UNIVERSITÉ DU QUÉBEC À MONTRÉAL

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

RAPPORT DE PROJET
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
KARIM IDOUDI

MAI 2014

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

IMPLEMENTATION OF A UNIFIED CONTROL PLANE FOR MULTILAYER NETWORKS IP/DWDM

MASTER THESIS
PRESENTED
AS A PARTIAL REQUIREMENT
FOR THE MASTER IN ELECTRICAL ENGINEERING

BY
KARIM IDOUDI

MAI 2014

REMERCIEMENTS

Je tiens à exprimer ma gratitude et mes respects les plus sincères à mon encadrante, Professeur Halima Elbiaze, pour l'aide qu'elle a bien voulu m'accorder tout au long des différentes étapes de mes travaux de recherche, moyennant ses critiques constructives, ses suggestions pertinentes, je m'incline par-devant ses qualités humaines et morales dont j'étais toujours impressionné, sa volonté permanente d'aider et ses idées innovantes qui m'ont permis d'élaborer mes travaux de recherches. Son attitude rigoureuse m'a doté d'une meilleure clairvoyance quant à mon avenir professionnel.

Je voudrais souligner la qualité des moyens informatiques et logistiques mis à la disposition des chercheurs du laboratoire «Optical Transport Network Laboratory» de l'UQAM et l'appui financier du CRSNG mis en œuvre par Professeur Halima Elbiaze.

D'autre part, je voudrais exprimer ma reconnaissance à tous ceux qui m'ont aidé durant mes travaux de maîtrise et à tous les enseignants tout au long de mon parcours académique.

Je remercie mes parents pour leur soutien tout au long de mes études, notamment ma mère, la grandiose Baya pour son amour et son engagement inconditionnel. J'espère qu'elle sera toujours fière de moi.

RÉSUMÉ

Actuellement, les couches IP et optiques sont exploitées séparément, sans interaction dynamique ; ce qui entraîne des coûts d'exploitation élevés, une faible efficacité du réseau, et une longue latence du provisionnement des chemins de bout en bout. Cette séparation est due essentiellement aux différences des mécanismes de contrôle pour ces deux domaines. Par conséquent, l'un des plus grands défis des opérateurs réseaux d'aujourd'hui est la réalisation d'un plan de contrôle unifié (UCP (Unified Control Plane)) pour gérer les deux réseaux simultanément .

Pour répondre à ce défi, nous proposons dans ce projet deux solutions pour implémenter un UCP qui permet d'assurer une interaction dynamique entre les réseaux optiques multi-couches IP/DWDM. La première solution est une extension du protocole OpenFlow, l'un des implémentations SDN (Software Defined Networking) la plus utilisée aujourd'hui. Dans ce rapport, nous présentons les différentes modifications réalisées et les modules implémentés afin que ce protocole réponde à nos besoins. La deuxième solution est basée sur l'utilisation du protocole GMPLS (Generalized Multi-Protocol Label Switching) dont l'implémentation reste complexe à cause du nombre important des équipements réseau qui ne sont pas encore compatibles avec ce protocole. Ce rapport montre l'adaptation de cette technologie avec nos équipements réseaux, situés dans notre laboratoire de recherche.

D'une part, la validation et la faisabilité des deux solutions implémentées ont été évalués en utilisant notre banc d'essai. D'autre part, les performances des deux solutions ont été comparées quantitativement en utilisant une simulation sur deux topologies de réseaux réelles.

Mots-clés : Réseaux IP/Optiques, OpenFlow, GMPLS, DRAGON.

ABSTRACT

Currently, IP and optical layers operate separately without dynamic interaction which leads to high operational cost, low network efficiency, and long processing latency for end-to end path provisioning. This separation is due primarily to the differences in control mechanisms for these two areas. Therefore, one of the biggest challenges facing network operators today is the realization of a unified control plane (UCP) to manage both networks simultaneously.

To meet this challenge, we propose in this project two solutions to implement a UCP which ensures a dynamic interaction between multilayer optical networks IP/ DWDM. The first solution is an extension of OpenFlow protocol, one of the widely used SDN (Software Defined Networking) implementations. In this report, we present the various changes made and implemented modules so that this protocol meets our needs. The second solution is based on the use of GMPLS (Generalized Multi-Protocol Label Switching) protocol whose implementation is complex due to the large number of network equipments that are not yet compatible with this protocol. This report explains the adaptation of this technology with our network equipments located in our research laboratory.

The validation and feasibility of two solutions implemented were evaluated using our testbed. On the other hand, the performance of the two solutions were compared quantitatively using a simulation on two real network topologies.

Keywords : IP/Optical Networks, OpenFlow, GMPLS, DRAGON.

TABLE DES MATIÈRES

RÉSUMÉ iv

ABSTRACT v

TABLE DES FIGURES ix

LISTE DES TABLEAUX xi

NOMENCLATURE xii

INTRODUCTION 1

CHAPITRE I
CADRE THÉORIQUE 5

1.1 Introduction 5

1.2 Réseaux IP et réseaux de transport 5

1.2.1 Réseaux IP 5

1.2.2 Réseaux de transport 7

1.3 OpenFlow 10

1.3.1 SDN 11

1.3.2 Concepts d'OpenFlow 12

1.3.3 Distinction entre le plan de contrôle et le plan de données 13

1.3.4 Composants d'un réseau OpenFlow 14

1.3.5 Switch OpenFlow 14

1.3.5.1 Tables OpenFlow 15

1.3.5.2 Actions d'OpenFlow 18

1.3.5.3 Canal sécurisé 18

1.3.6 Contrôleur OpenFlow 19

1.3.7 Protocole OpenFlow 19

1.3.7.1 Messages du protocole OpenFlow 19

1.3.7.2 Connexion entre le contrôleur et la switch 19

1.4 GMPLS 20

1.4.1 Origines 20

1.4.2 Opération de GMPLS 20
1.4.3 Routage GMPLS 21
1.4.3.1 OSPF-TE 21
1.4.4 Signalisation GMPLS 22
1.4.4.1 RSVP-TE 22
1.4.5 Conclusion 22

CHAPITRE II
PREMIÈRE SOLUTION : EXTENSION DU PROTOCOLE OPENFLOW 23

2.1 Introduction 23
2.2 État de l'art 23
2.3 Pourquoi OpenFlow ? 24
2.4 OpenFlow Optical Agent 25
2.5 Path Computation Element (PCE) 25
2.5.1 Executor 26
2.5.2 ONS Adapter 28
2.6 Expériences 28
2.6.1 Cadre empirique 28
2.6.2 Scénario A : Établissement d'un lightpath 28
2.6.3 Scénario B : Restauration du lightpath 30
2.7 Conclusion 31

CHAPITRE III
DEUXIÈME SOLUTION : GMPLS 32

3.1 Introduction 32
3.2 État de l'art 32
3.3 DRAGON 33
3.3.1 Introduction 33
3.3.2 Composants du plan de contrôle DRAGON 34
3.3.2.1 CSA (Client System Agent) 34
3.3.2.2 VLSR (Virtual Label Switch Router) 34
3.3.2.3 NARB (Network Aware Resource Broker) 35
3.4 Test du software DRAGON avec un commutateur Ethernet commercial 35

3.4.1 Scénario 1 : Infrastructure de base 35
3.4.2 Configuration logicielle requise 36
3.4.3 Installation 37
3.4.4 Configuration 37
3.4.5 Configuration du commutateur Cisco 38
3.4.6 Test 38
3.4.7 Signalisation GMPLS 40
3.5 Test du software DRAGON avec deux commutateurs Ethernet commerciaux 41
3.5.1 Scénario 2 : Infrastructure avancée (VLSR to VLSR) 41
3.5.2 Test 42
3.5.3 Signalisation GMPLS 42
3.6 Adaptation du software DRAGON pour la switch Cisco ONS 15454 43
3.6.1 Scenario 3 : Test du software DRAGON avec une switch Cisco ONS 15454 . . 44
3.6.2 Scenario 4 : Test du software DRAGON avec deux switches Cisco ONS 15454 45
3.7 Conclusion 47

CHAPITRE IV
COMPARAISON DES DEUX SOLUTIONS 48

4.1 Introduction 48
4.2 Résulats expérimentaux 48
4.3 Simulation 49
4.3.1 Introduction 49
4.3.2 La Topologie NSF (National Science Fondation) 50
4.3.3 La Topologie COST239 53
4.4 Conclusion 56

CHAPITRE V
CONCLUSION 57

CHAPITRE VI
PUBLICATIONS 59

6.1 Article accepté dans la conférence ICTON 59
6.2 Article accepté dans la conférence IEEE GLOBECOM 65

Appendice A : Scénario 1 et 3 73

Appendice B : Scénario 2 et 4 80

TABLE DES FIGURES

Figure		Page
1.1	Réseaux IP / Transport	6
1.2	Réseau IP (domaine intra)	8
1.3	Connectivité des routeurs IP dans un réseau de transport	9
1.4	Réseau de transport et ses clients	10
1.5	Architecture d'un réseau traditionnel vs Architecture d'un réseau SDN	12
1.6	Réseau OpenFlow	14
1.7	Entrées du "Flow Table"	16
1.8	Traitement des paquets dans un réseau OpenFlow	17
1.9	Entrées de "Group Table"	17
2.1	L'architecture unifiée d'un réseau de paquets-circuits convergent en utilisant OpenFlow	25
2.2	Extension du protocole OpenFlow	26
2.3	Path Computation Element (PCE)	27
2.4	Exemple de coloration de graph	27
2.5	La configuration du réseau et les messages échangés pour les scénarios A et B	29
2.6	Capture Wireshark : Scénario A	30
2.7	Capture Wireshark : Scénario B	31
3.1	Test du software DRAGON avec un commutateur Ethernet commercial	36
3.2	Message d'installation réussie de VLSR	37
3.3	LSP en mode « Edit »	39
3.4	LSP en mode « In service »	40
3.5	Création de quatre LSPs	40
3.6	Capture wireshark à partir du port eth0 du VLSR	41

3.7 Vue d'ensemble de l'infrastructure étendue (Scénario 2) 42

3.8 SNMP /TL1 Gateway 43

3.9 Messages échangés lors de la création d'un LSP 44

3.10 Test du software DRAGON avec une switch Cisco ONS 15454 45

3.11 Test du software DRAGON avec deux switches Cisco ONS 15454 46

4.1 L'algorithme exécuté au niveau du simulateur pour simuler le contrôleur OpenFlow 51

4.2 L'algorithme exécuté au niveau du simulateur pour simuler l'Agent OpenFlow 52

4.3 La topologie NSF 53

4.4 Topologie NSF : Temps d'établissement du lightpath (ms) vs la charge du réseau (Erlang) 53

4.5 Topologie NSF : Nombre de messages de contrôle vs la charge du réseau (Erlang) 53

4.6 Topologie NSF : Probabilité de blocage vs la charge du réseau (Erlang) 54

4.7 Topologie NSF : Nombre de noeuds par requête vs la charge du réseau (Erlang) 54

4.8 La topologie COST239 54

4.9 Topologie COST239 : Temps d'établissement du lightpath (ms) vs la charge du réseau (Erlang) 55

4.10 Topologie COST239 : Nombre de messages de contrôle vs la charge du réseau (Erlang) 55

4.11 Topologie COST239 : Probabilité de blocage vs la charge du réseau (Erlang) . 55

4.12 Topologie COST239 : Nombre de noeuds par requête vs la charge du réseau (Erlang) 55

LISTE DES TABLEAUX

Table Page

1.1 Champs du matching OpenFlow 15

4.1 Temps expérimental d'établissement des liens 49

4.2 Protocoles utilisés pour chaque solution 50

NOMENCLATURE

API	Application Programming Interface
AS	Autonomous System
CLI	Command Line Interface
CTC	Cisco Transport Controller
DRAGON	Dynamic Resource Allocation in GMPLS Optical Networks
DWDM	Dense Wavelength Division Multiplexing
E-BGP	External-Border Gateway Protocol
GMPLS	Generalized Multi-Protocol Label Switching
GRE	Generic Routing Encapsilation
I-BGP	Internal-Border Gateway Protocol
IP	Internet Protocol
LDP	Label Distribution Protocol
LSA	Link State Advertisement
LER	Label Edge Router
LSP	Label Switch Path
LSR	Label Switch Router
MPLS	Multi-Protocol Label Switching
NAT	Network Programming Translation
NSF	National Science Foundation
ONF	Open Networking Foundation
ONS	Optical Network Switch
OSPF	Open Shortest Path First
OTN	Optical Transport Network

