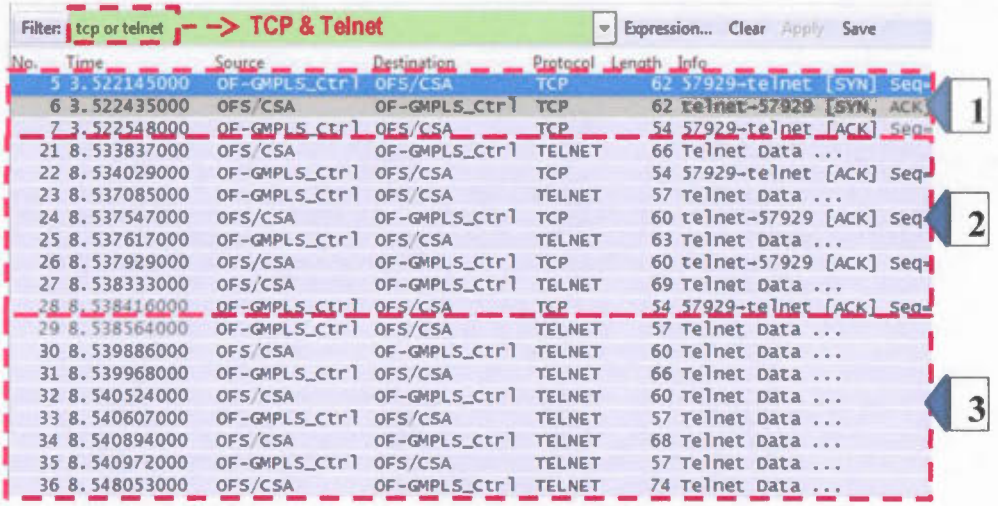


Figure 6.18 Échange des messages dans le processus de création automatique des tunnels Gre

A partir de cette capture, nous observons à la partie (1) l'établissement d'une connexion TCP en 3 étapes entre le contrôleur (OF-GMPLS_Ctrl) et le OFS/CSA (Switch OpenFlow servant aussi de CSA):

1 - Le contrôleur OF-GMPLS_Ctrl envoie un segment [SYN] au OFS/CSA

2 - Le OFS/CSA répond par un segment [SYN, ACK]

3 - Le contrôleur OF-GMPLS_Ctrl confirme par un segment [ACK]

A la partie (2), l'échange des données d'ouverture de session se fait avec accusé de réception. Cela concerne les informations du système distant (login de connexion et mot de passe, utilisateur racine et mot de passe).

La partie (3) est la phase de transfert des commandes où le contrôleur OpenFlow-GMPLS exécute les commandes sur le système distant ; ce dernier répond à chaque exécution d'une commande afin que le contrôleur transfère la commande suivante.
Il en est de même de toutes les autres interfaces GRE créées sur tous les équipements intermédiaires du domaine GMPLS-Dragon. Le code que nous avons développé concernant ce module se trouve à l'appendice A2, et a permis d'étendre le code standard de OpenFlow.

6.3.3 3ième élément (Démarrage automatique des composants GMPLS-Dragon)

Le troisième élément est chargé de démarrer automatiquement les composants GMPLS de Dragon sur tous les équipements intermédiaires dans le domaine GMPLS. Il s'agit des *daemon* zebra, ospf, rsvp et dragon. Après que ces éléments aient été démarrés sur le VLSR (VLSR d'adresse IP 10.10.23.4), nous avons la console de l'environnement de programmation qui se présente comme ci-dessous :

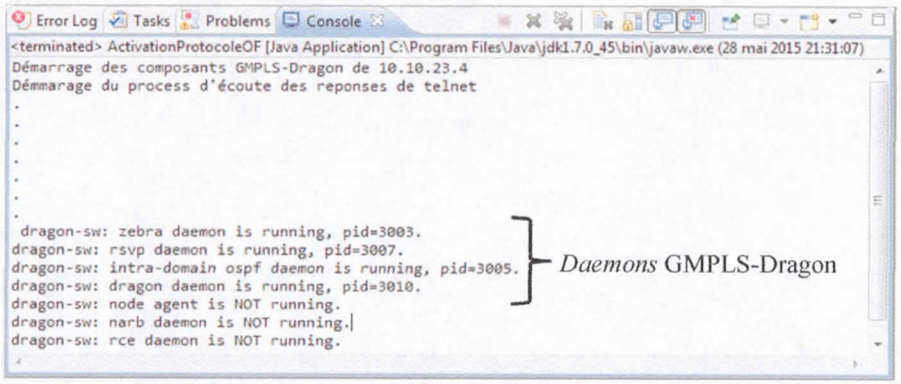


Figure 6.19 Démarrage automatique des composants du domaine GMPLS-Dragon

Nous observons que les quatre *daemon* GMPLS de Dragon (zebra, rsvp, ospf et dragon) sont en exécution. La capture Wireshark suivante prise pendant l'exécution de cette phase montre que l'algorithme se déroule avec les mêmes séquences que précédemment.

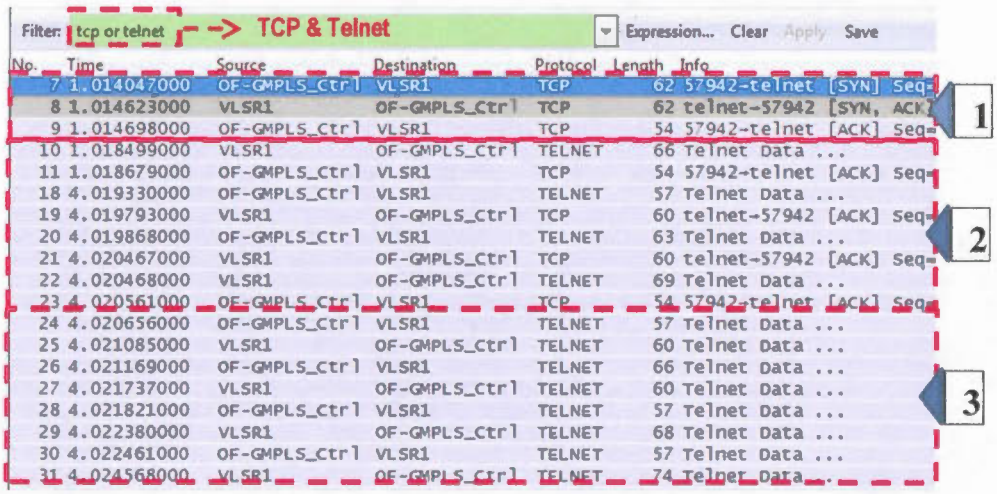


Figure 6.19 Échange des messages dans le processus de démarrage automatique des composants de GMPLS-Dragon

Le code que nous avons développé pour ce module se trouve à l'appendice A3, et a permis d'étendre aussi le code standard de OpenFlow.

Une fois que ces trois éléments aient été exécutés par notre plan de contrôle OpenFlow-GMPLS, l'environnement réseau est prêt à transmettre. Tous les équipements réseaux intermédiaires (commutateurs OpenFlow, CSA et VLSR) viennent ainsi d'être initialisés.

6.3.4 Passerelle OpenFlow-GMPLS

La passerelle OpenFlow-GMPLS ou agent OpenFlow-GMPLS permet la communication entre les portions OpenFlow et GMPLS dans le contrôleur. Elle effectue la création automatiquement de deux LSP (un principal et un de sauvegarde) entre un CSA source et un CSA de destination. Cette création est particulière et différente de celle exécutée dans les autres approches en ce sens que, non seulement elle n'est pas déclenchée manuellement, et en plus son élément déclencheur n'appartient pas au domaine GMPLS comme c'est souvent le cas, mais au domaine OpenFlow. Dans le guide de référence de DRAGON, en passant par plusieurs articles et mémoires traitants de la création des LSP, il est le plus souvent déclenché de façon manuelle à l'invite de commande ((USC) *et al.*, 2008

p. 22 ; Carela Español, 2007 p. 29 ; Meijerink et Prickaerts, 2006 p. 30). Le LSP peut être déclenché automatiquement par la signalisation de MPLS-TE dans le réseau du client (Kumaki, 2008 p. 5), ou par la mise en place d'un lien PSC TE (*Packet Switched Capable Traffic Engineering*) créé en utilisant un message de chemin RSVP-TE envoyé à partir du nœud source directement à l'autre extrémité de la même liaison (Martinez *et al.*, 2011 p. 3). Le code de l'agent OpenFlow-GMPLS se trouve à l'appendice A4, et son fonctionnement est expliqué dans la séquence de circulation des données dans l'environnement réseau.

6.3.5 Séquence de circulation de données dans l'architecture OpenFlow-GMPLS

Cette séquence est celle d'une première information envoyée entre le Client 1 et le Client 2.

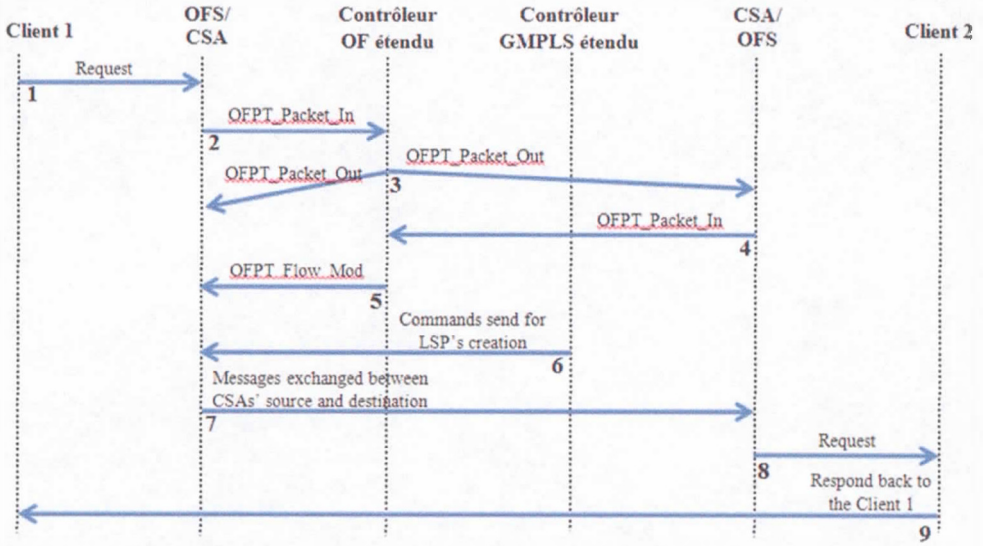


Figure 6.20 Séquence de circulation de données dans le contrôleur OpenFlow-GMPLS

Dans la figure ci-dessus, nous observons les étapes ci-dessous :

Étape 1 : le paquet va du Client 1 au OFS/CSA (Switch OpenFlow/CSA)

Étape 2 : le Switch envoie un « OFPT_Packet_In » au contrôleur OpenFlow étendu via le port 6633 si le paquet reçu ne correspond à aucune entrée dans la table de flux du Switch (c'est-à-dire que la destination du paquet n'est pas connue par le Switch).

Étape 3 : le contrôleur OpenFlow étendu envoie un message broadcast « OFPT_Packet_Out» à tous les ports des Switch OpenFlow avec qui il est connecté.

Étape 4 : le Switch OpenFlow de destination envoie un « OFPT_Packet_In » au contrôleur (réponse ARP contenant l'adresse MAC de destination).

Étape 5 : le contrôleur OpenFlow étendu ordonne au Switch d'ajouter un flux particulier à sa table de flux via un message « OFPT_Flow_Mod». Ce flux contient le numéro de port que doit emprunter le paquet pour se rendre à destination.

C'est à ce moment qu'intervient l'agent OpenFlow-GMPLS représenté dans la figure ci-dessous.

Légende

Contrôleur OpenFlow étendu

Contrôleur GMPLS étendu

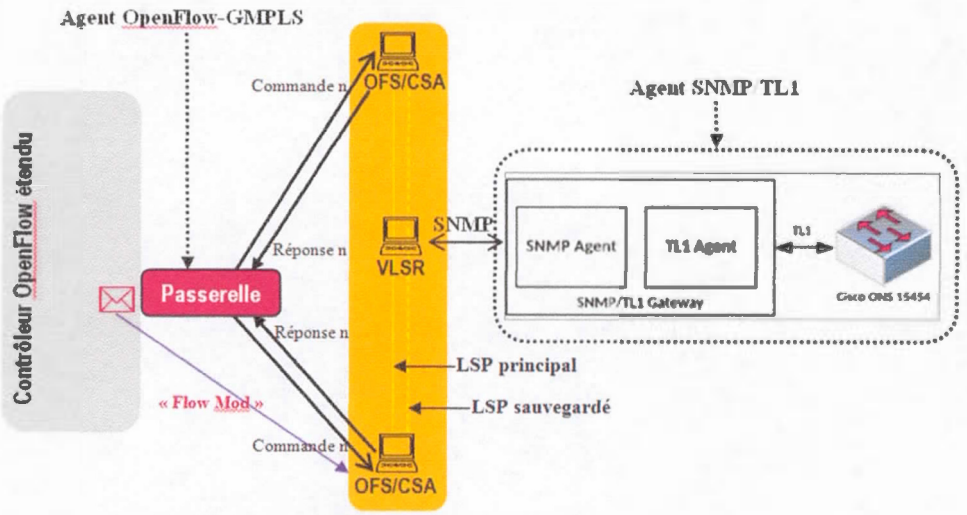


Figure 6.21 Fonctionnement de l'agent OpenFlow-GMPLS

Étape 6 : lorsque l'agent OpenFlow-GMPLS détecte qu'un message OFPT_Flow_Mod ou « Flow_Mod » est envoyé vers un Switch OF (OFS/CSA ou CSA/OFS), Il se connecte via la commande Telnet au plan de contrôle GMPLS (sur le CSA source) en passant les paramètres

d'ouverture de session, et attend la réponse de ce dernier. Une fois que le CSA ait répondu, l'algorithme ci-dessous que nous avons conçu est exécuté.

Algorithme :

Methode etablirLSP(idSource, idDestination){
Lire les paramètres de connexion Telnet (fichier ini)
Établir connexion TCP à ordinateur_cible
Initialiser le compteur de commande à 0
Tant que ordinateur_cible est connecté {
Lire les textes renvoyés par ordinateur_cible
Si le texte contient « : » {
Si nous sommes à la position 0, alors envoyer à ordinateur_cible le nom utilisateur et incrémenter
Si nous sommes à la position 1, alors envoyer à ordinateur_cible le mot de passe utilisateur et incrémenter position}
Sinon si le texte contient « ~$» et que nous sommes à la position 2 {
Envoyer la commande d'appel vers Telnet de dragon et incrémenter position}
Sinon si le texte contient « User Access Verification» et que nous sommes à la position 3 {
Envoyer le mot de passe utilisateur de dragon et incrémenter position}
Sinon si le texte contient « >» {
Envoyer la commande « edit lsp nom_LSP» à dragon et incrémenter position}
Sinon à chaque apparition de « #» {
Envoyer successivement dans l'ordre les commandes suivantes :
Envoyer commande 1
Recevoir réponse 1
Envoyer commande 2
Recevoir réponse 2
...
...
Envoyer commande n
Recevoir réponse n
Incrémenter à chaque fois position}
}
}

Après l'exécution de cet algorithme, le VLSR envoie des commandes SNMP à la passerelle pour la création de deux LSP (un LSP principal et un LSP de sauvegarde). Ces messages SNMP sont transmis à un agent SNMP/TL1. Ce dernier est chargé de les traduire en commandes TL1 pour qu'elles soient compréhensibles par les ROAMD Cisco ONS 15454 (Bahnasy, Idoudi et Elbiaze, 2015).