PCE	Path Computation Element
QoS	Quality-of-Service
RSVP	Resource Reservation Protocol
ROADM	Reconfigurable Optical Add-Drop Multiplexer
SDN	Software Defined Networking
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical NETwork
TCP	Transmission Control Protocol
TDM	Time Devision Multiplexing
TE	Traffic Engineering
TED	Traffic Engineering Database
TL1	Transaction Language 1
TLV	Type Length Value
TTL	Time To Live
UCP	Unified Control Plane
WAN	Wide Area Network
WDM	Wavelength Division Multiplexing

INTRODUCTION

Motivations

Les réseaux à grande échelle sont coûteux à exploiter et les fournisseurs de services sont toujours à la recherche des moyens pour réduire leur capital et les coûts opérationnels. Une approche consiste à réduire le nombre de différents types de réseaux dont ils sont propriétaires. Ceci peut être accompli en faisant converger les différents services à un seul réseau.
En effet, les opérateurs réseaux d'aujourd'hui, gérant à la fois les domaines IP et optiques, doivent affronter plusieurs défis. Les couches IP et optiques sont exploitées séparément par deux équipes distinctes, 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 (Liu et al., 2012a). Cette séparation est due à la différence au niveau des équipements utilisés par chaque réseau, le type de commutation et surtout le manque d'un mécanisme de contrôle commun qui permet de gérer les deux technologies d'une façon simple et unifiée. Quelles que soient les raisons, une chose est claire - l'exploitation de deux réseaux avec deux mécanismes totalement différents est plus coûteux et inefficace que gérer un seul réseau convergent avec un plan de contrôle unifié (UCP).
Ce défi énorme nous interpelle, chercheurs et praticiens, et nous amène à nous poser la question suivante : comment pouvons-nous implémenter un plan de contrôle unique, dans un environnement d'essai réel, pour la gestion simultanée des réseaux optiques multicouches IP/DWDM ?

Contexte et contributions du mémoire

Les solutions proposées et les plus avancées à ce jour pour un UCP sont celles utilisant les protocoles GMPLS (Generalized Multi-Protocol Label Switching) et OpenFlow.
D'une part, en dépit d'énormes progrès et d'expérimentation sur OpenFlow, il demeure efficace comme plan de contrôle pour les réseaux convergents IP-circuits (Das et al., 2010; Gudla et al., 2010; Simeonidou et al., 2011). Toutefois, il convient de noter que ce protocole est mature pour les réseaux de commutation de paquets, mais reste à un stade de départ

pour les réseaux optiques. Pour cela, dans ce travail, nous proposons une extension du protocole OpenFlow pour supporter les réseaux optiques. Cette extension contient trois éléments essentiels. Premièrement, nous développons un agent que nous appelons « ***OpenFlow Optical Agent*** »[1] afin d'assurer la communication entre le contrôleur OpenFlow et les nœuds optiques. La fonction primordiale de cet agent est de transformer les messages du protocole OpenFlow à des commandes permettant de configurer les switches optiques, qui sont dans notre cas les commandes TL1 (Transaction Language 1) (Headquarters, 2003). Deuxièment, nous ajoutons un nouveau module au contrôleur OpenFlow basé sur un élément de calcul de chemin PCE (***Path Computation Element***). Un PCE est une composante fonctionnelle du plan de contrôle, capable d'effectuer un calcul de chemin sur un graphique représentant un réseau optique.

D'autre part malgré plus d'une décennie de développement et de travaux de standardisation sur GMPLS, il demeure efficace comme technique de plan de contrôle pour les réseaux optiques seulement. Mais comme UCP, il est trop complexe à mettre en œuvre dans les réseaux multicouches IP/DWDM, en raison de la nature distribuée, du nombre de protocoles et les interactions entre les différentes couches (Kumaki, 2008; Shiomoto, 2008). En plus, il y'a encore beaucoup d'équipements réseaux qui ne sont pas compatibles avec GMPLS. Par conséquent, il n'y a pas de déploiements commerciaux de base UCP-GMPLS à ce jour, et le débat pour sa praticabilité dans un vrai scénario opérationnel croît en intensité (Farrel, 2010). ***DRAGON*** (Dynamic Resource Allocation in GMPLS Optical Networks) (Lehman et al., 2006) est une solution software open source permettent de résoudre ce problème dans les réseaux Ethernet en utilisant le protocole SNMP (Simple Network Management Protocol) pour rendre les commutateurs Ethernet capables de fonctionner dans un réseau GMPLS. Notre travail consiste d'abord à déployer cette solution dans notre banc d'essai. Ensuite, nous implémentons un ***SNMP/TL1 Gateway*** pour adapter la solution avec nos switches optiques. Nous testons également les deux solutions proposées dans un banc d'essai du réseau réel et nous comparons leurs performances en utilisant une simulation sur deux topologies de réseaux réelles.

Structure du mémoire

1. Une partie de l'agent OpenFlow a été dévelopée en collaboration avec l'équipe de recherche de notre laboratoire (OTN Lab) de l'UQAM.

Afin de communiquer efficacement le travail accompli ainsi que les contributions de ce mémoire, nous avons structuré ce document comme suit :

- Le chapitre 1 présente un aperçu théorique sur les différents composants de notre projet. Ce chapitre commence par donner une vision globale sur les réseaux IP et de transport, en citant leurs caractéristiques et en précisant la différence entre ces deux domaines. Ensuite, nous décrivons les protocoles OpenFlow et GMPLS, les éléments de base de nos solutions.
- Le chapitre 2 présente la première solution implémentée (Extension du protocole OpenFlow). Nous expliquons d'abord l'avantage apporté par OpenFlow afin de résoudre notre problème. Ensuite, nous détaillons les deux modules développés (OpenFlow Optical Agent et PCE). Enfin, nous présentons les scénarios expérimentaux réalisés afin de valider cette solution. Le premier scénario consiste à établir un lightpath entre deux clients en passant par les deux réseaux IP et optiques. Le deuxième scénario concerne la restauration du lien lors de la résiliation du premier lightpath.
- Le chapitre 3 décrit notre deuxième solution basée sur le protocole GMPLS. Dans la première partie de ce chapitre, nous présentons le software DRAGON que nous avons utilisé pour le déploiement du GMPLS dans notre infrastructure réseau. Nous détaillons les différentes étapes d'installation et de configuration pour mettre en place ce software. En plus, nous présentons deux cas d'utilisations avec une switch Cisco commerciale. Afin d'adapter le software DRAGON avec nos switches optiques, nous implémentons un SNMP/TL1 Gateway. Dans la deuxième partie de ce chapitre, nous détaillons cette implémentation, les différents modules développés et nous présentons les deux scénarios réalisés afin de valider notre solution.
- Le chapitre 4 concerne la comparaison des performances des deux solutions implémentées en termes de temps d'établissement de liens, le trafic de contrôle et la probabilité de blocage. Pour atteindre cet objectif, nous présentons tout d'abord les résultats expérimentaux. Ensuite, nous détaillons la simulation que nous avons réalisée sur deux topologies de réseaux réelles. Les résultats de simulation seront discutés à la fin de ce chapitre.
- Le chapitre 5 conclut ce mémoire avec un résumé des travaux effectués ainsi que les résultats obtenus. Nous identifions également les travaux futurs possibles.

- Le chapitre 6 présente les deux articles scientifiques que nous avons publiés pour décrire notre travail de recherche.

CHAPITRE I

CADRE THÉORIQUE

1.1 Introduction

Dans ce chapitre, nous présentons les aspects théoriques liés à notre projet. Nous commençons par nos domaines d'études ; les réseaux IP et les réseaux de transports. Nous détaillons les différents éléments de chaque réseau, le mode d'opérabilité et de contrôle, et la différence entre ces deux domaines. Ensuite, nous introduisons les deux protocoles OpenFlow et GMPLS que nous avons utilisés afin de définir un plan de contrôle unifié pour les deux domaines étudiés.

1.2 Réseaux IP et réseaux de transport

Les réseaux IP sont à commutation de paquets, c'est-à-dire que les paquets sont commutés individuellement de la source à la destination en passant par les différents routeurs IP. Cependant, les paquets sont transportés physiquement entre les routeurs d'un réseau basé sur les fibres optiques et les commutateurs de circuits (Figure 1.1). Ce domaine est appelé réseau de transport. Dans les sections suivantes, nous détaillons les deux différents types de réseaux.

1.2.1 Réseaux IP

L'internet est une collection de réseaux IP interconnectés. Les réseaux constituants l'internet ont des propriétaires et des équipes de maintenance et de gestion indépendantes. Donc,

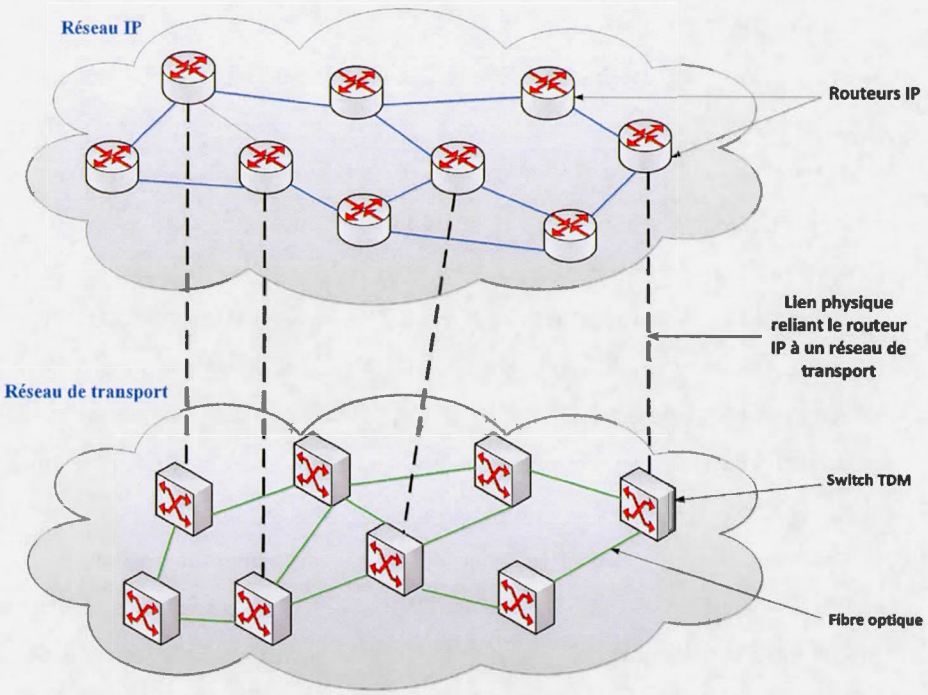


FIGURE 1.1: Réseaux IP / Transport

pour assurer une connectivité mondiale, ces réseaux utilisent le protocole E-BGP (External Border Gateway Protocol) pour annoncer l'accessibilité des adresses IP et choisir des chemins à travers des domaines de routage connus par les systèmes autonomes (AS). Dans notre travail, nous intéressons à l'architecture des réseaux IP au sein d'un AS dans un réseau étendu WAN (Wide Area Network). Les caractéristiques de ces domaines peuvent être résumées comme suit :

- Les réseaux IP ont des mécanismes de contrôle automatisés et entièrement distribués. Ces mécanismes impliquent des protocoles de routages (I-BGP, OSPF etc.) et dans certains cas des protocoles de signalisations (LDP, RSVP, etc.) (Figure 1.2). Dans les réseaux traditionnels, un routeur IP est à la fois l'élément de contrôle, qui prend les décisions de routage, ainsi que l'élément responsable de la transmission des données. Après la configuration d'un routeur IP (soit manuellement, soit en utilisant des scripts), il détecte automatiquement ses voisins et la topologie du réseau, échange les informations de routage et transmet les paquets. D'où l'aspect automatisé du contrôle au niveau des

réseaux IP.

- Les services dans les réseaux IP ont également tendance à avoir des implémentations réparties et interagissent de façon subtile avec les mécanismes de contrôle entièrement distribués (Figure 1.2). Ces interactions et la nature distribuée de leurs implémentations rendent ces services, offerts par un routeur IP, non standard (et par conséquent non opérables avec d'autres implémentations), même si les mécanismes de contrôle sont standardisés. À titre d'exemple, considérons la fonction d'ingénierie du trafic qui est assuré aujourd'hui par le protocole MPLS-TE (MultiProtocol Label Switching - Traffic Engineering). Elle dépend du plan de contrôle IP / MPLS qui comprend des protocoles normalisés. Mais la fonction elle-même reste non standardisée. L'ingénierie du trafic sur les routeurs **Cisco** ne fonctionne pas sur les routeurs de **Juniper** ou **Huawei**. Et donc, un réseau IP appartient généralement à un même fournisseur.
- Les fonctions de gestion des réseaux IP impliquent la configuration (généralement via une interface de ligne de commande (CLI), le suivi (généralement via SNMP) et l'entretien périodique. Les réseaux IP sont donc difficiles à gérer. Pour remédier à ceci, des équipes de personnel hautement qualifié gèrent et modifient manuellement ces réseaux, dans le but de parvenir à un équilibre entre les objectifs locaux de chaque fournisseur et le maintien de la connectivité mondiale.

Pour résumé, l'internet offre aujourd'hui un service de commutation de paquets « best effort » de bout en bout. Cependant, il est constitué d'un ensemble de réseaux IP intradomaine, qui sont surprovisionnés afin d'assurer un rendement acceptable et respecter les différentes classes de services ; ils sont automatisés, ils ont des mécanismes de contrôle entièrement distribués, ils ont des services distribués, ce qui les rend généralement des réseaux monofournisseurs et difficile à gérer.

1.2.2 Réseaux de transport

L'objectif fondamental d'un réseau de transport est de fournir une bande passante de communication d'un endroit géographique à un autre. Par exemple, le lien entre deux routeurs dans un réseau IP intradomaine étendu est logique. Il peut être établi avec un circuit (division temporelle ou longueur d'onde) dans le réseau de transport (Figure 1.3). Dans ce

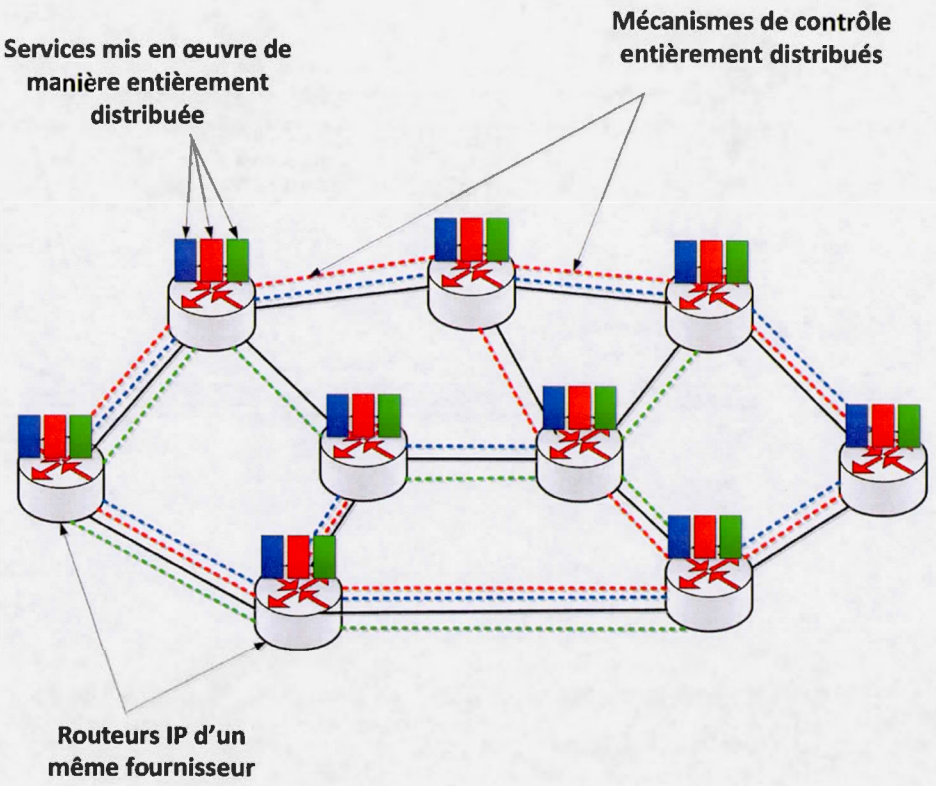


FIGURE 1.2: Réseau IP (domaine intra)

cas, le réseau IP est considéré comme un client sur le réseau de transport. Aujourd'hui, les réseaux de transport prennent en charge plusieurs « réseaux de clients » : les réseaux IP, le réseau téléphonique commuté public (RTCP), le réseau cellulaire, les réseaux point à point et les réseaux privés d'entreprises (Figure 1.4). Le réseau de transport comprend des fibres optiques avec de nombreux canaux de longueurs d'onde indépendantes qui se terminent par des systèmes de ligne WDM (Wavelength Division Multiplexing). Les mêmes longueurs d'onde dans les systèmes de ligne adjacents peuvent être réassemblées pour former un circuit de longueur d'onde (ou lightpath) par l'intermédiaire de câbles physiques ou une switch WDM. Chaque canal de longueur d'onde fonctionne à 2,5, 10 ou 40 Gbps. Puisque les canaux de longueurs d'onde fonctionnent à des taux élevés, le fournisseur de services de transport veut souvent les subdiviser pour fournir des connexions de granularité sous-longueurs d'onde à ses

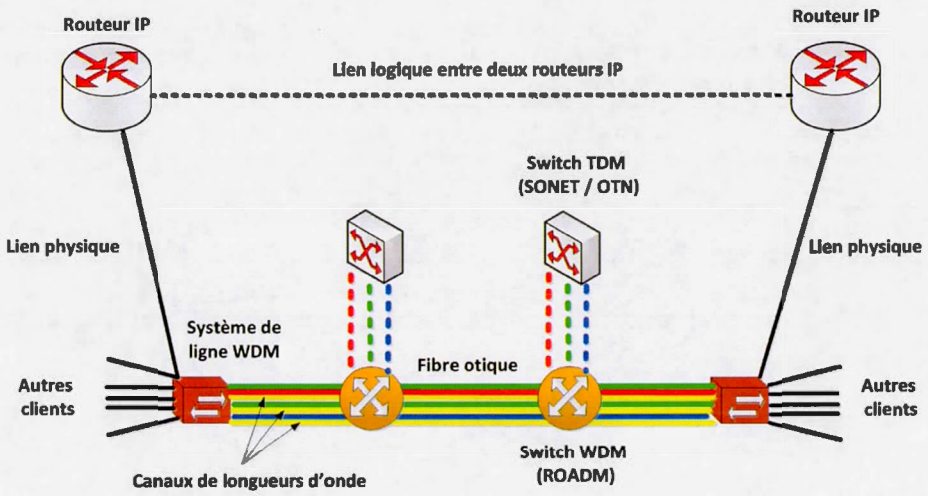


FIGURE 1.3: Connectivité des routeurs IP dans un réseau de transport

clients. Cette granularité est fournie par des switches TDM (Time Division Multiplexing). Nous faisons les observations suivantes sur les réseaux de transport :

- Contrairement aux réseaux IP, les réseaux de transport sont toujours intradomaine (intra-AS), c'est-à-dire qu'il n'existe pas d'équivalent à l'intersection interdomaine de l'Internet. Pour cela, le réseau de transport décrit les partitions comme des «domaines». La diversité des matériels provenant de différents fournisseurs, avec des procédures de gestion et de contrôle différentes, force le réseau de transport à être partitionné en îlots de vendeurs spécifiques. Ainsi, tant que le réseau de transport est multifournisseur (contrairement aux réseaux IP), il n'est pas automatisé. En fait, les réseaux de transport sont bien gérés, où une hiérarchie des éléments et systèmes de gestion de réseau avec les systèmes de soutien opérationnel effectue toutes les tâches de contrôle et de gestion. En général, ces systèmes ne sont pas des interfaces de programmation, mais plutôt des interfaces graphiques appartenant à des fournisseurs exclusifs qui sont gérées manuellement par les équipes des opérateurs de réseaux spécialisés.
- Fournir un service dans un réseau de transport est un processus manuel long. Considérons un service pris en charge par des circuits dans le plan de données : d'abord, le chemin est prévu par les différents groupes de planification en passant d'une ville

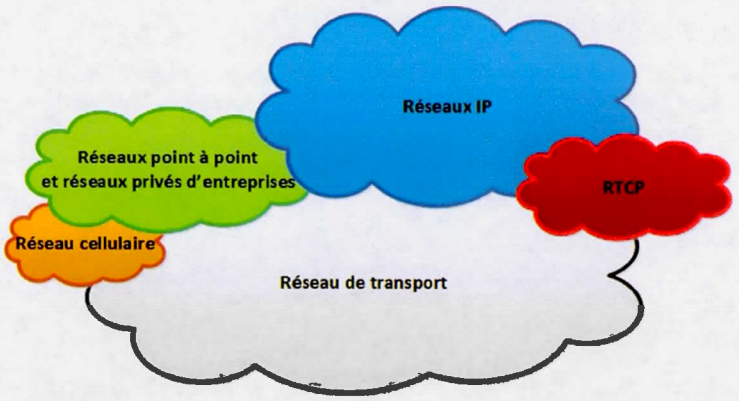


FIGURE 1.4: Réseau de transport et ses clients

à une autre. Ensuite, les différents groupes de provisionnement exécutent le plan en configurant manuellement les éléments de réseau de transport et leurs systèmes de gestion correspondants le long du chemin. Enfin, les équipes de test vérifient que le service fonctionne réellement et que les bases de données de gestion associées sont mises à jour. C'est pour cela que c'est trivial de voir pourquoi un tel service prend des semaines, parfois des mois, et pourquoi ces circuits sont statiques et restent en place pendant des mois et des années.