Pendant cette étape, nous avons obtenu les captures Wireshark ci-dessous :

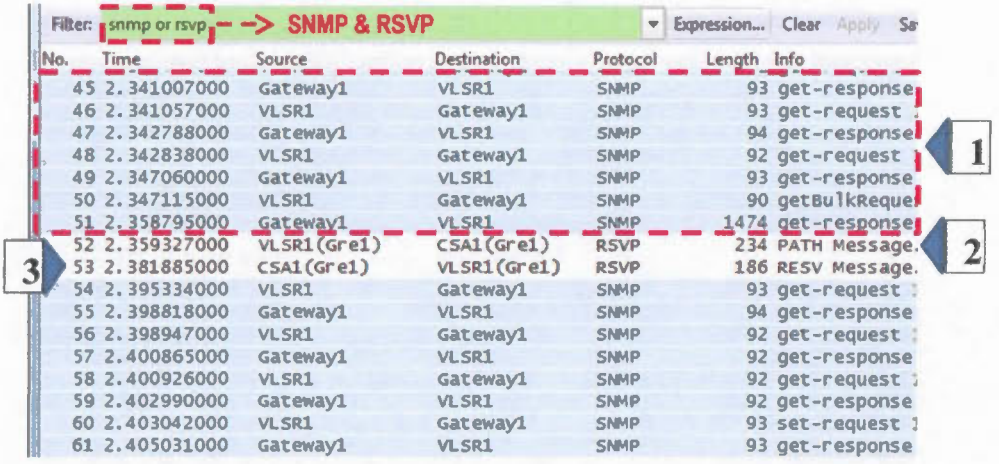


Figure 6.22 Échange des messages dans le processus de création automatique du LSP

Nous observons à la partie (1) les messages SNMP échangés entre un VLSR (VLSR1) et la passerelle OpenFlow-GMPLS (Gateway1). La partie (2) montre la ligne où le VLSR1 envoie un « *PATH Message* » au CSA1 via l'interface Gre1. Ce dernier répond avec un « *RESV Message* » à la ligne de la partie (3). Ceci montre que le LSP a été créé.

Étape 7 : les messages sont échangés entre le CSA source (OFS/CSA) et le CSA de destination (CSA/OFS).

Étape 8 : le Switch de destination envoie le flux de paquets vers le Client 2.

Étape 9 : le Client 2 envoie un message de réponse au Client 1.

A travers les étapes précédentes, le flux de paquets a circulé du Client 1 vers le Client 2.

Puisque notre plan de contrôle OpenFlow-GMPLS a initialisé l'environnement réseau de la figure 6.11, les LSP ont alors été créés entre les équipements intermédiaires sources (OFS/CSA) et destinations (CSA/OFS) afin d'établir la communication entre le Client 1 et le Client 2.

A présent que notre plan de contrôle OpenFlow-GMPLS est fonctionnel, nous passons maintenant à son évaluation.

6.4 Évaluation du plan de contrôle OpenFlow-GMPLS

Cette section montre les temps d'établissement des chemins dans l'architecture réseau proposée. Elle présente aussi une comparaison de notre plan de contrôle à un plan de contrôle basé sur OpenFlow d'une part, et à un autre basé sur GMPLS d'autre part afin de juger de son acceptabilité.

Pour évaluer la solution proposée, nous allons présenter les résultats d'expérimentation et les valider.

6.4.1 Résultat d'expérimentation de notre solution

Le tableau ci-dessous présente les temps d'établissement des chemins entre un nœud source et un nœud de destination. Ces données n'incluent pas les temps de configuration des équipements OF et GMPLS.

Tableau 6.1 Temps de transfert des données pour chaque chemin

	Contrôleur OF	RSVP-TE	ROADM 2	ROADM 1	ROADM 3	Total (ms)
Path1	28	141	105	-	105	362
	152					
Path2	32	143	94	39	103	394
	158					

Dans ce tableau, Path1 représente le chemin principal et Path2 le chemin de sauvegarde. Les nœuds du chemin principal sont : OFS/CSA (Contrôleur OF / RSVP-TE) → ROADM2 → ROADM 3 → CSA/OFS. Ceux du chemin de sauvegarde sont : OFS/CSA → ROADM2 → ROADM 1 → ROADM 3 → CSA/OFS.

Les colonnes « Contrôleur OF » et « RSVP-TE » contiennent les temps de traitement des informations dans chaque portion du contrôleur OpenFlow-GMPLS. La première colonne concerne la partie OF, et la seconde la partie GMPLS. Ces temps ne sont pas cumulatifs en ce sens que lorsque la partie OF commence son traitement, elle passe la main à la partie GMPLS avant la fin de son traitement. Les deux processus évoluent alors parallèlement pendant un laps de temps.

6.4.2 Validation des résultats

Bien que la validation et la comparaison sur deux seuls chemins ne semblent pas suffisantes pour aboutir à des conclusions générales sur la pertinence et l'efficacité de ces approches, nous avons comparé nos résultats (tableau 6.1) à ceux d'autres plans de contrôle. Le choix des solutions (plan de contrôle) avec lesquelles nous avons comparé nos résultats a été fait en fonction de deux critères principaux. Le premier critère est le même environnement technique (même cadre empirique et mêmes équipements). Le deuxième critère est la métrique de mesure (même simulateur intégré personnalisé). Notre solution a été comparée à deux autres plans de contrôle, l'un utilisant l'approche OpenFlow, et l'autre l'approche GMPLS. À partir de cette comparaison, nous obtenons les temps de latence des plans de contrôle concernant les trois principales approches de recherche évoquées dans la revue de littérature. Nous avons les résultats ci-dessous :

Tableau 6.2 Temps de transfert des données de deux solutions (OpenFlow étendue et GMPLS)

OpenFlow Extension Solution					
Switch Establishment					
	OF Controller	ROADM 2	ROADM 1	ROADM 3	Total (ms)
Path1	16	100	-	100	216
Path2	18	90	30	101	239
GMPLS Solution					
Switch Establishment					
	RSVP-TE	ROADM 2	ROADM 1	ROADM 3	Total (ms)
LSP	130	110	-	100	340

Source : (Bahnasy, Idoudi et Elbiaze, 2015)

Dans ce tableau, nous constatons que les temps de transfert du chemin Path1 est de 216 ms pour la solution OpenFlow étendue et de 340 ms pour la solution GMPLS.

Dans notre cas (tableau 6.1) nous avons obtenu un résultat de 362 ms en combinant OpenFlow et GMPLS-Dragon. Nos résultats sont convenables et ne sont pas surprenants à cause du déploiement complexe de GMPLS en raison de sa nature distribuée, du nombre de protocoles et des interactions entre les différentes couches. Aussi, afin de garantir qu'un LSP soit pleinement établi avant que le flux n'arrive au niveau du nœud optique entrant (ROADM2), OpenFlow doit attendre pendant un temps (temps de latence de signalisation

RSVP-TE) pour démarrer la transmission de flux. De plus, notre plan de contrôle possède un module de reprise après rupture de lien, contrairement aux deux autres plans de contrôle avec lesquels nous avons fait la comparaison. Ce module de reprise qui tourne en arrière-plan ajoute une légère lourdeur dans le fonctionnement de notre plan de contrôle.

Malgré les différences de 22 ms (362 - 340) entre notre plan de contrôle et celui basé uniquement sur GMPLS, et de 146 ms (362 - 216) avec celui basé seulement sur OpenFlow étendu, un plan de contrôle basé à la fois sur OpenFlow et GMPLS est nécessaire d'après les caractéristiques des protocoles des plans de contrôle représentées dans le tableau ci-dessous.

Tableau 6.3 Caractéristiques de comparaison des plans de contrôle

Plan de contrôle basé sur	**Réseau IP**	**Réseau optique**	**Réseau IP/Optique**
OpenFlow	Mature, efficace	Stade initial	Efficace
GMPLS	Efficace	Mature, efficace	Complexe
OpenFlow-GMPLS	Mature, efficace	Mature, efficace	Complexe, Mature, facilement évolutif

Nous observons qu'un plan de contrôle basé à la fois sur OpenFlow et GMPLS pour les réseaux IP/Optiques est complexe, mais mature et facilement évolutif. De ce fait, des recherches sur son efficacité (temps de traitement dans les différents noeuds) devront encore être plus poussées pour rivaliser en termes d'efficacité aux autres types des plans de contrôle.

Après cette évaluation, nous pouvons nous poser la question suivante : qu'arrive-t-il si un lien dans l'architecture réseau est rompu ?

6.5 Comportement du plan de contrôle OpenFlow-GMPLS après rupture de lien

Dans cette section, nous allons expliquer le fonctionnement de notre plan de contrôle après une rupture de lien via la description des évènements s'y déroulant. Nous allons aussi présenter les temps de rupture et de remise en service d'un LSP, et les temps de reprise après rupture du chemin principal.

Il était alors important pour nous de penser à une reprise après panne. En effet, toute panne de lien sur le plan de données a des répercussions sur le plan de contrôle. Autrement dit, lorsqu'un lien physique est coupé, le lien logique (LSP) l'est aussi. Nous nous occupons

d'abord de gérer le cas où le lien logique (LSP) est rompu; ensuite, nous ajouterons ce temps à celui du chemin de backup (Path2) lorsque le chemin principal (Path1) est rompu.
Pour cela, nous avons donc fait le choix d'étendre GMPLS-Dragon. Cette extension consiste en l'ajout d'un élément de reprise dans le domaine GMPLS. Le code du module d'élément de reprise que nous avons développé est présenté à l'appendice A5.
Lorsqu'un LSP est rompu, que ce soit d'origine physique (rupture physique d'un lien) ou d'origine logique (rupture d'un LSP à l'aide de la commande « ***delete lsp nomLSP*** »), il y a envoi des messages retournés le long du chemin d'accès au nœud d'entrée pour annoncer à tous les nœuds la rupture du LSP. A ce moment, le nœud d'entrée utilise une voie différente en contournant la panne pour alerter les nœuds supplémentaires (nœuds au-delà de la panne) sur ce chemin, afin que ces nœuds poursuivent la démolition du LSP concerné (Adami *et al.*, 2005 ; Ceccarelli, Caviglia et Fondelli, 2011).

La meilleure façon pour nous de décrire ou d'expliquer le fonctionnement de notre agent de reprise après panne est de le faire dans la description des évènements se déroulant lors d'une rupture. Le cas que nous allons décrire dans notre environnement réseau (dans le domaine GMPLS qui constitue l'ensemble des équipements intermédiaires) est celui où trois nœuds en aval sont concernés par la panne (CSA1, VLSR1 et CSA2). La figure ci-dessous donne une représentation de la situation.

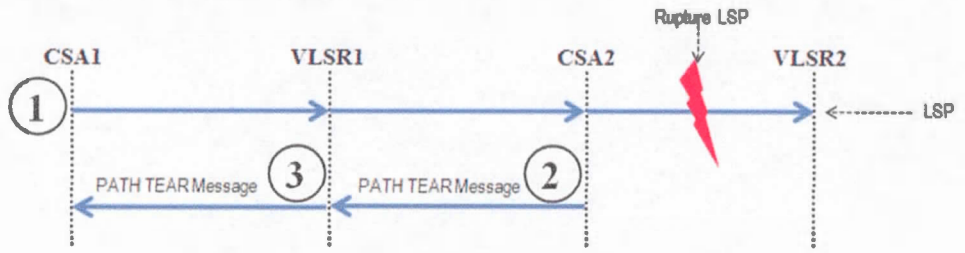
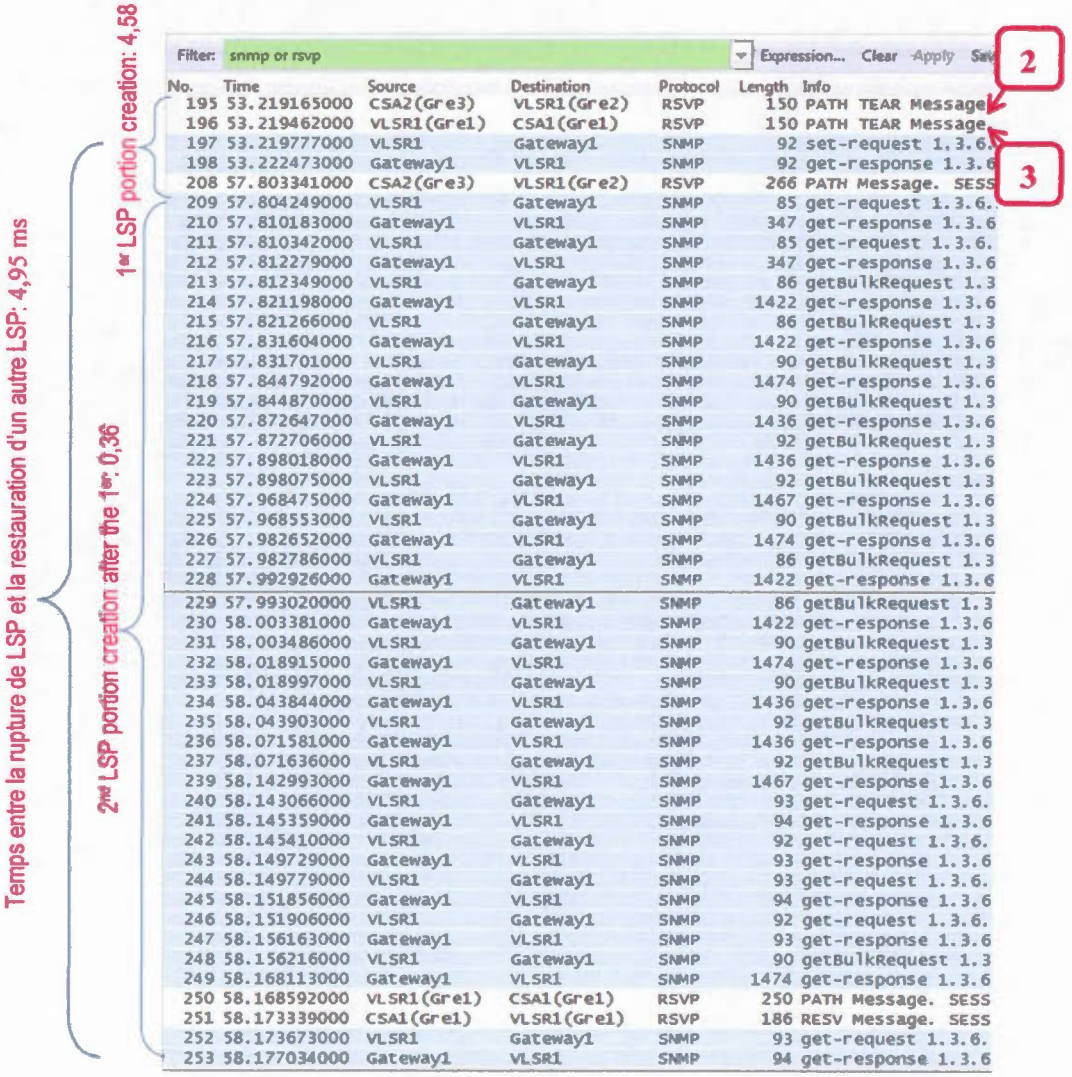


Figure 6.23 Séquence de rupture de LSP

La figure 6.23 montre les étapes d'une rupture de LSP entre les nœuds CSA2 et VLSR2. Après cette rupture, le nœud CSA2 envoie un message « PATH TEAR » (2) pour informer le précédent (VLSR1) de la rupture du lien entre CSA2 et VLSR2, afin que la portion de LSP entre les deux nœuds (VLSR1 et CSA2) soit rompue. Le nœud VLSR1 envoie à son tour un message « PATH TEAR » (3) pour informer le nœud précédent (CSA1) de la rupture du lien afin que la portion de LSP entre les deux nœuds (CSA1 et VLSR1) soit également rompue, et

ainsi de suite. S'il y avait d'autres nœuds après le VLSR2, le CSA1 (considéré comme le nœud d'entrée) devait à son tour envoyer en contournant la panne, un message « PATH TEAR » au nœud de sortie afin que par le procédé susmentionné, tous les nœuds du réseau soient informés de la rupture du lien.