- Traditionnellement, les réseaux de transport ont manqué un plan de contrôle distribué, mais ont toujours eu une séparation nette entre le plan de contrôle et le plan de données. En outre, le plan de contrôle automatisé est généralement déclenché manuellement.
- Un réseau de transport fournit toujours des garanties dures dans les contrats SLA (Service Level Agreement), généralement en termes de « gros tuyaux » de la bande passante à haute disponibilité, le délai et la gigue.

1.3 OpenFlow

L'ONF (Open Networking Foundation) (ONF, 2014), une organisation dédiée à la fondation et l'adoption de logiciels des réseaux SDN (Software Defiend Networking), gère le standard OpenFlow. L'ONF définit OpenFlow comme la première interface de communica-

tion standard qui est définie entre les couches de contrôle et de données d'une architecture SDN. Dans la suite, nous présentons le concept des réseaux SDN et le protocole OpenFlow en détail.

1.3.1 SDN

Le terme SDN (réseau défini par logiciel) a été inventé au cours des récentes années. Le concept derrière SDN est 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 unique dans les réseaux en général, et les réseaux optiques multicouches IP/DWDM en particulier, nous ont montré que l'architecture SDN reste la plus prometteuse (Channegowda et al., 2013b; Gringeri et al., 2013; Ji, 2012; Patel et al., 2013; Wang et al., 2013).

SDN, fourni une nouvelle architecture réseau dynamique qui transforme les dorsales des réseaux traditionnels en plateforme de fourniture de services riches. En découplant les plans de contrôle et de données de réseau, l'architecture basée sur SDN isole l'infrastructure sous-jacente des applications qui l'utilisent. Cela rend l'infrastructure réseau programmable et facile à gérer à l'échelle. L'adoption de SDN peut améliorer la facilité de gestion de réseau, son évolutivité et son dynamisme dans le centre de données d'entreprise (Azodolmolky et al., 2013). Aussi, SDN permet aux opérateurs de réseau de gérer de façon flexible les routeurs et commutateurs à l'aide de logiciels s'exécutant sur des serveurs externes. Ils peuvent en plus exploiter cette architecture pour offrir efficacement des services innovants, en améliorant la qualité de service et l'expérience des utilisateurs finaux. S'il est appliqué convenablement, il peut aussi entraîner des économies CAPEX (coûts d'investissement), et OPEX (les coûts récurrents) pour les opérateurs à long terme. Dans les réseaux de paquets, la nécessité d'adopter SDN vient principalement des nouvelles exigences réseaux imposées par de nouveaux services (mobiles haut débit, contenus vidéo déployés dans les réseaux d'accès, les services d'infonuagique et de centre de données déployés à travers les réseaux) (Shirazipour et al., 2012).

Dans l'architecture des réseaux traditionnels d'aujourd'hui, chaque nœud réseau comprend un plan de données, et un OS (Operating System) comprenant lui aussi un plan de contrôle et une couche application. Dans l'architecture SDN, le plan de contrôle et la couche application

sont extraits et constituent une couche qui se situe hors de l'OS. La figure 1.5 illustre bien en (a) l'architecture d'un réseau traditionnel, et en (b) l'architecture d'un réseau SDN (Sezer et al., 2013). Contrairement à l'architecture traditionnelle (Figure 1.5 (a)), celle des réseaux SDN (Figure 1.5 (b)) offrent une architecture où la couche infrastructure (de données) est bien séparée des couches contrôle et application. La communication entre ces couches se fait via un plan de contrôle (Contrôleur), qui les relie à l'aide d'API (Application Programming Interface).

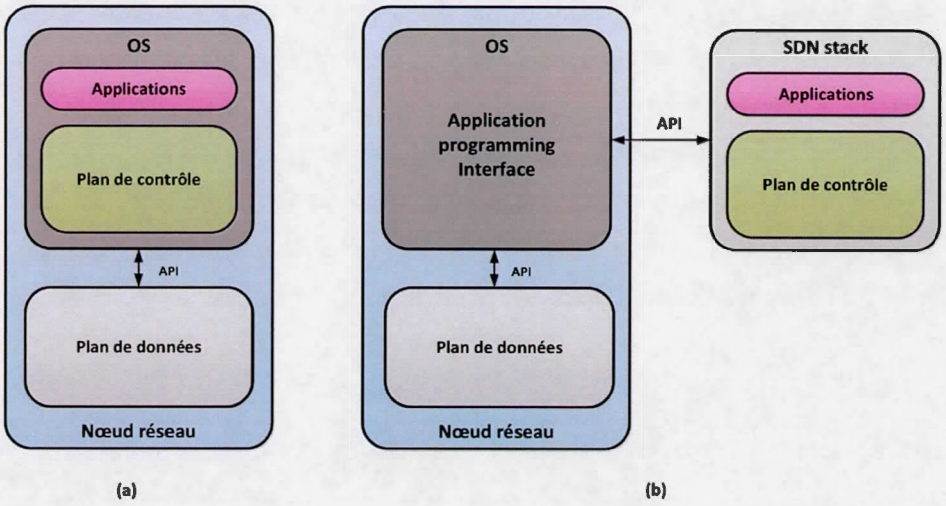


FIGURE 1.5: Architecture d'un réseau traditionnel vs Architecture d'un réseau SDN

1.3.2 Concepts d'OpenFlow

OpenFlow est le premier protocole de communication standard qui est définit entre les couches de contrôle et de données d'une architecture SDN. Il s'agit d'une nouvelle approche réseau basée sur des commutateurs Ethernet, qui met en place un vaste champ d'expérimentation sur le réseau réel sans avoir à en perturber le fonctionnement ni en dépenser des coûts supplémentaires pour d'autres types d'infrastructures. Ceci permet de faire de l'ingénierie de trafic dans le monde réel sans avoir à faire appel à des outils pour la simulation du réseau et du trafic et essayer de nouveaux protocoles de routage, de nouveaux modèles de sécurité,

etc...

L'idée de base est la suivante : La technologie exploite le fait que la plupart des switches et routeurs Ethernet modernes contiennent des tables de flux qui fonctionnent à haut débit pour implémenter des firewalls, NAT, QoS et pour collecter des statistiques. Ces tables de flux sont propres à chaque constructeur, mais on peut en identifier un ensemble de fonctionnalités qui sont implémentées sur la plupart des routeurs et des switches.

Le déploiement d'OpenFlow a commencé dans les universités. L'université de Stanford était la première à développer une méthode de déploiement. Plus tard, cette méthode a été aussi déployée dans plusieurs autres universités. Grâce à ce déploiement large, toute la communauté de chercheurs dans ces universités pourrait alors bénéficier d'une infrastructure réseau OpenFlow pour expérimenter. Actuellement l'université de Stanford collabore avec plusieurs fournisseurs d'équipements afin d'intégrer la fonctionnalité OpenFlow dans les routeurs, les commutateurs et les points d'accès. Cette intégration ne requiert aucun changement au niveau matériel, mais uniquement l'implémentation d'une table de flux.

1.3.3 Distinction entre le plan de contrôle et le plan de données

Dans les réseaux traditionnels, les décisions de traitement des paquets et la commutation des paquets sont dépendantes et résident dans le même équipement qui est la switch ou le routeur. L'approche OpenFlow permet de séparer ces deux plans dans deux équipements de nature différente : le plan de données (responsable de la commutation des paquets) et le plan de contrôle (responsable d'associer une décision aux paquets). Le plan de données doit être implémenté dans les switches OpenFlow, ce qui allègera les switches qui n'auront pour responsabilité que la commutation des paquets. Le plan de contrôle est assuré par une machine séparée, le contrôleur, qui réalise les traitements nécessaires pour déterminer les décisions à appliquer sur un flux de données. Cette séparation entre le plan de contrôle et celui des données permettra d'atteindre du très haut débit. Un autre avantage de cette séparation est que les composants de l'équipement responsable du plan de données seront optimisés pour la commutation des paquets ce qui évite d'alourdir le chemin de traitement avec des fonctionnalités qui relèvent du plan de contrôle.

1.3.4 Composants d'un réseau OpenFlow

La figure 1.6 détaille les éléments d'un réseau OpenFlow. Le réseau est formé essentiellement d'une ou plusieurs switches OpenFlow, un ou plusieurs contrôleurs et finalement le protocole OpenFlow définissant tous les messages échangés entre les contrôleurs et les switches et permettant de standardiser la communication. Les différents éléments du réseau OpenFlow seront détaillés dans les parties suivantes.

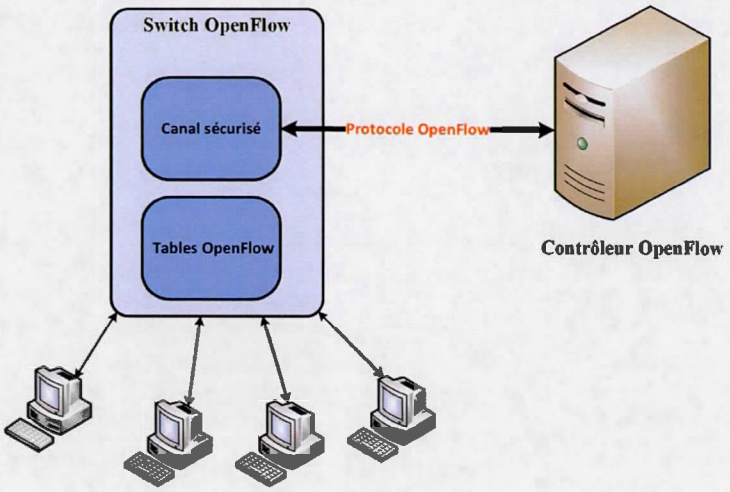


FIGURE 1.6: Réseau OpenFlow

1.3.5 Switch OpenFlow

Les éléments de base du switch OpenFlow sont les tables OpenFlow. Ces tables vont assurer la classification des paquets en se basant sur plusieurs champs des entêtes niveau 2, 3 ou 4 (La Table 1.1 illustre les champs pour la version 1.2 du protocole OpenFlow). On distingue deux types de switch OpenFlow :

- **Switch OpenFlow-only :** Supporte uniquement les actions requises pour le fonctionnement du protocole OpenFlow.
- **Switch OpenFlow-enabled :** En plus des actions requises pour le protocole OpenFlow, cette switch supporte les actions d'une switch ordinaire.

TABLE 1.1: Champs du matching OpenFlow

Champ	Bits	Application
Port d'entrée	32	tous les paquets
Adresse MAC source	48	tous les paquets sur les ports activés
Adresse MAC destination	48	tous les paquets sur les ports activés
Type Ethernet	16	tous les paquets sur les ports activés
ID VLAN	12	tous les paquets avec un tag VLAN
Priorité VLAN	3	tous les paquets avec un tag VLAN
Étiquette MPLS	20	tous les paquets avec un tag MPLS
Classe de trafic MPLS	3	tous les paquets avec un tag MPLS
Adresse IPv4 source	32	tous les paquets IPv4 and ARP
Adresse IPv4 destination	32	tous les paquets IPv4 and ARP
Protocole IPv4/Opcode ARP	8	tous les paquets IPv4 and IPv6 sur Ethernet, ARP
ToS bits IPv4	6	tous les paquets IPv4
Port de transport source/Type ICMP	16	tous les paquets TCP, UDP, SCTP and ICMP
Port de transport destination/Code ICMP	16	tous les paquets TCP, UDP, SCTP and ICMP

1.3.5.1 Tables OpenFlow

Cette section décrit la structure de la table de flux (Flow Table) et celle de la table de groupe (Group Table).