La capture wireshark du cas susmentionné nous donne le temps mis pour les étapes (2) et (3), ainsi que le temps de reprise après panne.



No.	Time	Source	Destination	Protocol	Length	Info
195	53.219165000	CSA2(Gre3)	VLSR1(Gre2)	RSVP	150	PATH TEAR Message
196	53.219462000	VLSR1(Gre1)	CSA1(Gre1)	RSVP	150	PATH TEAR Message
197	53.219777000	VLSR1	Gateway1	SNMP	92	set-request 1.3.6.
198	53.222473000	Gateway1	VLSR1	SNMP	92	get-response 1.3.6
208	57.803341000	CSA2(Gre3)	VLSR1(Gre2)	RSVP	266	PATH Message. SESS
209	57.804249000	VLSR1	Gateway1	SNMP	85	get-request 1.3.6.
210	57.810183000	Gateway1	VLSR1	SNMP	347	get-response 1.3.6
211	57.810342000	VLSR1	Gateway1	SNMP	85	get-request 1.3.6.
212	57.812279000	Gateway1	VLSR1	SNMP	347	get-response 1.3.6
213	57.812349000	VLSR1	Gateway1	SNMP	86	getBulkRequest 1.3
214	57.821198000	Gateway1	VLSR1	SNMP	1422	get-response 1.3.6
215	57.821266000	VLSR1	Gateway1	SNMP	86	getBulkRequest 1.3
216	57.831604000	Gateway1	VLSR1	SNMP	1422	get-response 1.3.6
217	57.831701000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
218	57.844792000	Gateway1	VLSR1	SNMP	1474	get-response 1.3.6
219	57.844870000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
220	57.872647000	Gateway1	VLSR1	SNMP	1436	get-response 1.3.6
221	57.872706000	VLSR1	Gateway1	SNMP	92	getBulkRequest 1.3
222	57.898018000	Gateway1	VLSR1	SNMP	1436	get-response 1.3.6
223	57.898075000	VLSR1	Gateway1	SNMP	92	getBulkRequest 1.3
224	57.968475000	Gateway1	VLSR1	SNMP	1467	get-response 1.3.6
225	57.968553000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
226	57.982652000	Gateway1	VLSR1	SNMP	1474	get-response 1.3.6
227	57.982786000	VLSR1	Gateway1	SNMP	86	getBulkRequest 1.3
228	57.992926000	Gateway1	VLSR1	SNMP	1422	get-response 1.3.6
229	57.993020000	VLSR1	Gateway1	SNMP	86	getBulkRequest 1.3
230	58.003381000	Gateway1	VLSR1	SNMP	1422	get-response 1.3.6
231	58.003486000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
232	58.018915000	Gateway1	VLSR1	SNMP	1474	get-response 1.3.6
233	58.018997000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
234	58.043844000	Gateway1	VLSR1	SNMP	1436	get-response 1.3.6
235	58.043903000	VLSR1	Gateway1	SNMP	92	getBulkRequest 1.3
236	58.071581000	Gateway1	VLSR1	SNMP	1436	get-response 1.3.6
237	58.071636000	VLSR1	Gateway1	SNMP	92	getBulkRequest 1.3
239	58.142993000	Gateway1	VLSR1	SNMP	1467	get-response 1.3.6
240	58.143066000	VLSR1	Gateway1	SNMP	93	get-request 1.3.6.
241	58.145359000	Gateway1	VLSR1	SNMP	94	get-response 1.3.6
242	58.145410000	VLSR1	Gateway1	SNMP	92	get-request 1.3.6.
243	58.149729000	Gateway1	VLSR1	SNMP	93	get-response 1.3.6
244	58.149779000	VLSR1	Gateway1	SNMP	93	get-request 1.3.6.
245	58.151856000	Gateway1	VLSR1	SNMP	94	get-response 1.3.6
246	58.151906000	VLSR1	Gateway1	SNMP	92	get-request 1.3.6.
247	58.156163000	Gateway1	VLSR1	SNMP	93	get-response 1.3.6
248	58.156216000	VLSR1	Gateway1	SNMP	90	getBulkRequest 1.3
249	58.168113000	Gateway1	VLSR1	SNMP	1474	get-response 1.3.6
250	58.168592000	VLSR1(Gre1)	CSA1(Gre1)	RSVP	250	PATH Message. SESS
251	58.173339000	CSA1(Gre1)	VLSR1(Gre1)	RSVP	186	RESV Message. SESS
252	58.173673000	VLSR1	Gateway1	SNMP	93	get-request 1.3.6.
253	58.177034000	Gateway1	VLSR1	SNMP	94	get-response 1.3.6

Figure 6.24 Échange des messages du processus de rupture de LSP

Dans la capture ci-dessus, la ligne portant le numéro (2) indique qu'un message « PATH TEAR » a été envoyé du nœud CSA2 au nœud VLSR1. La ligne portant le numéro (3) indique qu'un message « PATH TEAR » a été envoyé du nœud VLSR1 au nœud CSA1.

Le temps mis pour les étapes (2) et (3) correspondant à la destruction des portions du LSP est d'environ 0,0003 ms (c'est-à-dire 53,219462 - 53,219165).

C'est alors qu'entre en jeu notre agent de reprise après panne. Lorsqu'il detecte un PATH TEAR Message, il met en service le LSP de backup qui devient le LSP principal, et crée en mémoire un autre LSP qui devient à son tour le LSP de backup. C'est ainsi que tous les LSP qui sont mis en service entre les équipements d'extrémités dans le domaine GMPLS fonctionnent.

La figure 6.24 montre aussi le temps mis pour créer la première portion de LSP (nœuds CSA2 et VLSR1) qui est de 4,584176 ms (c'est-à-dire 57,803341 - 53,219165). Pour la seconde portion de LSP (nœuds VLSR1 et CSA1), le temps mis après la création de la première portion est de 0,365251 ms (c'est-à-dire 58,168592 -57,803341). D'où un temps total de 4,949427 ms.

Le tableau ci-dessous nous donne les temps entre la rupture totale d'un LSP et celui de la mise en service d'un autre. Ces temps ont été évalués en 10 tentatives pour obtenir une valeur moyenne maximale.

Tableau 6.4 Temps de rupture et de remise en service d'un LSP

Tentatives	1	2	3	4	5	6	7	8	9	10
Temps de reprise (ms)	4,95	4,89	4,97	4,94	4,90	4,92	4,95	4,90	4,94	4,93

Considérant le temps maximal de 5 ms, nous arrivons au temps de reprise total pour le chemin principal dans le tableau ci-dessous :

Tableau 6.5 Temps de reprise pour le chemin principal après rupture du lien

	OF Controller	RSVP-TE	ROADM 2	ROADM 1	ROADM 3	Total (ms)
Path2	37	148	94	39	103	399
	163					

Ces résultats sont attendus car le chemin de backup Path2 (OFS1/CSA1 → ROADM2 → ROADM 1 → ROADM 3 → OFS1/CSA1) comprend un nœud de plus que le chemin principal. En plus, il faut ajouter un temps de 5 ms (temps de recréation du LSP) au temps du chemin de backup qui est Path2.

A présent que nous avons réussi à bâtir, tester et évaluer un plan de contrôle basé à la fois sur OpenFlow et GMPLS, faisant fonctionner les équipements compatibles GMPLS ou non dans une architecture réseau respectant le principe de la technologie SDN, nous passons maintenant à la conclusion de ce chapitre.

6.6 Conclusion

Il était question de faire une implémentation d'un plan de contrôle OpenFlow-GMPLS pour centraliser de façon logique l'intelligence et l'état du réseau à partir des applications sur un super-ordinateur appelé contrôleur.

Pour réaliser cette implémentation, nous avons d'abord configuré l'environnement OpenFlow-GMPLS en configurant les postes clients, le contrôleur OpenFlow-GMPLS, les commutateurs OpenFlow, les interfaces GRE et les VLSR. Par la suite, nous avons effectué une description de l'implémentation à travers l'extension des protocoles OpenFlow standard et GMPLS-Dragon. Les trois éléments d'initialisation du plan de contrôle OpenFlow-GMPLS et l'élément de reprise après rupture de lien ont été conçus, implémentés et décrits. La passerelle OpenFlow-GMPLS a été aussi conçue, implémentée et son fonctionnement décrit. Enfin, nous avons évalué notre solution en présentant les résultats d'expérimentation et en les validant. A travers toutes ces étapes, nous avons pu mettre sur pied notre solution.

Une fois notre implémentation terminée, quelle conclusion pouvons-nous en faire ?

CONCLUSION

Notre recherche avait pour objectif de faire une implémentation pour assurer la gestion centralisée des réseaux multicouches IP/Optiques en permettant aux équipements non-compatibles GMPLS d'être fonctionnels dans un réseau GMPLS, tout en utilisant l'architecture SDN. Cet objectif a été atteint via la mise en œuvre d'un plan de contrôle basé à la fois sur OpenFlow et sur GMPLS-Dragon.

En effet, nous sommes partis de la conception et de la mise sur pied d'une architecture réseau IP/Optique respectant le principe SDN. Nous avons utilisé les protocoles OpenFlow standard et GMPLS-Dragon comme point de départ à l'implémentation. Nous avons fait une extension du protocole OpenFlow standard à l'aide de trois éléments (activation automatique du protocole OpenFlow, création automatique des interfaces GRE et démarrage automatique des composants GMPLS-Dragon) que nous avons conçu et implémenté. Nous avons également fait une extension de GMPLS-Dragon à l'aide d'un élément (élément de reprise) que nous avons conçu et implémenté. Les trois éléments d'extension de OpenFlow ont permis d'initialiser tous les équipements intermédiaires de l'environnement réseau afin qu'ils soient prêt à transmettre des données. L'élément d'extension de GMPLS-Dragon a permis l'activation du LSP secondaire en cas de rupture du LSP principal en 5 millisecondes environ. Afin que les portions IP et Optiques de l'environnement réseau puissent échanger des données, nous avons conçu et implémenté une passerelle permettant aux protocoles OpenFlow et GMPLS-Dragon de pouvoir communiquer entre eux. Cette passerelle OpenFlow-GMPLS traduit les instructions OpenFlow en instructions GMPLS et vice-versa. Nous avons enfin testé et évalué notre plan de contrôle OpenFlow-GMPLS, et nous l'avons comparé aux solutions OpenFlow étendue d'une part, et GMPLS d'autre part ; car ces deux solutions ont été implémentées dans le même cadre empirique que le nôtre, avec les mêmes équipements et nous avons utilisé la même métrique de mesure.

Bien que la comparaison sur deux seuls chemins ne semble pas suffisante pour aboutir à des conclusions générales sur la pertinence et l'efficacité de ces approches, les résultats montrent que la solution OpenFlow étendue serait la plus rapide. Nous observons d'après les caractéristiques de comparaison des plans de contrôle (tableau 6.3) qu'un plan de contrôle basé à la fois sur OpenFlow et GMPLS pour les réseaux IP/Optiques serait mature et facilement évolutif.

Aussi, notre plan de contrôle assure la restauration d'un lien réseau brisé dans l'architecture, contrairement aux deux autres plans de contrôle avec qui il a été comparé.

En plus, notre plan de contrôle basé sur OpenFlow et GMPLS permet aux équipements non-compatibles GMPLS de fonctionner dans un réseau GMPLS, contrairement aux autres plans de contrôle OpenFlow-GMPLS de l'heure.

Bien que nous ayons déployé toutes nos forces et compétences pour mener à bien cette recherche, nous avons toutefois identifié les limites suivantes :

- La petite taille de l'architecture réseau proposée; une architecture moyenne comportant plusieurs centaines de nœuds aurait permis de mieux apprécier les résultats de recherche.
- Le nombre de chemins sur lesquels les tests ont été réalisés; deux seuls chemins ne sauraient être suffisants pour généraliser les résultats de notre étude.
- Le cadre empirique qui est celui d'un laboratoire; ce cadre n'est pas assez pertinent contrairement à un cadre de réseau réel en exploitation.
- La métrique de mesure utilisée pour recueillir les résultats; nous avons utilisé un simulateur intégré personnalisé, contrairement à un simulateur universel standardisé.

Puisque ces limites constituent des failles à la validité des expérimentations menées, nous recommandons qu'elles soient prises en considération lors des recherches futures concernant les plans de contrôle OpenFlow-GMPLS.

Une question intéressante pour une recherche future serait donc de savoir si l'optimisation du temps de traitement dans un plan de contrôle basé à la fois sur OpenFlow et GMPLS, et gérant les équipements compatibles GMPLS ou non peuvent égaler voir surpasser en termes de temps de latence la solution OpenFlow étendue.

APPENDICE A

INSTALLATION DU LOGICIEL DRAGON

A.1 Installation de Dragon VLSR

Pour installer de façon basique le logiciel Dragon VLSR, il faut exécuter les commandes ci-dessous :

#tar –zxf dragon-sw-snapshot.xxxx.tar.gz *(décompresse le fichier dans un répertoire)*

Ensuite, il faut entrer dans le répertoire de travail où le paquet de DRAGON a été décompressé et taper les commandes suivantes:

 # ./do_build.sh *(configure et construit le paquet)*

 # sudo sh do_install.sh *(installe le logiciel DRAGON)*

Par défaut, le logiciel est installé dans le sous répertoire « /usr/local/dragon ».