Table de flux :

Un "Flow Table" est composé de plusieurs entrées de flux, chacune est structurée comme indiqué dans la figure 1.7 :

- *Match fields* : Utilisés lors de la recherche de l'entrée correspondante au paquet. Ils sont constitués essentiellement des entêtes des paquets et des ports d'entrées. L'ensemble des champs du match field est listé dans la table 1.1.
- *Counters* : Servent essentiellement à garder des statistiques sur les flux pour ensuite décider si une entrée de flux est active ou non. Pour chaque table, chaque flux, chaque port, chaque file d'attente et chaque groupe de tables, des compteurs de statistiques sont maintenus.
- *Instructions* : Représentent l'ensemble des instructions OpenFlow qui servent à modifier le traitement (pipeline) que va subir le paquet. Les instructions supportées sont :
  1. Apply-Actions : Pour appliquer les actions sur le paquet immédiatement.
  2. Clear-Actions : Pour supprimer une liste des actions du paquet.
  3. Write-Actions : Ajouter une liste d'actions au paquet.
  4. Write-Metadata : Ajouter des données utiles pour le séquencement entre les tables OpenFlow.
  5. Goto-Table : Indique que le paquet doit être acheminé vers une table d'indice supérieur.

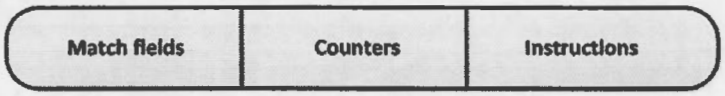


FIGURE 1.7: Entrées du "Flow Table"

Un flux de paquets à l'entrée de la switch va être traité par une succession de tables (qui est de 2 types : une qui ne contient que le traitement d'OpenFlow, et une autre (hybride) qui ajoute à ce traitement le traitement normal de la switch pour le flux de production par exemple). Ce traitement en pipeline spécifiera le comportement des paquets vis-à-vis de l'ensemble des "Flow Tables". Les instructions d'une entrée de flux peuvent explicitement diriger le paquet vers un autre "Flow Table" (via l'instruction "Goto"). Si l'entrée de flux ne dirige pas le paquet vers une autre table, le "pipeline processing" s'arrête à cette table. Dans ce cas le paquet subira les actions accumulées qui lui sont associées durant le pipeline. Si le paquet ne correspond à aucune entrée de flux, le comportement du switch vis-à-vis de ce paquet dépendra de la configuration de la table ; le comportement par défaut c'est d'envoyer

le paquet au contrôleur avec un message *OFPT_PACKET_IN*. Une autre option consiste à supprimer le paquet. La table peut aussi spécifier que dans ce cas le paquet doit continuer son chemin dans le pipeline vers la table suivante. La figure 1.8 présente le diagramme de traitement des paquets dans un réseau OpenFlow.

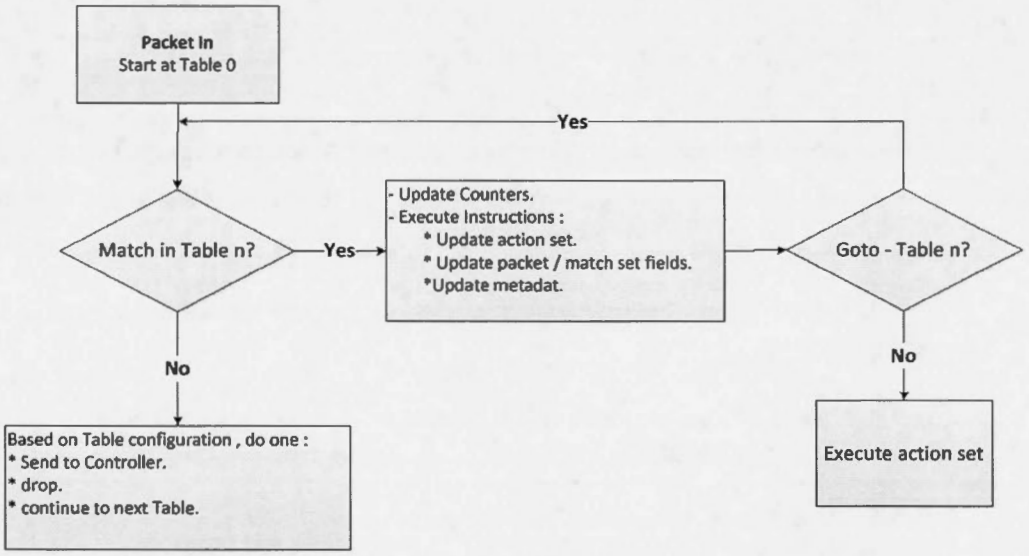


FIGURE 1.8: Traitement des paquets dans un réseau OpenFlow

Table de groupe :

Un "Group Table" se compose d'entrées de groupe. Ceci permet à OpenFlow d'envisager d'autres méthodes de transfert comme l'application d'un ensemble de transactions d'actions sur un type de flux. La figure 1.9 montre la structure d'une entrée de groupes :

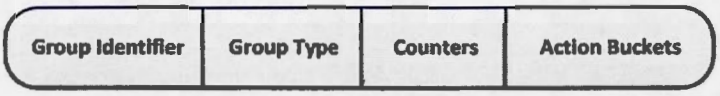


FIGURE 1.9: Entrées de "Group Table"

- *Group Identifier* : Utilisé pour identifier d'une façon unique un groupe donné.
- *Group Type* : Sert essentiellement à déterminer la sémantique du groupe.

- *Counters* : Ensemble des compteurs qui vont être mise à jour une fois qu'un paquet est traité par le groupe.
- *Action Buckets* : Contient une liste ordonnée de conteneurs d'actions, où chaque conteneur contient une série d'actions à exécuter et les paramètres associés.

1.3.5.2 Actions d'OpenFlow

À chaque entrée de "Flow Table", est associé un ensemble d'actions à exécuter sur les paquets, avant de les envoyer vers un port de sortie. Les actions doivent être exécutées dans le même ordre dont lequel elles sont présentées dans la table. Si une entrée ne contient aucune action, le paquet qui lui correspond sera supprimé. Une switch OpenFlow peut ne pas supporter tous les types d'actions. Cependant il doit supporter les actions requises pour le fonctionnement minimal d'OpenFlow. Lorsqu'une switch OpenFlow se connecte au contrôleur, elle doit indiquer à ce dernier quelles actions optionnelles elle supporte. Les actions requises pour une switch OpenFlow-only, sont comme suit :

1. Transférer le paquet à travers un ou plusieurs ports de sortie vers le nœud suivant dans le réseau.
2. Encapsuler et envoyer le paquet vers un contrôleur à travers le canal sécurisé. Cette action est principalement utilisée lorsqu'un paquet d'un flux est reçu pour la première fois. Le contrôleur peut décider si une nouvelle entrée doit être ajoutée dans la table, ou si le paquet doit être supprimé.
3. Supprimer tous les paquets appartenant à un flux est aussi une action qui peut être utilisée pour une raison de sécurité pour mettre fin à une attaque par exemple. Les actions optionnelles portent sur la modification de certains champs dans le paquet (Modification du VLAN ID, Réécriture de l'adresse MAC, recalcule du TTL...etc.).

1.3.5.3 Canal sécurisé

Le canal sécurisé est l'interface qui connecte chaque switch OpenFlow à un contrôleur. Cette interface permet au contrôleur de recevoir les messages de la switch et de pouvoir la

gérer à travers le réseau. Le canal doit être sécurisé afin d'assurer le bon déroulement des communications entre la switch et le contrôleur.

1.3.6 Contrôleur OpenFlow

Le contrôleur est un élément du réseau OpenFlow responsable de la gestion et la configuration des switches OpenFlow présentes dans le réseau. Il peut fonctionner d'une manière statique en ajoutant et supprimant des entrées dans le "Flow Table", ou d'une manière dynamique selon une configuration. Dans ce projet, nous avons utilisé le contrôleur ***Beacon***. C'est un contrôleur OpenFlow open source basé sur Java qui a été cré en 2010 (Erickson, 2013).

1.3.7 Protocole OpenFlow

Le contrôleur gère la switch via le protocole OpenFlow. En utilisant ce protocole, le contrôleur peut ajouter, modifier et supprimer des entrées de flux, à la fois d'une façon réactive (en réponse à des paquets) et de façon proactive.

1.3.7.1 Messages du protocole OpenFlow

Le protocole OpenFlow supporte trois types de messages :

1. *Messages du contrôleur vers la switch* : initiés par le contrôleur et utilisés pour gérer la switch ou pour savoir son état.
2. *Messages asynchrones* : initiés par la switch et utilisés pour mettre à jour le contrôleur.
3. *Messages symétriques* : initiés, soit par la switch soit par le contrôleur, et envoyés sans être préalablement sollicités par le destinataire.

1.3.7.2 Connexion entre le contrôleur et la switch

La switch OpenFlow doit être capable d'établir une communication TCP/IP via un port préalablement spécifié lors de la configuration et en utilisant une adresse IP configurable par

l'utilisateur. Lorsqu'une connexion OpenFlow est établie pour la première fois, la switch et le contrôleur doivent échanger un message *OFPT_HELLO*, qui contient un chiffre indiquant la plus grande version du protocole OpenFlow supportée. Une fois ce message est reçu, les deux parties de la connexion calculent la version du protocole OpenFlow supportée par l'autre. Si la switch et le contrôleur ont des versions OpenFlow compatibles, l'échange va continuer. Sinon, la machine qui ne supporte pas la version de l'autre va envoyer un message d'erreur et la connexion sera rompue.

1.4 GMPLS

1.4.1 Origines

Il est facile de deviner que GMPLS (Mannie, 2004) vient de MPLS. MPLS a été introduit dans les années 90 et ses meilleures caractéristiques sont qu'il pourrait mettre en place de multiples tunnels et appliquer des propriétés d'ingénierie de trafic pour eux. GMPLS est une extension du MPLS qui résout certains problèmes et ajoute de nouvelles fonctionnalités. Il dispose d'un ensemble de cinq interfaces comme le multiplexage temporel, commutateur de longueurs d'onde ou des interfaces de commutation à base de fibres optiques ainsi que le commutateur de paquets ou des interfaces de commutation de la couche 2 héritées de MPLS. En outre, GMPLS élimine la nécessité d'un opérateur et rend l'ensemble du réseau automatisé. Donc, aucune intervention humaine ne sera nécessaire dans le processus de tunnellisation.

1.4.2 Opération de GMPLS

Pour comprendre comment fonctionne GMPLS, il est nécessaire de comprendre d'abord comment fonctionne la technologie MPLS. MPLS utilise une étiquette ou bien « label » qui est ajouté à l'en-tête d'un paquet. Cette étiquette peut représenter un intervalle de temps, un espace physique ou même une seule longueur d'onde. Quand un paquet entre dans un réseau MPLS, une étiquette est ajoutée au paquet par le LER (Label Edge Router). Avec cette étiquette, le paquet se déplace sur la LSP (Labeled Switched Path). Ce paquet va passer à travers chaque LSR (Label Switched Router) qui appartient à la LSP. Chaque LSR

extrait l'étiquette, prend une décision de transmission et cherche l'étiquette correspondante à l'interface spécifique dans sa base de données de labels, puis insère cette étiquette. Quand le paquet arrive au dernier saut du réseau MPLS, le LER extrait l'étiquette et envoie le paquet à l'interface correspondante. Un mécanisme de contrôle est nécessaire afin de rendre ce processus possible. En MPLS, cela est fait par un protocole de distribution de labels qui est dans notre cas le protocole de réservation de ressources avec ingénieure de trafic RSVP-TE (Resource Reservation Protocol - Traffic Engineering). En outre, il est nécessaire de connaître la topologie du réseau et c'est la fonction d'un protocole de routage. Le protocole OSPF-TE (Open Shortest Path First- Traffic Engineering) est l'une des solutions possibles. Les deux protocoles seront expliqués plus tard. Ce qui différencie GMPLS par rapport à MPLS, c'est qu'il fonctionne avec différentes technologies et que le plan de contrôle et de données peuvent être physiquement séparés.