Pour faire une installation étape par étape, il faut installer le logiciel DRAGON incluant KOM-RSVP (DRAGON RSVPTE) et GNU ZEBRA (y compris DRAGON OSPF-TE et le *daemon* dragon).

Pour installer KOM-RSVP (autre implémentation de RSVP), il faut entrer les commandes suivantes :

#cd kom-rsvp *(entre dans le répertoire kom-rsvp)*

#. /configure --with-snmp=/usr/local *(configure kom-rsvp avec snmp fourni ; les fichiers d'en-tête de net-snmp sont installés dans « /usr/local »).*

Après les étapes ci-dessus, il faut aussi exécuter les commandes suivantes :

 # gmake clean *(nettoie le répertoire)*

 # gmake depend *(reconstruit les fichiers de dépendances)*

 # gmake *(recompile le code)*

 # sudo gmake install *(installe les nouveaux fichiers binaires et de bibliothèque)*

Pour installer GNU ZEBRA, il faut entrer les commandes suivantes :

 # cd zebra *(entre dans le sous- répertoire « zebra » se trouvant dans le répertoire « dragon-sw »)*

#. /configure --prefix=/usr/local/dragon —enable-dragon *(configure le protocole ; le préfixe de la commande définit le répertoire où ZEBRA-OSPF doit être installé.)*

 # make *(compile le code)*

 # sudo make install *(installe les fichiers)*

A.2 Installation des tunnels GRE

Les configurations ci-dessous doivent être faites sur l'hôte 1.

/sbin/ip tunnel add *gre1* ***mode*** *gre* ***remote*** *129.174.42.12* ***local*** *129.174.43.90* ***ttl*** *255*

/sbin/ip link set *gre1* ***up***

/sbin/ip addr add *10.10.0.1* ***dev*** *gre1*

/sbin/ip route add *10.10.0.0/24* ***dev*** *gre1*

La première ligne nous permet d'ajouter un tunnel gre1. Ce tunnel utilise le mode de protocole GRE «gre». L'adresse distante (de la carte réseau du VLSR reliée au Hub) est 129.174.42.12. Les paquets de *tunneling* doivent provenir de 129.174.43.90 (adresse de la carte réseau de l'hôte 1 reliée au Hub). Le champ TTL des paquets a la valeur 255.

La deuxième ligne nous permet d'activer le tunnel gre1.

La troisième ligne nous permet d'affecter l'adresse IP 10.10.0.1 à l'interface gre1 (interface de l'hôte 1 reliée au Switch).

La quatrième ligne nous permet d'ajouter une route à la machine de contrôle avec l'adresse IP 10.10.0.0 et un masque de sous réseau « /24 ».

Les configurations ci-dessous doivent également être faites sur l'hôte 2.

/sbin/ip tunnel add *gre2* ***mode*** *gre* ***remote*** *129.174.42.12* ***local*** *129.174.42.221* ***ttl*** *255*

/sbin/ip link set *gre2* ***up***

/sbin/ip addr add *10.20.0.1* ***dev*** *gre2*

/sbin/ip route add *10.20.0.0/24* ***dev*** *gre2*

Ces quatre lignes sont interprétées de façon similaire aux précédentes (sous-section 3.5.1.1), à la différence que le nom du tunnel gre1 a été remplacé par gre2. Les adresses de la carte réseau de l'hôte 2 reliées au Hub (adresse locale), de l'interface gre2 et de la route à ajouter à la machine de contrôle ont changé.

Les configurations suivantes doivent être faites sur le VLSR. Concernant le tunnel gre1, nous devons avoir :

***/sbin/ip tunnel add** gre1 **mode** gre **remote** 129.174.43.90 **local** 129.174.42.12 **ttl** 255*

/sbin/ip link set** gre1 **up

***/sbin/ip addr add** 10.10.0.2 **dev** gre1*

***/sbin/ip route add** 10.10.0.0/24 **dev** gre1*

Ces quatre lignes sont interprétées de façon similaire à celles de la configuration de l'hôte 1, à la différence que l'adresse de l'hôte distant est celle de la carte réseau de l'hôte 1 reliée au hub, et l'adresse de l'hôte locale est celle de la carte réseau du VLSR reliée au hub. Aussi, l'adresse de l'interface gre1 est celle du Switch reliée à la carte réseau de l'hôte 1.

Concernant le tunnel gre2, nous devons avoir :

***/sbin/ip tunnel add** gre2 **mode** gre **remote** 129.174.42.221 **local** 129.174.42.12 **ttl** 255*

/sbin/ip link set** gre2 **up

***/sbin/ip addr add** 10.20.0.2 **dev** gre2*

***/sbin/ip route add** 10.20.0.0/24 **dev** gre2*

Ces quatre lignes sont interprétées de façon similaire à celles de la configuration de l'hôte 2, à la différence que l'adresse de l'hôte distant est celle de la carte réseau de l'hôte 2 reliée au hub, et l'adresse de l'hôte locale est celle de la carte réseau du VLSR reliée au hub. Aussi, l'adresse de l'interface gre2 est celle du Switch reliée à la carte réseau de l'hôte 2.

Dans le cas où le tunnel GRE n'est pas défini correctement, il peut être enlevé comme suit :

#/sbin/ip link set** gre1 **down

***#/sbin/ip tunnel del** gre1*

La première ligne permet d'arrêter le fonctionnement de l'interface gre1 du tunnel GRE, et la seconde permet de supprimer cette interface.

A.3 Processus de configuration de DRAGON OSPF-TE, DRAGON RSVP-TE, DRAGON *daemon* et ZEBRA

A.3.1 Configuration de DRAGON OSPF-TE

Pour configurer le protocole DRAGON OSPF-TE, il faut modifier le contenu du fichier nommé « ospfd.conf » (fichier contenant des commandes de configuration) sur les systèmes clients (hôte 1 et hôte 2) et sur le VLSR.

Ici, nous spécifions le nom de l'hôte, le mot de passe, les interfaces de tunnel GRE entre les hôtes, le VLSR et enfin les ID (identifiant) du routeur et la capacité de commutation. Les fichiers de configuration se trouvent dans le répertoire « /usr/local/dragon/etc ».

Note : « etc » est le nom d'un répertoire du système d'exploitation Linux.

a) Configuration de DRAGON OSPF-TE sur l'hôte 1

Pour configurer l'hôte 1, il faut ouvrir le fichier « /usr/local/dragon/etc/ospfd.conf », copier et coller ce qui suit puis enregistrer.

Note : les mots qui suivent le caractère « ! » sont des commentaires

```
!
! zebra-ospfd configuration file for cnl_host1
!
hostname cnl_host1-ospf
password [specify password]
log stdout
!
!
interface gre10
description GRE tunnel between cnl_host1 and cnl_narb
ip ospf network point-to-point
interface gre1
description GRE tunnel between cnl_host1 and cnl_vlsr
ip ospf network point-to-point
!
!
router ospf
ospf router-id 129.174.43.90
network 10.1.0.0/24 area 0.0.0.0
network 10.10.0.0/24 area 0.0.0.0
!
ospf-te router-address 129.174.43.90
!
ospf-te interface gre10
```

```
exit
!
ospf-te interface gre1
level gmpls
data-interface ip 10.1.10.2
swcap lsc encoding Ethernet
exit
!
line vty
!
```

b) Configuration de DRAGON OSPF-TE sur l'hôte 2

Pour configurer l'hôte 2, il faut ouvrir le fichier « /usr/local/dragon/etc/ospfd.conf », copier et coller ce qui suit puis enregistrer.

```
! zebra-ospfd configuration file for cnl_host2
!
hostname cnl_host2-ospf
password [specify password]
log stdout
!
!
interface gre2
description GRE tunnel between cnl_host2 and cnl_vlsr
ip ospf network point-to-point
!
!
router ospf
ospf router-id 129.174.42.221
network 10.20.0.0/24 area 0.0.0.0
ospf-te router-address 129.174.42.221
ospf-te interface gre2
level gmpls
data-interface ip 10.1.20.2
swcap lsc encoding Ethernet
exit
!
line vty
!
```

c) Configuration de DRAGON OSPF-TE sur le VLSR

Pour configurer le VLSR, il faut ouvrir le fichier « /usr/local/dragon/etc/ospfd.conf », copier et coller ce qui suit puis enregistrer.

```
!
! zebra-ospfd configuration file for cnl_vlsr
!
hostname cnl_vlsr-ospf
password [specify here]
log stdout
!
!
interface gre1
description GRE tunnel between cnl_vlsr and cnl_host1
ip ospf network point-to-point
!
!
interface gre2
description GRE tunnel between cnl_vlsr and cnl_host2
ip ospf network point-to-point
!
router ospf
ospf router-id 129.174.42.12
network 10.10.0.0/24 area 0.0.0.0
network 10.20.0.0/24 area 0.0.0.0
!
ospf-te router-address 129.174.42.12
!
ospf-te interface gre1
level gmpls
data-interface ip 10.1.10.1 protocol snmp switch-ip
10.1.1.2 switch-port 9
swcap lsc encoding Ethernet
exit
!
ospf-te interface gre2
level gmpls
data-interface ip 10.1.20.1 protocol snmp switch-ip
10.1.1.2 switch-port 10
swcap lsc encoding Ethernet
exit
!
line vty
!
```

A.3.2 Configuration de DRAGON RSVP-TE

Pour configurer le protocole DRAGON RSVP-TE, il faut spécifier les interfaces des tunnels GRE dans le fichier « RSVPD.conf » et le mettre dans le répertoire « /usr/local/dragon/etc ».

a) Configuration de DRAGON RSVP-TE sur l'hôte 1

Pour configurer l'hôte 1, il faut ouvrir le fichier « /usr/local/dragon/etc/RSVPD.conf », copier et coller ce qui suit puis enregistrer.

!
interface gre1 tc none mpls
api 4000

b) Configuration de DRAGON RSVP-TE sur l'hôte 2

Pour configurer l'hôte 2, il faut ouvrir le fichier « /usr/local/dragon/etc/RSVPD.conf », copier et coller ce qui suit puis enregistrer.

!
interface gre2 tc none mpls
api 4000

c) Configuration de DRAGON RSVP-TE sur le VLSR

Pour configurer le VLSR, il faut ouvrir le fichier « /usr/local/dragon/etc/RSVPD.conf », copier et coller ce qui suit puis enregistrer.

!
interface gre1 tc none mpls
interface gre2 tc none mpls
api 4000

A.3.3 Configuration du *daemon* DRAGON

Pour configurer le *daemon* DRAGON, il faut spécifier le nom d'hôte et le mot de passe pour chaque machine dans le fichier « dragon.conf» (fichier des commandes de configuration du *daemon* DRAGON) et le mettre dans le répertoire « /usr/local/dragon/etc ».

a) Configuration du *daemon* DRAGON sur l'hôte 1

Pour configurer l'hôte 1, il faut ouvrir le fichier « /usr/local/dragon/etc/dragon.conf », copier et coller ce qui suit puis enregistrer.

! DRAGON configuration file for cnl_host1
!
hostname cnl_host1-dragon
password [specify password here]

b) Configuration du *daemon* DRAGON sur l'hôte 2

Pour configurer l'hôte 2, il faut ouvrir le fichier « /usr/local/dragon/etc/dragon.conf », copier et coller ce qui suit puis enregistrer.

```
! DRAGON configuration file for cnl_host2
!
hostname cnl_host2-dragon
password [specify password here]
```

c) Configuration du *daemon* DRAGON sur le VLSR

Pour configurer le VLSR, il faut ouvrir le fichier « /usr/local/dragon/etc/dragon.conf », copier et coller ce qui suit puis enregistrer.

```
! DRAGON configuration file for cnl_host2
!
hostname cnl_vlsr-dragon
password [specify password here]
```

A.3.4 Configuration de ZEBRA

ZEBRA est un *daemon* qui fait fonctionner l'OSPF *daemon*. Il ne contient aucun ajout de DRAGON. Pour le configurer, il faut spécifier certaines informations pour chaque machine dans le fichier « zebra.conf» (fichier des commandes de configuration de ZEBRA) et le mettre dans le répertoire « /usr/local/dragon/etc ».

a) Configuration de ZEBRA sur l'hôte 1

Pour configurer l'hôte 1, il faut ouvrir le fichier « /usr/local/dragon/etc/zebra.conf », copier et coller ce qui suit puis enregistrer.

```
! zebra configuration file for cnl_host1
!
hostname cnl_host1-zebra
password [specify password]
! enable password [specify password]
!
! Interface description.
!
interface lo
interface gre1
interface gre10
! description test of desc.
```

```
!
!interface sit0
! multicast
!
! Static default route sample.
!
!log file zebra.log
```

b) Configuration de ZEBRA sur l'hôte 2

Pour configurer l'hôte 2, il faut ouvrir le fichier « /usr/local/dragon/etc/zebra.conf », copier et coller ce qui suit puis enregistrer.

```
! zebra configuration file for cnl_host2
!
hostname cnl_host2-zebra
password [specify password]
! enable password [specify password]
!
! Interface's description.
!
interface lo
interface gre2
! description test of desc.
!
!interface sit0
! multicast
!
! Static default route sample.
!
!log file zebra.log
```

c) Configuration de ZEBRA sur le VLSR

Pour configurer le VLSR, il faut ouvrir le fichier « /usr/local/dragon/etc/zebra.conf », copier et coller ce qui suit puis enregistrer.

```
! zebra configuration file for cnl_vlsr
!
hostname cnl_vlsr-zebra
password dragon
! enable password dragon
!
! Interface's description.
!
interface lo
interface gre1
```

```
interface gre2
! description test of desc.
!
! interface sit0
! multicast
!
! Static default route sample.
!
!log file zebra.log
```