1.4.3 Routage GMPLS

En GMPLS, le routage a des responsabilités différentes de celle d'autres protocoles de couche 3. Il est utilisé pour partager des informations sur la topologie du réseau et la configuration de la qualité de service au niveau du plan de données. Dans notre cas, nous allons utiliser OSPF-TE qui fonctionne de manière similaire à OSPF, mais avec quelques extensions.

1.4.3.1 OSPF-TE

En utilisant le protocole OSPF-TE, le LSR peut connaître et transporter toutes les informations d'ingénierie de trafic nécessaires. Afin de conserver les informations de « Traffic Engineering », chaque hôte aura une base de données TED (Traffic Engineering Database). Le protocole OSPF ajoute des propriétés dynamiques à l'algorithme de calcul de route comme le maximum de la bande passante réservée, la bande passante non réservée et disponible. Ces champs sont répartis entre les nœuds du réseau via les champs TLV (Type Length Value).

1.4.4 Signalisation GMPLS

La signalisation GMPLS permet de créer et résilier un LSP. Dans notre cas, nous allons utiliser RSVP-TE.

1.4.4.1 RSVP-TE

Quand un LSR veut mettre en place un LSP, il envoie un message d'établissement de LSP. En RSVP, ce message est nommé « RSVP_PATH » et envoyé vers le LSR de destination. Quand ce dernier envoie un message d'acceptation au LSR d'origine, le LSP sera actif. Ce message est nommé « RSVP_RESV» et contient l'étiquette qui sera utilisée pour ce LSP. Afin de résilier le LSP, le LSR qui l'a demandé envoie un message « RSVP_TEAR » pour informer chaque LSR entre la source et la destination que le LSP sera supprimé et les ressources seront à nouveau disponibles. Ce message n'a pas besoin d'une réponse.

1.4.5 Conclusion

Dans ce chapitre, nous avons présenté les différents concepts théoriques liés à notre projet. Les deux domaines étudiés (réseau IP et réseau de transport) sont contrôlés séparément. Notre but est de définir un plan de contrôle commun pour avoir une interaction dynamique entre ces deux types de réseaux. Dans les deux chapitres suivants, nous montrons les solutions proposées pour réaliser cet objectif .Ces solutions sont basées sur les protocoles OpenFlow et GMPLS.

CHAPITRE II

PREMIÈRE SOLUTION : EXTENSION DU PROTOCOLE OPENFLOW

2.1 Introduction

Dans ce chapitre, nous introduisons notre première solution (Extension du protocole OpenFlow). Nous commençons par présenter les travaux faits par d'autres chercheurs et justifier le choix du protocole OpenFlow. Ensuite nous détaillons les différents modules implémentés. Enfin, nous présentons les expériences que nous avons réalisées afin de valider cette solution.

2.2 État de l'art

Des recherches ont été faites sur la conception, la réalisation et l'évaluation expérimentale d'un plan de contrôle unifié pour les réseaux IP/Optiques. Nous élucidons les architectures des principaux auteurs ayant travaillé sur ce sujet. Cela nous permet de comprendre le fonctionnement intrinsèque de ce plan de contrôle.

En effet, (Channegowda et al., 2013a) ont défini une approche basée sur une extension du protocole OpenFlow (OF) en 2012 et 2013. Ils proposent une approche OF pure étendue, qui comprend un agent OF sur chaque élément de réseau. L'agent développé utilise une API interne pour communiquer avec les éléments de réseau, et l'extension du protocole pour communiquer avec le contrôleur OF développé.

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 dans un plan de contrôle externe, découplée et automatisé.

De même, (Gudla et al., 2010) nous présentent OpenFlow comme une architecture et un plan de contrôle unifié pour les réseaux multicouches IP/Optiques. Ils nous 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).

(Liu et al., 2013) 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), assurera une flexibilité maximale aux opérateurs réseaux pour contrôler un réseau étant donné son architecture centralisée, et simplifiera et facilitera la gestion.

2.3 Pourquoi OpenFlow ?

OpenFlow garantit la séparation entre le plan de contrôle et le plan de données ainsi que le traitement des paquets comme des flux de données, où un flux de paquets est défini comme une combinaison d'entêtes de couche 2, 3 et 4. Ceci combiné avec des flux de circuits de la couche physique fournit une abstraction de flux simple qui s'adapte bien avec les deux types de réseaux (paquets et circuits). Ainsi, OpenFlow présente une plateforme commune qui traite les flux de granularité différente tout en assurant le contrôle et la gestion du plan de données à l'aide d'un logiciel extérieur au niveau du contrôleur OpenFlow (Figure 2.1). Le protocole OpenFlow est mature pour les réseaux de commutation de paquets, mais reste à un stade de départ pour les réseaux optiques. Afin de résoudre ce problème, nous proposons dans ce travail une extension du protocole OpenFlow 2.2. D'une part, nous implémentons un agent que nous appelons « OpenFlow Optical Agent » afin d'assurer la communication entre le contrôleur OpenFlow et les nœuds optiques. D'autre part, nous ajoutons un nouveau module au contrôleur OpenFlow qui est basé sur un élément de calcul de chemin (PCE). L'implémentation de ces deux éléments est détaillée dans les sections qui suivent.

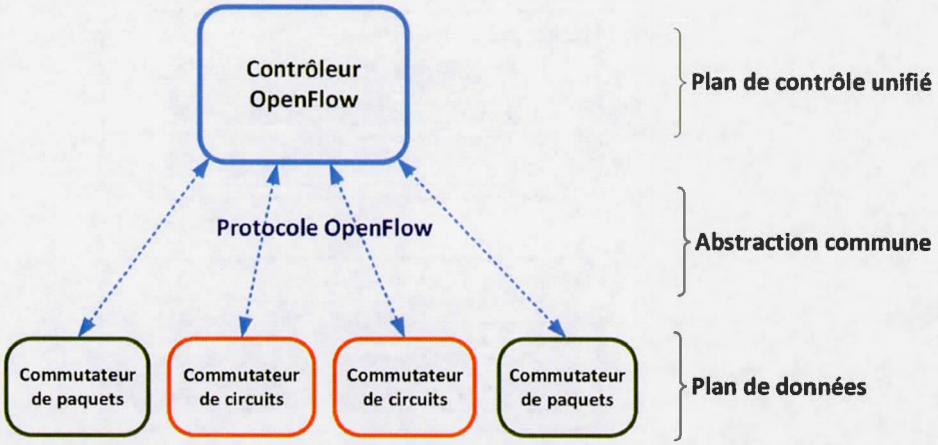


FIGURE 2.1: L'architecture unifiée d'un réseau de paquets-circuits convergent en utilisant OpenFlow

2.4 OpenFlow Optical Agent

La fonction primordiale de cet agent est de transformer les messages du protocole OpenFlow à des commandes TL1 permettant d'assurer la communication entre le contrôleur OpenFlow et les switches optiques. Cet agent est ajouté au niveau de chaque nœud optique et agit comme un commutateur virtuel. Il est composé de 3 éléments essentiels (Figure 2.2) ; (i) un *Canal Sécurisé* d'une switch OpenFlow ordinaire pour assurer l'échange de messages avec le contrôleur OpenFlow , (ii) un module de translation *OpenFlow/TL1* pour convertir les messages OpenFlow à des commandes TL1 et vice versa, et (iii) un module d'*Émulation des ports* pour associer les ports OpenFlow aux ports physiques d'une switch optique.

2.5 Path Computation Element (PCE)

L'objectif de PCE est d'implémenter un algorithme permettant la configuration d'un lightpath entre chaque paire de nœuds optiques afin de créer une topologie logique entièrement connectée (Zang et al., 2000). Nous avons créé une base de données TED pour enregistrer toutes les informations sur la topologie du réseau. Comme le contrôleur OpenFlow a une gestion centralisée pour les éléments de réseau, la TED sera mise à jour en cas de création ou

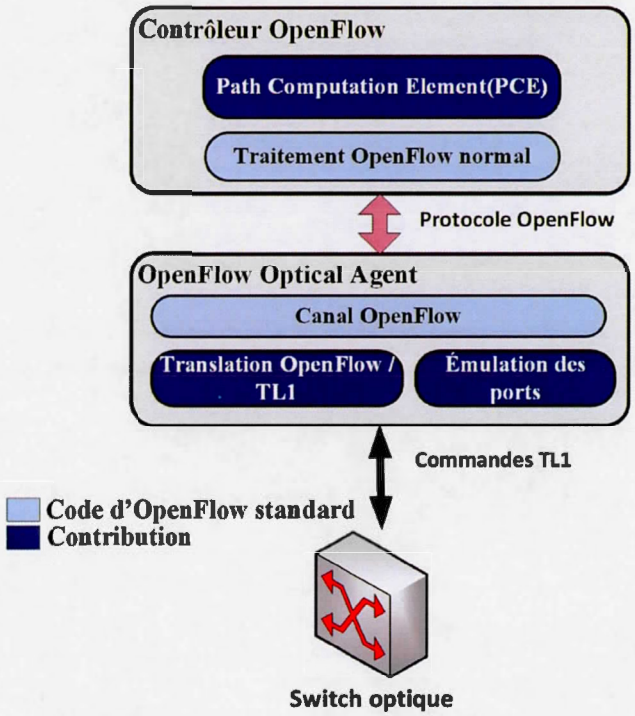


FIGURE 2.2: Extension du protocole OpenFlow

suppression d'un ligthpath ou le changement d'états des ports. Nous avons implémenté deux modules afin d'atteindre notre objectif; *Executor* et *ONS Adapter* (Figure 2.3).

2.5.1 Executor

L'objectif est de garantir l'utilisation d'une longueur d'onde qu'une seule fois dans la même fibre. Chaque longueur d'onde assure l'acheminement du trafic entre une paire de nœuds optiques. Par conséquent, plusieurs longueurs d'onde sont réservées dans une seule fibre optique afin d'établir plusieurs lightpahts. Ces connexions entre les nœuds optiques se font en deux étapes :

- **Routage :** Nous utilisons l'algorithme de Dijkstra afin de trouver le plus court chemin entre chaque paire de nœuds. Dans notre cas, nous intéressons à une topologie de réseau qui contient à la fois des switches OpenFlow et des switches optiques (ONS).

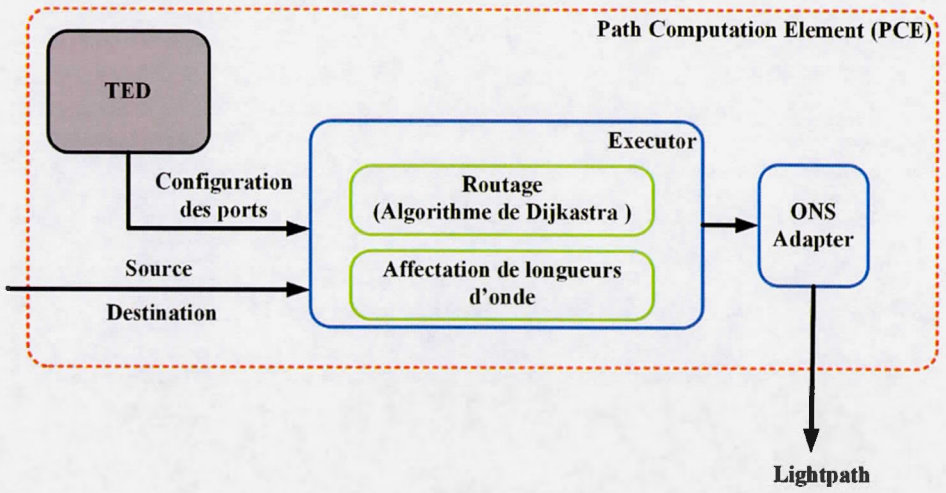


FIGURE 2.3: Path Computation Element (PCE)

- **Affectation de longueurs d'onde :** Une fois que les différentes routes sont déterminées, le problème d'affectation de longueurs d'onde peut être représenté comme un problème de coloration de graphe (Berge, 1958). En théorie des graphes, colorer un graphe revient à attribuer une couleur à chacun de ses sommets de manière à ce que deux sommets reliés par une arête soient de couleur différente (Figure 2.4). La solution optimale consiste à utiliser un nombre minimal de couleurs, dit *nombre chromatique.* Dans notre cas, chaque longueur d'onde dans une fibre optique est présentée par un couleur.