APPENDICE B

EXTENSION EFFECTUÉE DANS LE CODE OPENFLOW

B.1 Code d'activation automatique du protocole OpenFlow

```
private String fileNameOF = "ParamActivOF.txt";
private String fileNameGmpls = "ParamActivGmpls.txt";
private TelnetTools telnet;
private static ConfigSwitchOF configSwitch;
private static boolean isOnGoing = false;

public static void main(String[] args) {
    ActivationProtocoleOF activ = new ActivationProtocoleOF();
    try {
        activ.activerOF();
    } catch (FileNotFoundException e) {
        System.out.println("Fichier de configuration inexistant");
        e.printStackTrace();
    }
    try {
        activ.activerGMPLS();
    } catch (FileNotFoundException e) {
        System.out.println("Fichier de configuration inexistant");
        e.printStackTrace();
    }

}

public void activerOF() throws FileNotFoundException {
    // lire le lichier dactivation
    List<ConfigSwitchOF> lstParam = LectureFichierOF
            .lireParamSwitchOF(fileNameOF);
    for (ConfigSwitchOF configSwitchOF : lstParam) {
        if (!configSwitchOF.getType().equalsIgnoreCase("OF"))
            continue;
```

```
while (isOnGoing) {
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }
}
isOnGoing = true;
configSwitch = configSwitchOF;
System.out
.println("configuration OF telnet du "+configSwitchOF.getIpSwitch());
// connection au serveur telnet

try {
    telnet = new TelnetTools(configSwitchOF.getIpSwitch(), 23);
} catch (Exception e) {
    System.out
            .println("Erreur lors de la connexion au serveur telnet");
    e.printStackTrace();
    return;
}

(new Thread(new Runnable() {

    @Override
    public void run() {
        System.out
                .println("demmarage du process decoute des reponses de telnet");
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }

        try {
            int position = 0;
            int ch;
            StringBuffer temp = new StringBuffer();
            do {
                try {
                    ch = telnet.getTelnetRetourCmd().read();// telnetRetourCmd.read();
                } catch (IOException e) {
                    e.printStackTrace();
                    System.out
                            .println("process d'ecoute encore non disponible");
                    return;
                }
                if (ch < 0)
                    return;
                //System.out.print((char) ch);
                System.out.flush();
                temp.append((char) ch);
                if (temp.toString().contains(":")
                        && (position <= 1)) {
                    temp = new StringBuffer();
                    synchoniserCommande(position++);
                    // } else if ((temp.toString().contains(":"))
                    // && (position == 1)) {
                    // temp = new StringBuffer();
                    // synchoniserCommande(position++);
```

```
            } else if (temp.toString().contains("-S")
                    && (position == 2)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("sudo")
                    && temp.toString().contains(
                            configSwitch.getUser())
                    && (position == 3)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("#")&& (position == 4)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }else if (temp.toString().contains(configSwitch.getRepOF()+"#")) {/
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }else if (temp.toString().contains(configSwitch.getIpVirtuelle()
                    + ":"
                    + configSwitch.getNumPort()+ ": connected")) {//serveur cor
                System.out
                .println(configSwitch.getIpVirtuelle()
                        + ":"
                        + configSwitch.getNumPort()+ ": not connected");
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }else if (temp.toString().contains(": waiting 4 seconds")) {//serve
                System.out
                .println(configSwitch.getIpVirtuelle()
                        + ":"
                        + configSwitch.getNumPort()+ ": not connected");
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }
            if (position > 7)
                return;
        } while (true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void synchoniserCommande(int i) {
    // se connecter
    try {
        switch (i) {
        case 0: {
            telnet.envoyerCommande(configSwitch.getUser());
            return;
        }
        case 1: {
            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 2: {

            telnet.envoyerCommande("sudo -s");
            return;
        }
        case 3: {

            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 4: {// #
            // 1 cnl_host2> edit lsp 5to3-A-
            telnet.envoyerCommande("cd "
                    + configSwitch.getRepOF());
            return;
        }
        case 5: {// configSwitchOF.getRepOF()#
            telnet.envoyerCommande("./udatapath/ofdatapath --detach punix:/var/run/dp0 -d "
                    + configSwitch.getIdSwitch()
                    + " -i "
                    + configSwitch.getInterfaceVersClient()
                    + ","
                    + configSwitch.getInterfaceVersSwitch());
            return;
        }
        case 6: {// configSwitchOF.getRepOF()#
            telnet.envoyerCommande("./secchan/ofprotocol unix:/var/run/dp0 tcp:"
                    + configSwitch.getIpVirtuelle()
                    + ":"
                    + configSwitch.getNumPort());
            return;
        }
        case 7: {// ancien 9
            telnet.deconnexionTelnet();
            isOnGoing = false;
            return;
        }

        default: {
            System.out
                    .println("Erreur lors de l'exécution des commandes sur le serveur telnet");
        }
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
start();
```

B.2 Code de création des interfaces GRE

```
private void activerGmplsGre(List<ConfigSwitchOF> lstParam) {
    for (ConfigSwitchOF configSwitchOF : lstParam) {
        if (!configSwitchOF.getType().equalsIgnoreCase("G"))
            continue;
        while (isOnGoing) {
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        isOnGoing = true;
        configSwitch = configSwitchOF;
        System.out
        .println("configuration GRE du "+configSwitchOF.getIpSwitch());
        // connection au serveur telnet
        try {
            telnet = new TelnetTools(configSwitchOF.getIpSwitch(), 23);
        } catch (Exception e) {
            System.out
                    .println("Erreur lors de la connexion au serveur telnet");
            e.printStackTrace();
            return;
        }

        (new Thread(new Runnable() {

            @Override
            public void run() {
                System.out
                        .println("demmarage du process decoute des reponses de telnet");
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }

                try {
                    int position = 0;
                    int ch;
                    StringBuffer temp = new StringBuffer();
                    StringBuffer msg = new StringBuffer();
                    do {
                        try {
                            ch = telnet.getTelnetRetourCmd().read();// telnetRetourCmd.reac
                        } catch (IOException e) {
                            e.printStackTrace();
                            System.out
                                    .println("process d'ecoute encore non disponible");
                            return;
                        }
                        if (ch < 0)
                            return;
```

```
            //System.out.print((char) ch);
            System.out.flush();
            temp.append((char) ch);
            if (temp.toString().contains(":")
                    && (position <= 1)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("~$")
                    && (position == 2)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("sudo")
                    && temp.toString().contains(
                            configSwitch.getUser())
                    && (position == 3)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("#")) {
                if (position==4){
                String[] val = temp.toString().substring(("./"
                        + configSwitch.getIdSwitch()).length()+1).split("gre");
                if (val.length>1){
                    System.out.println("###");
                    for(int i=1;i<val.length;i++){
                        System.out.println("gre"+val[i]);
                    }
                }
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }
            if (position > 6)
                return;
        } while (true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void synchoniserCommande(int i) {
    // se connecter
    try {
        switch (i) {
        case 0: {
            telnet.envoyerCommande(configSwitch.getUser());
            return;
        }
        case 1: {
            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 2: {

            telnet.envoyerCommande("sudo -s");
            return;
        }
        case 3: {

            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 4: {// #
            // 1 onl_host2> edit lsp 5to3-A-
            telnet.envoyerCommande("cd "
                    + configSwitch.getRepOF());
            return;
        }
        case 5: {// configSwitchOF.getRepOF()#
            telnet.envoyerCommande("./"
                    + configSwitch.getIdSwitch());
            return;
        }
        case 6: {// ancien 9
            telnet.deconnexionTelnet();
            isOnGoing = false;
            return;
        }

        default: {
            System.out
                    .println("Erreur lors de l'exécution des commandes sur le serveur telnet");
        }
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
.start();
```

B.4 Code d'activation des composants GMPLS

```
private void activerGmplsVlsr(List<ConfigSwitchOF> lstParam) {
    for (ConfigSwitchOF configSwitchOF : lstParam) {
        if (!configSwitchOF.getType().equalsIgnoreCase("VV"))
            continue;
        while (isOnGoing) {
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
        isOnGoing = true;
        configSwitch = configSwitchOF;
        System.out
        .println("configuration VLSR du "+configSwitchOF.getIpSwitch());
        // connection au serveur telnet
        try {
            telnet = new TelnetTools(configSwitchOF.getIpSwitch(), 23);
        } catch (Exception e) {
            System.out
                    .println("Erreur lors de la connexion au serveur telnet");
            e.printStackTrace();
            return;
        }

        (new Thread(new Runnable() {

            @Override
            public void run() {
                System.out
                        .println("demmarage du process decoute des reponses de telnet");
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }

                try {
                    int position = 0;
                    int ch;
                    StringBuffer temp = new StringBuffer();
                    do {
                        try {
                            ch = telnet.getTelnetRetourCmd().read();// telnetRetourCmd.read();
                        } catch (IOException e) {
                            e.printStackTrace();
                            System.out
                                    .println("process d'ecoute encore non disponible");
                            return;
                        }
                        if (ch < 0)
                            return;
```

```
            if (position==7)
            System.out.print((char) ch);
            System.out.flush();

            if (String.valueOf((char) ch).equals("." )){
                if (position==7)
                System.out.println("");
            }

            temp.append((char) ch);
            if (temp.toString().contains(":")
                    && (position <= 1)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("~$")
                    && (position == 2)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("sudo")
                    && temp.toString().contains(
                            configSwitch.getUser())
                    && (position == 3)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains(configSwitch.getRepOF()+"#")) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("#")) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }
            if (position > 7)
                return;
        } while (true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void synchoniserCommande(int i) {
    // se connecter
    try {
        switch (i) {
        case 0: {
            telnet.envoyerCommande(configSwitch.getUser());
            return;
        }
        case 1: {
            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 2: {
            telnet.envoyerCommande("sudo -s");
            return;
        }
        case 3: {
            telnet.envoyerCommande(configSwitch.getPwdUser());
            return;
        }
        case 4: {// #
            // 1 cnl_host2> edit lan 5to3-A-
            telnet.envoyerCommande("cd "
                    + configSwitch.getRepOF());
            return;
        }
        case 5: {// configSwitchOF.getRepOF()#
            telnet.envoyerCommande("./dragon.sh "
                    + configSwitch.getIdSwitch());
            return;
        }
        case 6: {// configSwitchOF.getRepOF()#
            telnet.envoyerCommande("./dragon.sh status");
            return;
        }
        case 7: {// ancien 9
            telnet.deconnexionTelnet();
            isOnGoing = false;
            return;
        }
        default: {
            System.out
                    .println("Erreur lors de l'exécution des commandes sur le serveur telnet");
        }
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
start();
```