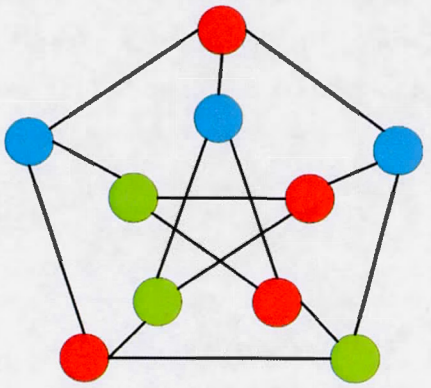


FIGURE 2.4: Exemple de coloration de graph

2.5.2 ONS Adapter

Chaque ONS (Optical Network Switch) est constitué d'un ensemble de cartes et chaque carte contient un ensemble de ports configurables (CiscoONS, 2014). Un ONS d'extrémité est connecté à une switch OpenFlow via les cartes WSS et DMX, alors que les ONS dans le cor du réseau sont interconnectés via des cartes LINE. En plus, les connexions entre les ONS sont bidirectionnelles. Ces spécifications particulières aux ONSs nous ont amenés à ajouter ce module.

2.6 Expériences

Afin de valider notre solution basée sur OpenFlow, nous avons réalisé deux expériences. La première consiste à créer un lightpatgh entre deux clients (Scénario A) et la deuxième est pour créer un chemin optique de restauration en cas d'échec du premier lightpath (Scénario B).

2.6.1 Cadre empirique

Notre environnement d'expérimentation est élaboré avec des équipements pour les réseaux IP et optiques. Il est composé de deux clients A et B qui sont reliés directement à deux switches OpenFlow. Chaque switch est liée à un convertisseur électrique / optique (Cisco Cat 6504-A et Cisco Cat 6504-B). Chaque nœud optique (Cisco ONS 15454) est contrôlé par un agent OpenFlow en utilisant les commandes TL1. Ces agents et les switches OpenFlow sont connectés à un contrôleur OpenFlow. Cet environnement, faisant office de banc d'essai, est situé dans notre laboratoire de réseau informatique à l'UQAM. Il est schématisé dans la figure 2.5.

2.6.2 Scénario A : Établissement d'un lightpath

Comme le montre la figure 2.5, on considère le premier paquet envoyé du client A vers le client B comme une requête d'établissement de lien. La connexion est établie entre les deux

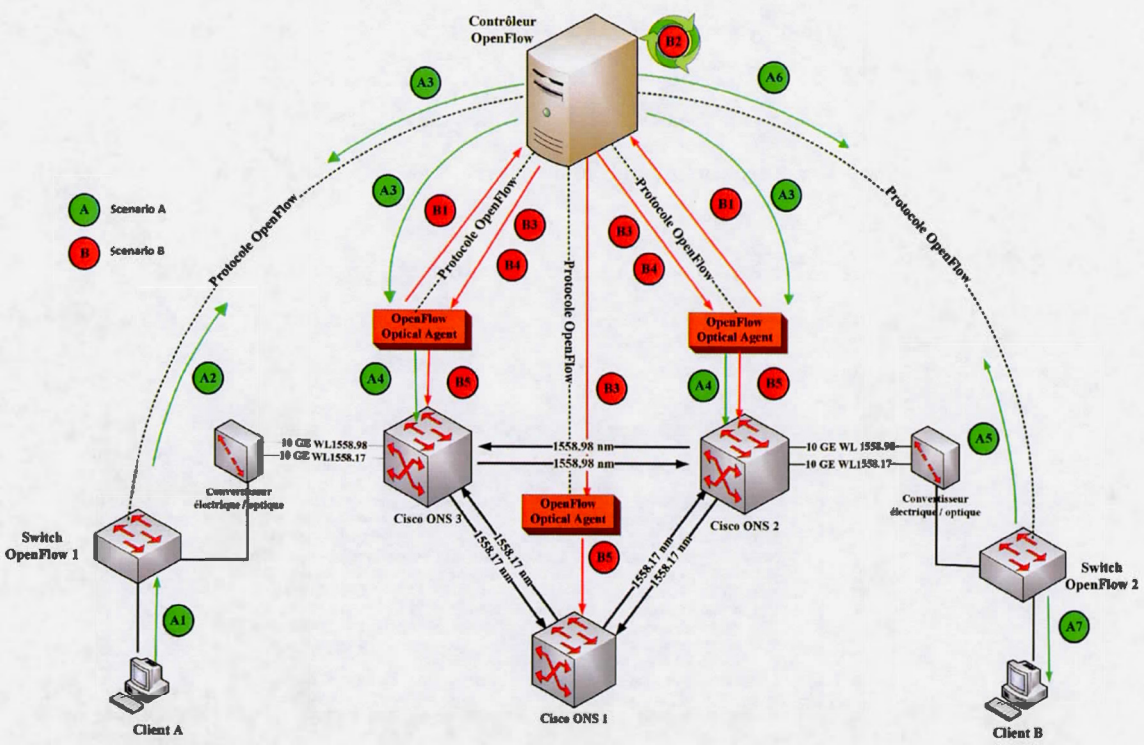


FIGURE 2.5: La configuration du réseau et les messages échangés pour les scénarios A et B

clients selon les étapes suivantes (Figure 2.5) :

- **Étape A1 :** Le premier paquet envoyé par le client A au client B arrive à la première switch OpenFlow.
- **Étape A2 :** La switch OpenFlow 1 ne trouve aucun flow entry pour transmettre ce paquet puisque sa table de flux est initialement vide. Donc, elle l'encapsule dans un message *OFPT_PACKET_IN* et l'envoie au contrôleur OpenFlow.
- **Étape A3 :** Le contrôleur calcule le chemin du client A vers le client B en utilisant le PCE et envoie un message *OFPT_PACKET_OUT* à la switch OpenFlow 1 et des messages de type *OFPT_FLOW_MOD* aux agents OpenFlow pour créer le lightpath.
- **Étape A4 :** Lorsque les agents OpenFlow reçoivent ces messages, ils les traduisent aux commandes TL1 appropriées et les envoient aux switches optiques.
- **Étape A5 :** Après la création du lightpath, le paquet traverse le domaine optique jusqu'à la switch OpenFlow 2. Ce dernier ne trouve pas un flow entry dans sa table

de flux pour l'acheminer. Donc elle l'envoie au contrôleur dans un message de type *OFPT_PACKET_IN*.

- **Étape A6 :** Le contrôleur envoie un message *OFPT_PACKET_OUT* à la switch OpenFlow 2 contenant une nouvelle règle pour transmettre le paquet à la destination finale (client B).
- **Étape A7 :** La switch OpenFlow 2 transmet le paquet au client B

La figure 2.6 montre la capture wireshark au niveau du contrôleur OpenFlow qui présente les différents messages échangés afin de créer le lightpath.

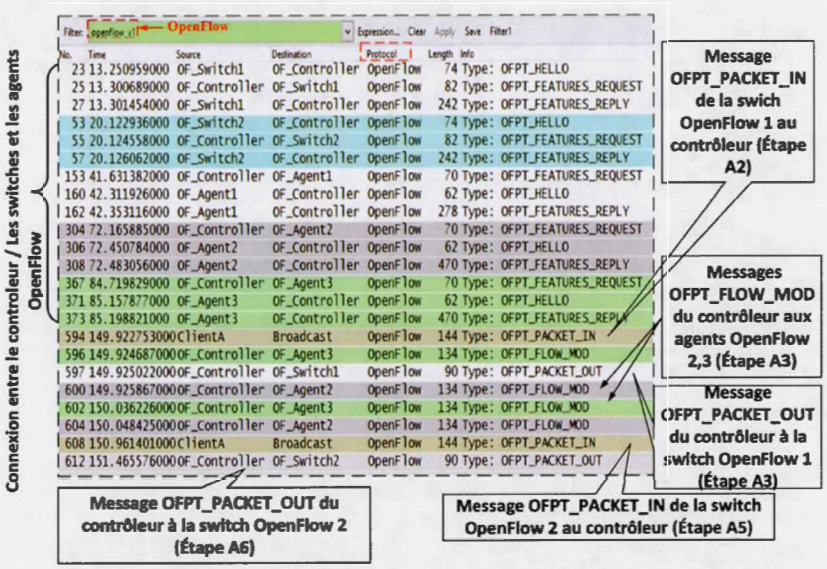


FIGURE 2.6: Capture Wireshark : Scénario A

2.6.3 Scénario B : Restauration du lightpath

Ce scénario montre comment le contrôleur OpenFlow agit lorsqu'il y'a une rupture d'un lien optique. La restauration du lien se fait selon les étapes suivantes (Figure 2.5) :

- **Étape B1 :** Lorsque la connexion entre l'ONS 3 et l'ONS 2 échoue, les deux agents optiques correspondant à ces ONSs envoient au contrôleur OpenFlow des messages *OFPT_PORT_STATUS* pour l'informer de ce changement d'état.

- **Étape B2 :** Le contrôleur OpenFlow calcule un nouveau chemin optique suite à ce changement d'état.
- **Étape B3 :** Le contrôleur envoie des messages *OFPT_FLOW_MOD* (Type = ADD-FLOW) aux agents OpenFlow pour la création d'un nouveau lightpath. Dans ce cas, le lien devient entre l'ONS 2 et l'ONS 3 en passant par l'ONS 1.
- **Étape B4 :** Le contrôleur envoie des messages *OFPT_FLOW_MOD* (Type = DE-LETEFLOW) aux agents OpenFlow correspondants aux ONS 2 et 3 pour supprimer l'ancien lightpath.
- **Étape B5 :** Lorsque les agents OpenFlow reçoivent ces messages, ils les traduisent aux commandes TL1 appropriées puis les envoient aux switches optiques.

La figure 2.7 montre la capture wireshark au niveau du contrôleur OpenFlow qui présente les différents messages échangés lors de la restuaration du lightpath .