B.5 Code de l'agent OpenFlow-GMPLS

```
package org.openflow.example.ofGmpls;

import java.io.IOException;
import java.io.OutputStream;
import java.net.URL;
import java.net.URLConnection;

import thor.net.DefaultTelnetTerminalHandler;
import thor.net.TelnetURLConnection;

/*
 * @autor Patrice KUEKEM TCHOUA.
 */
public class ControleurOpenFlowGMPLS extends DefaultTelnetTerminalHandler {
    private TelnetTools telnet;
    private static ControleurOpenFlowGMPLS controleurOpenFlowGMPLS;
    String src, dest;

    private static int idOFSrc = 0xac106265;
    private static int idOFDest = 0xac106266;
    private static String ipSrc = "10.10.23.5";
    private static String ipDest = "10.10.23.3";
    private static String nomLsp = "Stm3-A";
    private static String userTelnet = "xxxx";
    private static String pwdTelnet = "xxxx";
    private static String ipSrcTelnet = "10.10.23.5";
    private static String pwdDragon = "xxxx";
    private static String idSwitchOFSrc = "004E46324301";
    private static String idSwitchOFDest = "004E46324302";

    private ControleurOpenFlowGMPLS() {
        super();
    }

    public static ControleurOpenFlowGMPLS getInstance() {
        if (controleurOpenFlowGMPLS == null) {
            controleurOpenFlowGMPLS = new ControleurOpenFlowGMPLS();
        }
        return controleurOpenFlowGMPLS;
    }

    /*
     * Etablir LSP a partir du switchOpenFlow
     */
    public void etablirLSP(long idSwitch) {
        if (idSwitch == Long.parseLong(idSwitchOFSrc, 16)) {
            etablirLSP(idOFSrc, idOFDest);
        } else if (idSwitch == Long.parseLong(idSwitchOFDest, 16)) {
            etablirLSP(idOFDest, idOFSrc);
        }
    }
```

```
/*
 * Etablir LSP entre deux identifiant OF de machine
 */
public void etablirLSP(int idScr, int idDest) {
    if ((idScr == idOFSrc) && (idDest == idOFDest)) {
        src = ipSrc;
        dest = ipDest;
    } else if ((idScr == idOFDest) && (idDest == idOFSrc)) {
        src = ipDest;
        dest = ipSrc;
    } else {
        return;
    }
    // connection au serveur telnet
    try {
        telnet = new TelnetTools(ipSrcTelnet, 23);
    } catch (Exception e) {
        System.err.println("Erreur lors de la connexion au serveur telnet");
        e.printStackTrace();
        return;
    }

    (new Thread(new Runnable() {

        @Override
        public void run() {
            System.err
                    .println("demarrage du process decoute des reponses de telnet");
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }

            try {
                int position = 0;
                int ch;
                StringBuffer temp = new StringBuffer();
                do {
                    try {
                        ch = telnet.getTelnetRetourCmd().read();// telnetRetourCmd.read();
                    } catch (IOException e) {
                        e.printStackTrace();
                        System.err
                                .println("process d'ecoute encore non disponible");
                        return;
                    }
                    if (ch < 0)
                        return;
                    System.out.print((char) ch);
                    System.out.flush();
                    temp.append((char) ch);
                    if (temp.toString().contains(":") && (position == 0)) {
                        temp = new StringBuffer();
                        synchoniserCommande(0);
                        position = 1;
                    } else if ((temp.toString().contains(":"))
                            && (position == 1)) {
                        temp = new StringBuffer();
                        synchoniserCommande(position++);
```

```
            } else if (temp.toString().contains("--$")) {
                temp = new StringBuffer();
                synchoniserCommande(3);
                position = 3;
            } else if ((temp.toString()
                    .contains("User Access Verification"))
                    && (position == 3)) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains(">")) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("#")) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }
            if (position > 5)
                return;
        } while (true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void synchoniserCommande(int i) {
    // se connecter
    try {
        switch (i) {
        case 0: {
            telnet.envoyerCommande(userTelnet);
            return;
        }
        case 1: {
            telnet.envoyerCommande(pwdTelnet);
            return;
        }
        case 2: {

            telnet.envoyerCommande("telnet localhost 2611");
            return;
        }
        case 3: {

            telnet.envoyerCommande(pwdDragon);
            return;
        }
        case 4: {

            // 1 cnl_host2> edit lsp 5to3-A-
            telnet.envoyerCommande("edit lsp " + nomLsp);
            return;
        }
        case 5: {

            // 2 cnl_host2(edit-lsp-5to3-A-)# set source ip-address
            // 10.10.23.5
            // lsp-id 150 destination ipaddress 10.10.23.3 tunnel-id
            // 250
            telnet.envoyerCommande("set source ip-address " + src
                    + " lsp-id 150 destination ip-address " + dest
                    + " tunnel-id 250");
            return;
        }
        case 6: {

            // 3 cnl_host2(edit-lsp-5to3-A-)# set bandwidth giga
            // awcap l2sc
            // encoding
            // ethernet gpid ethernet
            telnet.envoyerCommande("set bandwidth giga awcap l2sc encoding ethernet gpid ethernet");
            return;
        }
        case 7: {

            // 4 cnl_host2(edit-lsp-5to3-A-)# exit
            telnet.envoyerCommande("exit");
            return;
        }
        case 8: {
            telnet.envoyerCommande("commit lsp " + nomLsp);
            return;
        }
```

```
        case 5: {
            telnet.deconnexionTelnet();
            return;
        }

        default: {
            System.err
                    .println("Erreur lors de l'exécution des commandes sur le serveur telnet");
        }
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
start();


 public static void main(String[] args) throws IOException {
     ControleurOpenFlowGMPLS ctrl = ControleurOpenFlowGMPLS.getInstance();
     ctrl.etablirLSP(0xac106265, 0xac106266);
 }
```

```
package org.openflow.example.ofGmpls;
//@autor Patrice KUEKEM TCHOUA

public class ConfigSwitchOF {
    private String type ="";
    private String ipSwitch = "";
    private String repOF = "";
    private String idSwitch = "";
    private String interfaceVersClient = "";
    private String interfaceVersSwitch = "";
    private String ipVirtuelle = "";
    private String numPort = "";
    private String user = "";
    private String pwdUser = "";

    public ConfigSwitchOF(String ligne) {
        super();

        String[] tabLigne = ligne.split(";");
        int i=1;
        if (tabLigne[0].equalsIgnoreCase("o")){
            type="O";
            ipSwitch = tabLigne[i++];
            repOF = tabLigne[i++];
            idSwitch = tabLigne[i++];
            interfaceVersClient = tabLigne[i++];
            interfaceVersSwitch = tabLigne[i++];
            ipVirtuelle = tabLigne[i++];
            numPort = tabLigne[i++];
            user = tabLigne[i++];
            pwdUser = tabLigne[i++];
        }else if (tabLigne[0].equalsIgnoreCase("g")){
            type="G";
            ipSwitch = tabLigne[i++];
            repOF = tabLigne[i++];//nom du repertoire contenant le script
            idSwitch = tabLigne[i++];// nom du script
            user = tabLigne[i++];
            pwdUser = tabLigne[i++];
        }else if (tabLigne[0].equalsIgnoreCase("v")){
            type="V";
            ipSwitch = tabLigne[i++];
            repOF = tabLigne[i++];//nom du repertoire dragon
            idSwitch = tabLigne[i++];// de la commande dragon
            user = tabLigne[i++];
            pwdUser = tabLigne[i++];
        }
    }
```

```
/**
 * @return the ipSwitch
 */
public String getIpSwitch() {
    return ipSwitch;
}

/**
 * @param ipSwitch
 *            the ipSwitch to set
 */
public void setIpSwitch(String ipSwitch) {
    this.ipSwitch = ipSwitch;
}

/**
 * @return the repOF
 */
public String getRepOF() {
    return repOF;
}

/**
 * @param repOF
 *            the repOF to set
 */
public void setRepOF(String repOF) {
    this.repOF = repOF;
}

/**
 * @return the idSwitch
 */
public String getIdSwitch() {
    return idSwitch;
}

/**
 * @param idSwitch
 *            the idSwitch to set
 */
public void setIdSwitch(String idSwitch) {
    this.idSwitch = idSwitch;
}

/**
 * @return the interfaceVersClient
 */
public String getInterfaceVersClient() {
    return interfaceVersClient;
}
```

```
    /**
     * @param interfaceVersClient
     *            the interfaceVersClient to set
     */
    public void setInterfaceVersClient(String interfaceVersClient) {
        this.interfaceVersClient = interfaceVersClient;
    }

    /**
     * @return the interfaceVersSwitch
     */
    public String getInterfaceVersSwitch() {
        return interfaceVersSwitch;
    }

    /**
     * @param interfaceVersSwitch
     *            the interfaceVersSwitch to set
     */
    public void setInterfaceVersSwitch(String interfaceVersSwitch) {
        this.interfaceVersSwitch = interfaceVersSwitch;
    }

    /**
     * @return the ipVirtuelle
     */
    public String getIpVirtuelle() {
        return ipVirtuelle;
    }

    /**
     * @param ipVirtuelle
     *            the ipVirtuelle to set
     */
    public void setIpVirtuelle(String ipVirtuelle) {
        this.ipVirtuelle = ipVirtuelle;
    }

    /**
     * @return the numPort
     */
    public String getNumPort() {
        return numPort;
    }

    /**
     * @param numPort
     *            the numPort to set
     */
    public void setNumPort(String numPort) {
        this.numPort = numPort;
    }
```

```
    /**
     * @return the user
     */
    public String getUser() {
        return user;
    }

    /**
     * @param user
     *            the user to set
     */
    public void setUser(String user) {
        this.user = user;
    }

    /**
     * @return the pwdUser
     */
    public String getPwdUser() {
        return pwdUser;
    }

    /**
     * @param pwdUser
     *            the pwdUser to set
     */
    public void setPwdUser(String pwdUser) {
        this.pwdUser = pwdUser;
    }

    /**
     * @return the type
     */
    public String getType() {
        return type;
    }

    /**
     * @param type the type to set
     */
    public void setType(String type) {
        this.type = type;
    }
}
```

```
/**
 * @return the ipSwitch
 */
public String getIpSwitch() {
    return ipSwitch;
}

/**
 * @param ipSwitch
 *            the ipSwitch to set
 */
public void setIpSwitch(String ipSwitch) {
    this.ipSwitch = ipSwitch;
}

/**
 * @return the repOF
 */
public String getRepOF() {
    return repOF;
}

/**
 * @param repOF
 *            the repOF to set
 */
public void setRepOF(String repOF) {
    this.repOF = repOF;
}

/**
 * @return the idSwitch
 */
public String getIdSwitch() {
    return idSwitch;
}

/**
 * @param idSwitch
 *            the idSwitch to set
 */
public void setIdSwitch(String idSwitch) {
    this.idSwitch = idSwitch;
}

/**
 * @return the interfaceVersClient
 */
public String getInterfaceVersClient() {
    return interfaceVersClient;
}
```

```
package org.openflow.example.ofGmpls;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;
import java.net.URLConnection;

import thor.net.DefaultTelnetTerminalHandler;
import thor.net.TelnetURLConnection;

public class TelnetTools extends DefaultTelnetTerminalHandler {
    private OutputStream telnetEnvoieCmd;
    public static InputStream telnetRetourCmd;
    private URLConnection urlTelnetConnection;

    /**
     * @throws IOException
     *
     */
    public TelnetTools(String telnetHost, int telnetPort) throws IOException {
        super();
        connexionTelnet(telnetHost, telnetPort);
    }

    public void deconnexionTelnet() {
        try {
            telnetEnvoieCmd.close();
            telnetRetourCmd.close();
            ((TelnetURLConnection) urlTelnetConnection).disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * connection sur le poste telnet
     *
     * @param telnetHost
     * @param telnetPort
     * @throws IOException
     */
    public void connexionTelnet(String telnetHost, int telnetPort)
            throws IOException {
        if ((urlTelnetConnection != null)
                && (((TelnetURLConnection) urlTelnetConnection).connected())) {
            if (telnetEnvoieCmd != null) {
                telnetEnvoieCmd.close();
            }
            if (telnetRetourCmd != null) {
                telnetRetourCmd.close();
```

```
        }
    } else {
        urlTelnetConnection = getUrlConnexionTelnet(telnetHost, telnetPort);
    }
    telnetEnvoieCmd = urlTelnetConnection.getOutputStream();
    telnetRetourCmd = urlTelnetConnection.getInputStream();
}

private URLConnection getUrlConnexionTelnet(String telnetHost,
        int telnetPort) throws IOException {

    // voir la classe telne dans les sources ds la librairie
    URL url = new URL("telnet", telnetHost, telnetPort, "",
            new thor.net.URLStreamHandler());
    URLConnection urlTelnetConnection = url.openConnection();
    urlTelnetConnection.connect();
    return urlTelnetConnection;
}

public OutputStream getOutputStream(URLConnection urlTelnetConnection)
        throws IOException {
    if (urlTelnetConnection instanceof TelnetURLConnection) {
        ((TelnetURLConnection) urlTelnetConnection)
                .setTelnetTerminalHandler(this);
    }
    return urlTelnetConnection.getOutputStream();
}
public void envoyerCommande(String cmd) throws Exception {
    System.out.println(".");
    //System.out.println("[" + cmd + "]");
    getTelnetEnvoieCmd().write((cmd + "\r\n").getBytes());

    try {
        Thread.sleep(2000);
    } catch (InterruptedException e1) {
        e1.printStackTrace();
    }

}

/**
 * @return the telnetEnvoieCmd
 */
public OutputStream getTelnetEnvoieCmd() {
    return telnetEnvoieCmd;
}
```

```
    /**
     * @param telnetEnvoieCmd the telnetEnvoieCmd to set
     */
    public void setTelnetEnvoieCmd(OutputStream telnetEnvoieCmd) {
        this.telnetEnvoieCmd = telnetEnvoieCmd;
    }

    /**
     * @return the telnetRetourCmd
     */
    public static InputStream getTelnetRetourCmd() {
        return telnetRetourCmd;
    }

    /**
     * @param telnetRetourCmd the telnetRetourCmd to set
     */
    public static void setTelnetRetourCmd(InputStream telnetRetourCmd) {
        TelnetTools.telnetRetourCmd = telnetRetourCmd;
    }

    /**
     * @return the urlTelnetConnection
     */
    public URLConnection getUrlTelnetConnection() {
        return urlTelnetConnection;
    }

    /**
     * @param urlTelnetConnection the urlTelnetConnection to set
     */
    public void setUrlTelnetConnection(URLConnection urlTelnetConnection) {
        this.urlTelnetConnection = urlTelnetConnection;
    }
}
```

B.6 Code de restauration de LSP après rupture de lien

```
package org.openflow.example.ofGmpls;
//@autor Patrice KUEKEM TCHOUA

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

public class ServeurReprise {
    private TelnetTools telnet;

    public static void main(String args[]) throws Exception {
        ServeurReprise svr = new ServeurReprise();
        svr.runListener();
    }
    public void traiterPaquet(DatagramPacket receivePacket){
        //obtenir le type de du paquet
        System.out.println(receivePacket.getData());
        //si c'est un tiers on lance la procedure
        //ControleurOpenFlowGMPLS ctrl = ControleurOpenFlowGMPLS.getInstance();
        //ctrl.retablirLSP(0xac106265, 0xac106266);
    }
    public void runListener(){
        // connection au serveur telnet
        try {
            telnet = new TelnetTools(ControleurOpenFlowGMPLS.ipSrcTelnet, 23);
        } catch (Exception e) {
            System.err.println("Erreur lors de la connexion au serveur telnet");
            e.printStackTrace();
            return;
        }

        (new Thread(new Runnable() {

            @Override
            public void run() {
                System.err
                        .println("demarrage du process decoute des reponses de telnet");
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }

                try {
                    int position = 0;
                    int ch;
                    StringBuffer temp = new StringBuffer();
                    do {
                        try {
                            ch = telnet.getTelnetRetourCmd().read();// telnetRetourCmd.read();
                        } catch (IOException e) {
                            e.printStackTrace();
                            System.err
                                    .println("process d'ecoute encore non disponible");
```

```
    return;
}
if (ch < 0)
    return;
//System.out.print((char) ch);
//System.out.flush();
temp.append((char) ch);
if (temp.toString().contains(":") && (position == 0)) {
    temp = new StringBuffer();
    synchoniserCommande(position++);
} else if ((temp.toString().contains(":"))
        && (position == 1)) {
    temp = new StringBuffer();
    synchoniserCommande(position++);
} else if (temp.toString().contains("-S")&& (position == 2)) {
    temp = new StringBuffer();
    synchoniserCommande(position++);
}  else if (temp.toString().contains("mode")
        && temp.toString().contains(
                ControleurOpenFlowGMPLS.userTelnet)
        && (position == 3)) {
    temp = new StringBuffer();
    synchoniserCommande(position++);
} else if ((temp.toString()
        .contains("User Access Verification"))
        && (position == 4)) {//telnet.envoyerCommande(pwdDragon);
    temp = new StringBuffer();
    synchoniserCommande(position++);
} else if (temp.toString().contains("dragon>")) {
    boolean lstManquant=false;
    if (position==7){
        System.err
        .println(".....");
        //traiter
        if((!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspPrinc))&&
                (!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspBack))){
            System.err
            .print(" ");
        }else if((!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspPrinc))
                ||(!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspBack))){
            if (!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspPrinc)){
                lstManquant = true;
                ControleurOpenFlowGMPLS.nomLsp2 = ControleurOpenFlowGMPLS.nomLspBack;
                ControleurOpenFlowGMPLS.nomLsp = ControleurOpenFlowGMPLS.nomLspPrinc;
            }else if (!temp.toString().contains(ControleurOpenFlowGMPLS.nomLspBack)){
                lstManquant = true;
                ControleurOpenFlowGMPLS.nomLsp = ControleurOpenFlowGMPLS.nomLspBack;
                ControleurOpenFlowGMPLS.nomLsp2 = ControleurOpenFlowGMPLS.nomLspPrinc;
            }
            if (lstManquant){
                System.out.println(temp.toString());
                System.err
                .println("reactivation "+ControleurOpenFlowGMPLS.nomLsp2);
            }
        }
```

```
                    //temporiser
                     while (ControleurOpenFlowGMPLS.isOnGoing) {
                            try {
                                Thread.sleep(2000);
                            } catch (InterruptedException e1) {
                                e1.printStackTrace();
                            }
                        }
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e1) {
                        e1.printStackTrace();
                    }
                    //reinitiliser la position pour qu'il reste sur place
                    if (!lstManquant)
                    position=0;
                }else if(position ==13){
                    position=0;
                }
                temp = new StringBuffer();
                synchoniserCommande(position++);
            } else if (temp.toString().contains("1")) {
                temp = new StringBuffer();
                synchoniserCommande(position++);
            }
        } while (true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void synchoniserCommande(int i) {
    // se connecter
    try {
        switch (i) {
        case 0: {
            telnet.envoyerCommande(ControleurOpenFlowGMPLS.userTelnet);
            return;
        }
        case 1: {
            telnet.envoyerCommande(ControleurOpenFlowGMPLS.pwdTelnet);
            return;
        }
        case 2: {

            telnet.envoyerCommande("sudo -s");
            return;
        }
        case 3: {

            telnet.envoyerCommande(ControleurOpenFlowGMPLS.pwdTelnet);
            return;
        }
        case 4: {
            telnet.envoyerCommande("telnet localhost 2611");
            return;
        }
        case 5: {
            telnet.envoyerCommande(ControleurOpenFlowGMPLS.pwdDragon);
            return;
        }
        case 6: {
            telnet.envoyerCommande("show lsp");
            return;
        }
        case 7: {
            telnet.envoyerCommande("commit lsp " + ControleurOpenFlowGMPLS.nomLsp2);
            return;
        }
        case 8: {
            // 1 cnl_host2> edit lsp 5to3-A-
            telnet.envoyerCommande("delete lsp " + ControleurOpenFlowGMPLS.nomLsp);
            return;
        }
        case 9: {
            // 1 cnl_host2> edit lsp 5to3-A-
            telnet.envoyerCommande("edit lsp " + ControleurOpenFlowGMPLS.nomLsp);
            return;
        }
        case 10: {

            // 2 cnl_host2(edit-lsp-5to3-A-)# set source ip-address
            // 10.10.23.5
            // lsp-id 150 destination ipaddress 10.10.23.3 tunnel-id
            // 250
```

```
                telnet.envoyerCommande("set source ip-address "
                        + ControleurOpenFlowGMPLS.ipSrc + " lsp-id " + ControleurOpenFlowGMPLS.lspId
                        + " destination ip-address " + ControleurOpenFlowGMPLS.ipDest
                        + " tunnel-id " + ControleurOpenFlowGMPLS.tunnelId);
                return;
            }
            case 11: {

                // 3 cnl_host2(edit-lsp-5to3-A-)# set bandwidth gige
                // swcap l2sc
                // encoding
                // ethernet gmid ethernet
                telnet.envoyerCommande("set bandwidth gige swcap l2sc encoding ethernet gmid ethernet");
                return;
            }
            case 12: {

                // 4 cnl_host2(edit-lsp-5to3-A-)# exit
                telnet.envoyerCommande("exit");
                return;
            }

            default: {
                System.err
                        .println("Erreur lors de l'exécution des commandes sur le serveur telnet");
            }
            }
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
).start();
```

APPENDICE C

FICHIERS DES COMMANDES DE CONFIGURATION DES TUNNELS GRE

C.1 Contenu du fichier greScriptHost1

```
########################################
# Bash script to create the GRE tunnels
########################################
#!/bin/sh
#touch /var/lock/subsys/local

/sbin/modprobe ip_gre
/sbin/ip tunnel del gre1
/sbin/ip tunnel add gre1 mode gre remote 10.10.23.4 local 10.10.23.3 ttl 255
/sbin/ip link set gre1 up
/sbin/ip addr add 10.10.0.1/30 dev gre1
/sbin/ip route add 10.10.0.2 dev gre1

/sbin/ifconfig
```

C.2 Contenu du fichier greScriptVLSR1

```
########################################
# Bash script to create the GRE tunnels
########################################
#!/bin/sh
#touch /var/lock/subsys/local

sudo /sbin/modprobe ip_gre

sudo /sbin/ip tunnel del gre2
sudo /sbin/ip tunnel add gre2 mode gre remote 10.10.23.4 local 10.10.23.6 ttl 255
sudo /sbin/ip link set gre2 up
sudo /sbin/ip addr add 10.20.0.2/30 dev gre2
sudo /sbin/ip route add 10.20.0.1 dev gre2
```

```
sudo /sbin/ip tunnel del gre3
sudo /sbin/ip tunnel add gre3 mode gre remote 10.10.23.5 local 10.10.23.6 ttl 255
sudo /sbin/ip link set gre3 up
sudo /sbin/ip addr add 10.30.0.2/30 dev gre3
sudo /sbin/ip route add 10.30.0.1 dev gre3

sudo /sbin/ifconfig
```

C.3 Contenu du fichier greScriptVLSR2

```
########################################
# Bash script to create the GRE tunnels
########################################
#touch /var/lock/subsys/local

sudo /sbin/modprobe ip_gre

sudo /sbin/ip tunnel del gre1
sudo /sbin/ip tunnel add gre1 mode gre remote 10.10.23.3 local 10.10.23.4 ttl 255
sudo /sbin/ip link set gre1 up
sudo /sbin/ip addr add 10.10.0.2/30 dev gre1
sudo /sbin/ip route add 10.10.0.1 dev gre1

sudo /sbin/ip tunnel del gre2
sudo /sbin/ip tunnel add gre2 mode gre remote 10.10.23.6 local 10.10.23.4 ttl 255
sudo /sbin/ip link set gre2 up
sudo /sbin/ip addr add 10.20.0.2/30 dev gre2
sudo /sbin/ip route add 10.20.0.1 dev gre2

sudo /sbin/ifconfig
```

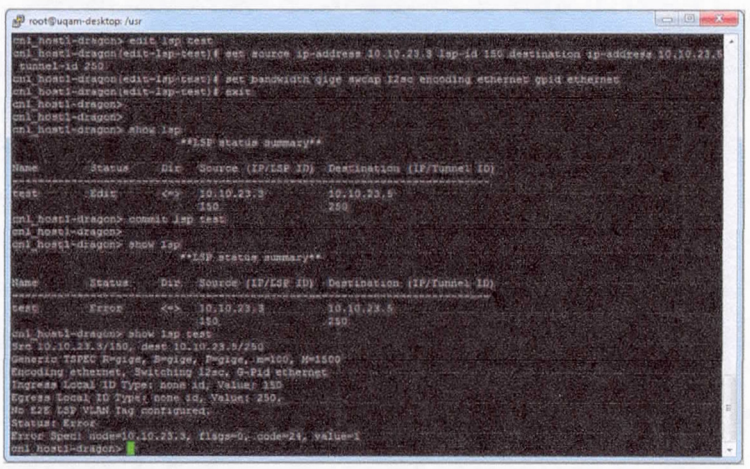
C.4 Contenu du fichier greScriptHost2

```
########################################
# Bash script to create the GRE tunnels
########################################
#touch /var/lock/subsys/local

sudo /sbin/modprobe ip_gre
sudo /sbin/ip tunnel del gre3
sudo /sbin/ip tunnel add gre3 mode gre remote 10.10.23.6 local 10.10.23.5 ttl 255
sudo /sbin/ip link set gre3 up
sudo /sbin/ip addr add 10.30.0.1/30 dev gre3
sudo /sbin/ip route add 10.30.0.2 dev gre3

sudo /sbin/ifconfig
```

APPENDICE D

RÉSOLUTION DES ERREURS

Erreur de création du LSP nommé « test » sur l'hôte 1



Solution :

Exécuter correctement les commandes ci-dessous :

1 cnl_host1-dragon> **edit lsp *test***

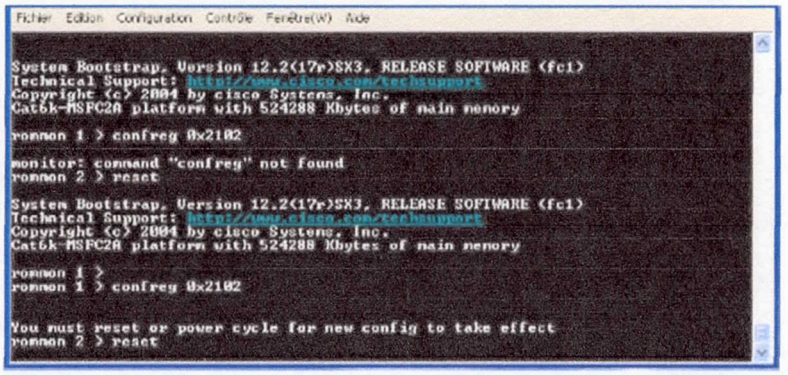
2 cnl_host1-dragon(edit-lsp-test)#**set source ip-address *10.10.23.5* lsp-id *150* destination ipaddress *10.10.23.3* tunnel-id *250***

3 cnl_host1-dragon(edit-lsp-test)**# set bandwidth gige swcap l2sc encoding ethernet gpid ethernet**

4 cnl_host1-dragon(edit-lsp-test)# **exit**

5 cnl_host1-dragon> **commit**

Erreur de configuration des Switch Cisco 6504

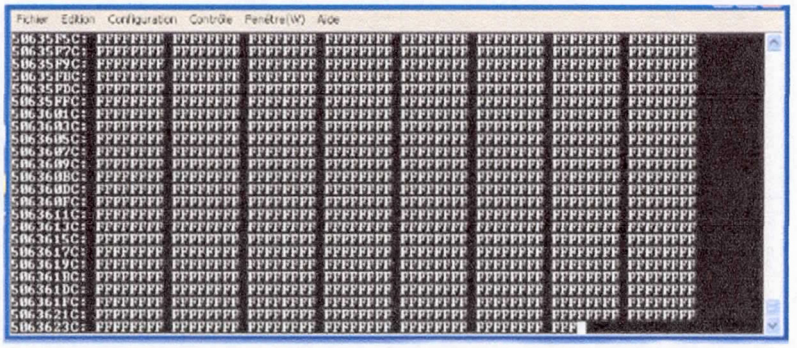


Solution :

Configurer l'interface Fa0/5 sur les deux Switch Cisco 6504 à l'aide des commandes suivantes :

*Swtich>**En***
*Swtich#**Configure terminal***
*Swtich(config)#**Int Fa0/5***
*Swtich(config-if)#**Swtichport mode access***
*Swtich(config-if)#**Swtichport access vlan 98***
*Swtich(config-if)#**Exit***
*Swtich#**wr***

Erreur de démarrage Switch Cisco 6504



Solution :

Éteindre le Switch, attendre environ 5 minutes et redémarrer

Erreur sur le VLSR2

Solution :

Assure la connexion physique du VLSR2 au réseau

Erreur 1 du démarrage du processus d'écoute

Solution :

Modifier le contenu du fichier « ParamActivGmpls.txt » pour obtenir les lignes ci-dessous :

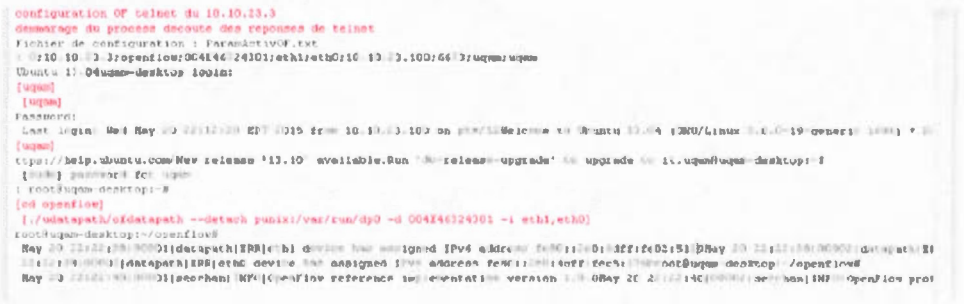
G;10.10.23.3;Desktop;greScriptHost1;uqam;uqam
G;10.10.23.5;Desktop;greScriptHost2;uqam;uqam
G;10.10.23.4;Bureau;greScriptVLSR;uqam;uqam
**G;10.10.23.6;Desktop;greScriptVLSR2;uqam;uqam*
V;10.10.23.3;/usr/local/dragon/bin;start-vlsr;uqam;uqam
V;10.10.23.5;/usr/local/dragon/bin;start-vlsr;uqam;uqam
V;10.10.23.4;/usr/local/dragon/bin;start-vlsr;uqam;uqam
**V;10.10.23.6;/usr/local/dragon/bin;start-vlsr;uqam;uqam*

Erreur 2 de démarrage du processus d'écoute

Solution :

Fermer le fichier de configuration « ParamActiveOF.txt » avant l'exécution du code Java :

Erreur 3 de démarrage du processus d'écoute

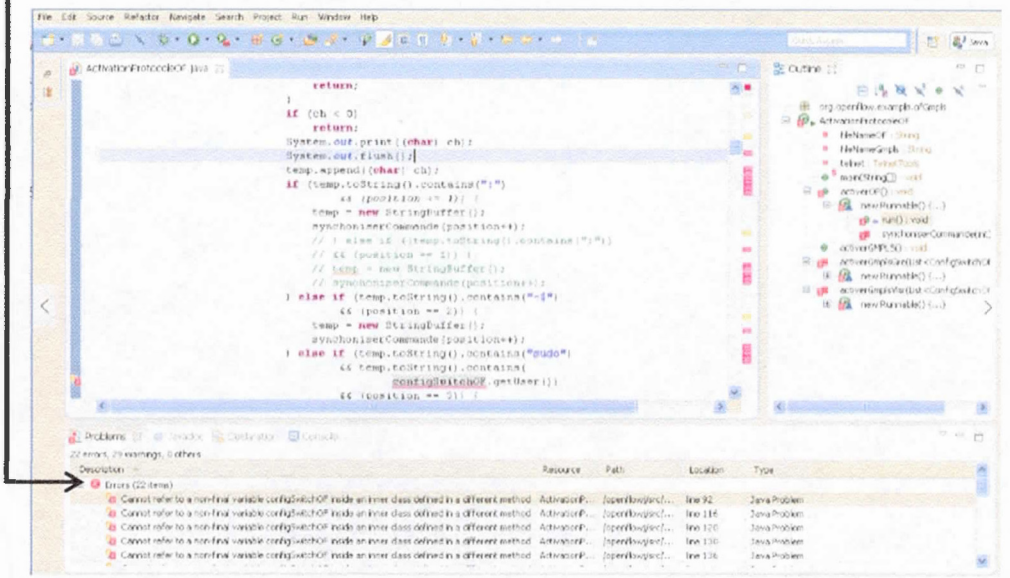


Solution :

Modifier le contenu du fichier « ParamActiveOF.txt » pour obtenir les lignes ci-dessous :

O;10.10.23.3;openflow;004E46324301;eth1;eth0;10.10.23.100;6633;uqam;uqam
O;10.10.23.5;openflow;004E46324302;eth1;eth0;10.10.23.102;6633;uqam;uqam

22 erreurs du test du code Java



Solutions :

Modifier les lignes de code concernées pour obtenir les lignes ci-dessous.

S1	import java.io.FileNotFoundException;
S2	import java.io.IOException;
S3	List<ConfigSwitchOF> lstParam = LectureFichierOF

S4	public TelnetTools(String telnetHost, int telnetPort) throws IOException {
	super();
	connexionTelnet(telnetHost, telnetPort);
	}

S5	urlTelnetConnection = getUrlConnexionTelnet(telnetHost, telnetPort);
S6	isOnGoing = true;
S7	(new Thread(new Runnable() {

S8	System.out
	.println("demmarage du process decoute des reponses de telnet");
	try {
	Thread.sleep(3000);
	} catch (InterruptedException e1) {

	e1.printStackTrace();
	}
	}

S9	if (temp.toString().contains(":")
	&& (position <= 1)) {
	temp = new StringBuffer();
	synchoniserCommande(position++);

S10	} else if (temp.toString().contains("#")&& (position == 4)) {
S11	+ configSwitch.getNumPort()+ ": connected")) {
S12	telnet.envoyerCommande(configSwitch.getUser());
S13	telnet.envoyerCommande("sudo -s");
S14	telnet.envoyerCommande("./udatapath/ofdatapath --detach punix:/var/run/dp0 -d "
S15	telnet.envoyerCommande("./secchan/ofprotocol unix:/var/run/dp0 tcp:"
S16	telnet.deconnexionTelnet(); isOnGoing = false;

S17	for (ConfigSwitchOF configSwitchOF : lstParam) {
	if (!configSwitchOF.getType().equalsIgnoreCase("G"))
	continue;

S18	while (isOnGoing) {
	try {
	Thread.sleep(2000);
	} catch (InterruptedException e1) {
	e1.printStackTrace();
	}
	}

S19	ch = telnet.getTelnetRetourCmd().read();
S20	else if (temp.toString().contains("sudo")
S21	for(int i=1;i<val.length;i++){ System.out.println("gre"+val[i]); }}
S22	if (position > 6) return;

BIBLIOGRAPHIE

(USC), University of Southern California, Information Sciences Institute (ISI), University of Maryland (UMD), Mid-Atlantic Crossroads (MAX) et George Mason University (GMU). 2008. «Virtual Label Switching Router Implementation Guide». vol. 2.1b.

Adami, Davide, Christian Callegari, Stefano Giordano, Fabio Mustacchio, Michele Pagano et F Vitucci. 2005. «Overview of the RSVP-TE Network Simulator: Design and Implementation». Proceedings of Internet Performance, Simulation, Monitoring and Measurements (IPS-MOME), p. 276-281.

Azodolmolky, S., R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou et D. Simeonidou. 2011. Integrated OpenFlow-GMPLS control plane: An overlay model for software defined packet over optical networks: Optical Communication (ECOC), 2011 37th European Conference and Exhibition on (18-22 Sept. 2011). 1-3 p.

Azodolmolky, Siamak, Philipp Wieder et Ramin Yahyapour. 2013. SDN-based cloud computing networking: Transparent Optical Networks (ICTON), 2013 15th International Conference on (23-27 June 2013). 1-4 p.

Bahnasy, Mahmoud, Karim Idoudi et Halima Elbiaze. 2015. «OpenFlow and GMPLS Unified Control Planes: Testbed Implementation and Comparative Study». Journal of Optical Communications and Networking, vol. 7, no 4, p. 301-313.

Berger, Lou. 2003. «Generalized multi-protocol label switching (GMPLS) signaling resource reservation protocol-traffic engineering (RSVP-TE) extensions».

Carela Español, Valentín. 2007. «Development of an SNMP agent for Ethernet switches».

Casellas, R., R. Martinez, R. Munoz, L. Liu, T. Tsuritani et I. Morita. 2013. An integrated stateful PCE / OpenFlow controller for the control and management of flexi-grid optical networks: Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013 (17-21 March 2013). 1-3 p.

Ceccarelli, Daniele, Diego Caviglia et Francesco Fondelli (2011). Message passing to assure deletion of label switched path, Google Patents

Channegowda, M, R Nejabati, M Rashidi Fard, S Peng, N Amaya, G Zervas, D Simeonidou, R Vilalta, R Casellas et R Martínez. 2013. «Experimental demonstration of an OpenFlow based software-defined optical network employing packet, fixed and

flexible DWDM grid technologies on an international multi-domain testbed». Optics express, vol. 21, no 5, p. 5487-5498.

Das, S., G. Parulkar, N. McKeown, P. Singh, D. Getachew et L. Ong. 2010. Packet and circuit network convergence with OpenFlow: Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC) (21-25 March 2010). 1-3 p.

Domancich, Sebastian, et Yannarak Wannasai. 2009. «PCE for GMPLS networks».

Foundation, Open Networking (2012). Software-Defined Networking: The New Norm for Networks. ONF White Paper: 12 p En ligne. <https://www.opennetworking.org/>.

Gringeri, S., N. Bitar et T. J. Xia. 2013. «Extending software defined network principles to include optical transport». Communications Magazine, IEEE, vol. 51, no 3, p. 32-40.

Gudla, V., S. Das, A. Shastri, G. Parulkar, N. McKeown, L. Kazovsky et S. Yamashita. 2010. Experimental demonstration of OpenFlow control of packet and circuit switches: Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC) (21-25 March 2010). 1-3 p.

Ji, P. N. 2012. Software defined optical network: Optical Communications and Networks (ICOCN), 2012 11th International Conference on (28-30 Nov. 2012). 1-4 p.

Jiayuan, Wang, Yan Ying et L. Dittmann. 2013. On the design of energy efficient optical networks with Software Defined Networking control across core and access networks: Optical Network Design and Modeling (ONDM), 2013 17th International Conference on (16-19 April 2013). 47-52 p.

Kompella, Kireeti, et Yakov Rekhter. 2005. «OSPF extensions in support of generalized multi-protocol label switching (GMPLS)».

Kumaki, Kenji. 2008. «Interworking Requirements to Support operation of MPLS-TE over GMPLS networks».

Lei, Liu, Zhang Dongxu, T. Tsuritani, R. Vilalta, R. Casellas, Hong Linfeng, I. Morita, Guo Hongxiang, Wu Jian, R. Martinez et R. Munoz. 2013. «Field Trial of an OpenFlow-Based Unified Control Plane for Multilayer Multigranularity Optical Switching Networks». Lightwave Technology, Journal of, vol. 31, no 4, p. 506-514.

Lei, Liu, T. Tsuritani et I. Morita. 2012a. Experimental demonstration of OpenFlow/GMPLS interworking control plane for IP/DWDM multi-layer optical networks: Transparent Optical Networks (ICTON), 2012 14th International Conference on (2-5 July 2012). 1-4 p.

--------. 2012b. From GMPLS to PCE/GMPLS to OpenFlow: How much benefit can we get from the technical evolution of control plane in optical networks?: Transparent Optical Networks (ICTON), 2012 14th International Conference on (2-5 July 2012). 1-4 p.

Martinez, R., R. Casellas, R. Munoz et R. Vilalta. 2011. Requirements and enhancements for the evolution of the GMPLS control plane of the ADRENALINE testbed to support

multi-layer (MPLS-TP/WSON) capabilities: Transparent Optical Networks (ICTON), 2011 13th International Conference on (26-30 June 2011). 1-4 p.

Meijerink, Mark, et Rob Prickaerts. 2006. «Generalized MPLS The DRAGON Project implementation at SARA». Universiteit van Amsterdam.

OF-Foundation. 2011. «OpenFlow Tutorial». En ligne. <http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial>. Consulté le 10/10/2014.

Papadimitriou, D., B. Berde, R. Martinez, J. G. Ordas, R. Theillaud et S. Verbrugge. 2006. Generalized Multi-Protocol Label Switching (GMPLS) Unified Control Plane Validation: Communications, 2006. ICC '06. IEEE International Conference on (June 2006). 2717-2724 p.

Patel, A. N., P. N. Ji et Wang Ting. 2013. QoS-aware optical burst switching in OpenFlow based Software-Defined Optical Networks: Optical Network Design and Modeling (ONDM), 2013 17th International Conference on (16-19 April 2013). 275-280 p.

Sezer, S., S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller et N. Rao. 2013. «Are we ready for SDN ? Implementation challenges for software-defined networks». Communications Magazine, IEEE, vol. 51, no 7.

Shiomoto, Kohei. 2008. «Framework for MPLS-TE to GMPLS migration».

Shirazipour, M., W. John, J. Kempf, H. Green et M. Tatipamula. 2012. Realizing packet-optical integration with SDN and OpenFlow 1.1 extensions: Communications (ICC), 2012 IEEE International Conference on (10-15 June 2012). 6633-6637 p.

Simeonidou, D., R. Nejabati et S. Azodolmolky. 2011. Enabling the future optical Internet with OpenFlow: A paradigm shift in providing intelligent optical network services: Transparent Optical Networks (ICTON), 2011 13th International Conference on (26-30 June 2011). 1-4 p.

Simeonidou, D., R. Nejabati et M. P. Channegowda. 2013. Software defined optical networks technology and infrastructure: Enabling software-defined optical network operations: Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013 (17-21 March 2013). 1-3 p.

GLOSSAIRE

Accès *root*	Niveau de connexion d'administrateur qui donne la possibilité de prendre le contrôle complet d'un système.
Adresse IP	Étiquette numérique attribuée à chaque appareil (par exemple : ordinateur, imprimante, camera, tablette, etc.) et participant à un réseau informatique qui utilise le protocole Internet pour la communication.
Backbones	Canaux de communication de transmission à très haut débit, qui relient les principaux nœuds du réseau.
C	Langage de programmation impératif (paradigme de programmation qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur), généraliste, conçu pour la programmation système.
Classes Java	Description de données Java appelées attributs, et d'opérations appelées méthodes.
Contrôleur OpenFlow	Serveur ayant les capacités d'accueillir différentes applications de gestion et de contrôle réseau pour gérer efficacement le réseau de manière centralisée ou distribuée.
Daemon	Processus ou ensemble de processus qui s'exécutent en arrière-plan plutôt que sous le contrôle direct d'un utilisateur.
DWDM	Technique utilisée pour les communications optiques à grande bande passante, permettant de faire passer dans une seule fibre optique plusieurs

	signaux de longueur d'onde différentes. Ces signaux sont mélangés à l'entrée à l'aide d'un multiplexeur (MUX), et séparés à la sortie à l'aide d'un démultiplexeur (DEMUX).
Eclipse	Environnement de développement (IDE) historiquement destiné au langage Java, même si grâce à un système de plugins il peut également être utilisé avec d'autres langages de programmation, dont le C/C++ et le PHP.
Éthane	Architecture de gestion de sécurité qui combine des commutateurs (basés sur des flux) à un contrôleur central de gestion d'admission et de routage des flux.
FastEthernet	Dénomination pour décrire une variété de technologies utilisées pour implémenter le standard Ethernet (Implémentation au niveau de la couche physique et de la sous-couche MAC) à des débits jusqu'à 100 Mbit/s.
FreeBSD	Système d'exploitation informatique souvent utilisé pour alimenter les serveurs, ordinateurs de bureau modernes et plateformes embarquées.
Gigabit Ethernet	Terme utilisé pour décrire une variété de technologies permettant de mettre en œuvre le standard Ethernet à des taux de transfert de données d'un gigabit par seconde (ou 1 000 mégabits par seconde).
GMPLS	Généralisation du protocole MPLS dans les domaines des réseaux informatiques et de télécommunications, et qui permet de posséder une structure de contrôle unique sur les trois premières couches du réseau.
GNU	Projet de système d'exploitation libre lancé en 1983 par Richard Stallman, puis maintenu par le projet GNU.
Hub	Station qui assure la coordination d'un groupe de stations ou de sous-réseaux, ainsi que leurs accès éventuels à d'autres réseaux.

Infonuagique	Modèle d'organisation informatique permettant l'accès à des ressources numériques dont le stockage est externalisé sur plusieurs serveurs.
Linux	Système d'exploitation libre fonctionnant avec le noyau Linux qui est une implémentation libre du système UNIX respectant les spécifications POSIX.
Masque de sous réseau	Succession de bits permettant de distinguer la partie de l'adresse utilisée pour le routage et celle utilisée pour numéroter des interfaces.
Mhz (mégahertz)	Multiple de l'hertz qui est l'unité de fréquence du système international.
Mo (mégaoctet)	Unité de capacité de la mémoire des ordinateurs.
Nœud	Équipement intermédiaire du réseau (routeur, commutateur, hub, etc.)
OpenFlow	Technologie basée sur des commutateurs Ethernet; une technologie habilitante pour les réseaux SDN (*Software-Defined Networking*). Selon la ONF (*Open Networking Foundation*), il permet l'accès direct et la manipulation du plan d'acheminement des périphériques réseaux tels que les commutateurs et les routeurs, qu'ils soient physiques ou virtuels (basés sur un hyperviseur).
OSPF	Protocole de routage qui permet au routeur de prendre des décisions fondées sur la charge de trafic, le débit, le coût des circuits et les priorités affectées aux trames. ®
Peer-to-peer	Technologie permettant l'échange direct de données entre deux ordinateurs reliés à Internet, sans passer par un serveur central. (On dit aussi poste à poste).
Pentium III	Microprocesseur de la gamme x86 d'Intel. Il est de la 6e génération.
Perl	Langage de programmation de haut niveau avec un héritage éclectique

	écrit par Larry Wall et un bon millier de développeurs.
Ping	Utilitaire de logiciel d'administration de réseau informatique utilisé pour tester l'accessibilité d'un hôte sur un réseau de protocole Internet (IP) et de mesurer le temps aller-retour pour les messages envoyés à partir de l'hôte d'origine vers un ordinateur de destination.
Protocole	Spécification d'un ensemble de règles permettant d'établir un type de communication particulier entre deux entités informatiques.
Proxy	Composant logiciel informatique qui joue le rôle d'intermédiaire entre deux équipements pour faciliter ou surveiller leurs échanges.
PuTTy	Programme permettant de se connecter à distance à des serveurs en utilisant les protocoles SSH, Telnet ou Rlogin.
Réseau déterministe	Réseau permettant à la fois des débits élevés et une garantie dans la promptitude.
Réseau DWDM	Réseau optique utilisé pour la transmission point à point à l'aide de la technique DWDM.
Réseau informatique	Ensemble d'ordinateurs ou de terminaux qui sont interconnectés, généralement de façon continue au moyen des voix des télécommunications.
RFC	Série numérotée de documents officiels décrivant les aspects techniques d'Internet, ou de différents matériels informatiques.
Sous-réseau	Ensemble constitué d'un ou plusieurs systèmes ouverts intermédiaires servant de relais au travers duquel des systèmes ouverts d'extrémité peuvent établir des connexions de réseau.
SSH	Programme informatique et protocole de communication sécurisé qui

	permet la connexion distante sécurisée.
Streaming	Mode de diffusion en continu; procédé permettant de diffuser un programme via Internet avant son téléchargement complet.
Technologie client-serveur	Mode de communication à travers un réseau entre plusieurs programmes ou logiciels : l'un qualifié de client qui envoie des requêtes et l'autre qualifié de serveur qui reçoit les requêtes des clients et y répondent.
Telnet	Protocole d'émulation de terminal TCP/IP qui permet aux utilisateurs d'un ordinateur hôte de se connecter aux ressources matérielles et logicielles d'un autre ordinateur via un réseau.
TTL	Mécanisme qui limite la durée de vie des données dans un ordinateur ou un réseau.
Tunneling	Encapsulation de données d'un protocole réseau dans un autre, situé dans la même couche du modèle en couches, ou dans une couche de niveau supérieur.
Unix	Système d'exploitation multitâche et multi-utilisateur qui repose sur un interpréteur ou superviseur (le shell).
Utilitaire	Logiciel conçu pour aider à gérer et à régler un équipement informatique.
Virtualisation	Phénomène qui consiste à faire fonctionner un ou plusieurs systèmes d'exploitation comme simple logiciel au-dessus d'un autre système d'exploitation.
VLAN	Réseau d'ordinateurs utilisant des inter-réseaux comme liaisons de données transparentes pour les utilisateurs, et qui n'impose pas de restrictions sur les protocoles, de sorte que ce réseau se comporte comme un réseau local. ®

Wireshark	Analyseur de paquets libre utilisé dans le dépannage et l'analyse des réseaux informatiques, le développement de protocoles, l'éducation et la rétro-ingénierie.
X forwarding	Option de connexion utilisée par l'émulateur puTTY qui permet à partir d'un ordinateur de lancer des applications graphiques sur un ordinateur distant.