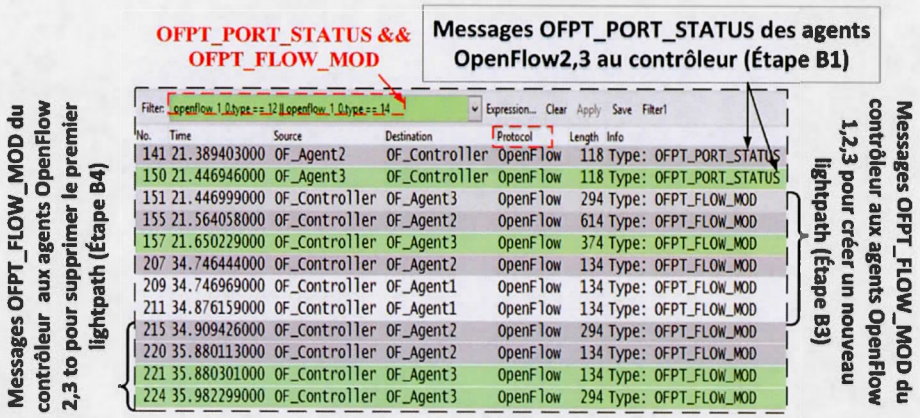


FIGURE 2.7: Capture Wireshark : Scénario B

2.7 Conclusion

Dans ce chapitre, nous avons présenté la première solution proposée. Deux modules ont été implémentés pour faire l'extension du protocole OpenFlow (OpenFlow Optical Agent et PCE). Les deux expériences réalisées montrent bien la validité de la solution et comment OpenFlow permet de contrôler d'une façon dynamique les deux couches IP et optiques.

CHAPITRE III

DEUXIÈME SOLUTION : GMPLS

3.1 Introduction

GMPLS peut être utilisé comme solution afin de définir un plan de contrôle unifié pour les réseaux de paquets et les réseaux de circuits. Cependant l'implantation de cette technologie est complexe et difficile car il y a encore beaucoup d'éléments du réseau qui ne sont pas compatibles avec GMPLS. DRAGON (DRAGON, 2006) est un software open source qui permet de résoudre ce problème dans les réseaux Ethernet en utilisant le protocole SNMP afin de contrôler ces éléments et les rendre capables d'opérer dans un réseau GMPLS.
Dans ce chapitre, nous présentons tout d'abord les travaux de recherche effectués dans ce domaine. Puis, nous introduisons l'outil DRAGON en montrant ses différents éléments. Ensuite, nous expliquons comment nous avons déployé cette solution dans notre banc d'essai en présentant les différents tests que nous avons réalisés.. Enfin, nous détaillons notre contribution afin d'adapter DRAGON avec notre switch optique (Cisco ONS 15454).

3.2 État de l'art

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 unique pour les réseaux optiques multicouches IP/Optiques. Plusieurs chercheurs se sont orientés vers ce chemin, et les techniques utilisées permettent de mettre en évidence la faisabilité. Nous montrons les architectures des principaux auteurs réputés dans ce domaine.

Comme plan de contrôle unique, 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 contrôles 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 à ces objectifs à travers le développement d'une plate-forme comprenant tous les éléments constitutifs de ces réseaux (dimensionnement de réseau, plan de gestion et plan de contrôle GMPLS). En plus le projet TBONES 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 par couche de commutation du plan de données) vers un modèle d'interconnexion de plan de contrôle unifié où une instance de plan de contrôle unique conduit 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. (Liu et al., 2012b) nous disent que les plans de contrôle unique basés sur SDN sont très importants pour les réseaux, car ils peuvent permettre l'approvisionnement dynamique des chemins optiques, la restauration, l'amélioration de l'intelligence du réseau et la réduction du temps de traitement et les dépenses opérationnelles. Ces dernières années, il y a eu de grandes progressions dans ce domaine, allant de l'architecture GMPLS traditionnelle au PCE / GMPLS (élément de calcul). Ils présentent les techniques habilitantes pour chaque plan de contrôle, et nous résume leurs avantages et leurs inconvénients.

3.3 DRAGON

3.3.1 Introduction

Le projet DRAGON étudie et développe des logiciels open source permettant le provisionnement dynamique des ressources du réseau dans un domaine intra contenant des technologies hétérogènes. Le projet permet la communication entre les différents types de réseaux grâce au contrôle GMPLS. Pour son implémentation, DRAGON déploie l'infrastructure de réseau

IP et crée une topologie de réseau optique capable d'opérer avec GMPLS et permettant le provisionnement dynamique des paths en réponse à des requêtes d'utilisateurs finaux. Les équipements optiques de transport et de commutation agissant comme des LSRs permettent ainsi de fournir des ressources de réseau déterministes au paquet, longueur d'onde et la fibre cross-connect.

3.3.2 Composants du plan de contrôle DRAGON

Le software DRAGON est censé à fonctionner comme un plan de contrôle au sein d'un réseau GMPLS. L'architecture de ce plan de contrôle comporte trois éléments de base : (i) CSA (Client System Agent), (ii) VLSR (Virtual Label Switch Router) et (iii) NARB (Network Aware Resource Broker).

3.3.2.1 CSA (Client System Agent)

Le CSA est un logiciel qui fonctionne sur tout système qui termine le lien de plan de données du service provisionné. C'est le logiciel qui participe au protocole GMPLS pour permettre le provisionnement de demande de bout en bout entre deux clients.

3.3.2.2 VLSR (Virtual Label Switch Router)

Aujourd'hui, la technologie GMPLS ne s'est pas étendue sur une grande échelle, car il y a encore de nombreux éléments dans le réseau qui ne la supportent pas. Afin de franchir cette difficulté, le projet DRAGON a développé le VLSR. Il permet aux commutateurs Ethernet d'agir comme des switches de labels dans un réseau GMPLS. Ce logiciel implémente la partie contrôle d'un nœud GMPLS et peut se connecter à des commutateurs Ethernet via diverses interfaces. VLSR est prêt à travailler avec différents types de commutateurs et nous allons commencer avec le commutateur Ethernet. Ensuite, nous allons expliquer comment adapter ce logiciel pour travailler avec notre switch optique (Cisco ONS 15454). VLSR utilise le protocole SNMP pour communiquer avec le commutateur Ethernet, mais il peut aussi communiquer par le biais d'une interface de ligne de commande (CLI) ou d'autres méthodes. Pour communiquer

avec d'autres VLSRs et CSAs, il utilise le protocole de routage OSPF-TE et le protocole de signalisation RSVP-TE. Ces deux protocoles ont été expliqués dans le chapitre 1 mais ils ont été étendus pour supporter les besoins de DRAGON. Les LSPs Ethernet mis en place par DRAGON, utilisent les VLAN-tags comme étiquettes. Contrairement à la technologie MPLS, le projet DRAGON ne prend en charge qu'une valeur constante de VLAN par LSP, en d'autres termes pas d'échange d'étiquette VLAN le long de data-path du LSP.

3.3.2.3 NARB (Network Aware Resource Broker)

Le NARB est un élément du projet DRAGON qui représente un système autonome (AS). Sa fonction principale est le calcul de chemin interdomaine. Le NARB permet aussi de communiquer avec les protocoles de routage intradomaine. Dans notre cas c'est le protocole OSPF-TE.

3.4 Test du software DRAGON avec un commutateur Ethernet commercial

Afin de comprendre le fonctionnement du software DRAGON, nous avons créé une infrastructure très simple qui va nous aider à comprendre son fonctionnement.

3.4.1 Scénario 1 : Infrastructure de base

Pour ce test, 3 PCs [1] à moins de 500 MHz avec 256 Mo de RAM et 1 Go d'espace libre sur le disque dur avec deux cartes Ethernet chacun ont été utilisés. Nous avons utilisé le modèle Cisco ME-3400 comme commutateur commercial pour interagir avec le software DRAGON. Afin de mettre en place l'environnement pour le software DRAGON, nous nous référons à la configuration présentée par la figure 3.1.

1. Les hôtes et les machines de contrôle VLSR peuvent fonctionner sous Linux (version du noyau 2.4.20 ou supérieur) ou FreeBSD (version du noyau 4.11 ou au-dessus), ou un système hybride de Linux et FreeBSD.

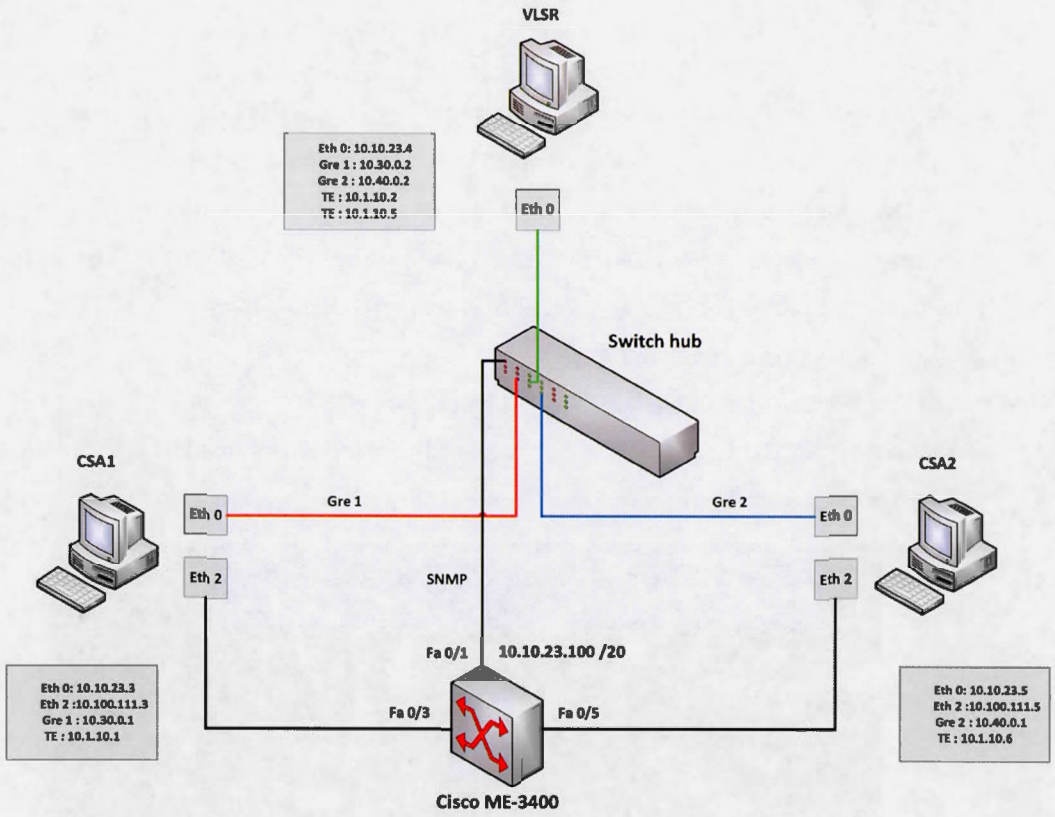


FIGURE 3.1: Test du software DRAGON avec un commutateur Ethernet commercial

3.4.2 Configuration logicielle requise

Les logiciels requis avant l'installation du software DRAGON sont contenus dans un seul dossier qui est disponible via le lien ci-dessous :

http ://dragon.east.isi.edu/twiki/pub/Main/VLSR/dragon.dependencies.tar.gz

Le package peut être décompressé en tapant la commande suivante :

#tar –zxf dragon.dependencies.tar.gz

Les commandes suivantes permettent d'installer les logiciels requises :

#cd dependence-package

./makealldeps.sh

3.4.3 Installation

Le logiciel VLSR peut être téléchargé à partir de : http ://dragon.east.isi.edu
Le package peut être décompressé en tapant la commande suivante :
#tar –zxf dragon-sw-snapshot.xxxx.tar.gz (voir le site web pour les dernières valeurs "xxxx")
Après être entré dans le répertoire de travail où le package DRAGON a été décompressé, les commandes suivantes permettent d'installer le software DRAGON :
./do_build.sh pour configurer le paquage
sudo sh do_install.sh pour installer le software DRAGON
Par défaut, le logiciel est installé dans /usr/local/dragon. Après une installation réussie, le message suivant s'affiche (Figure 3.2) :

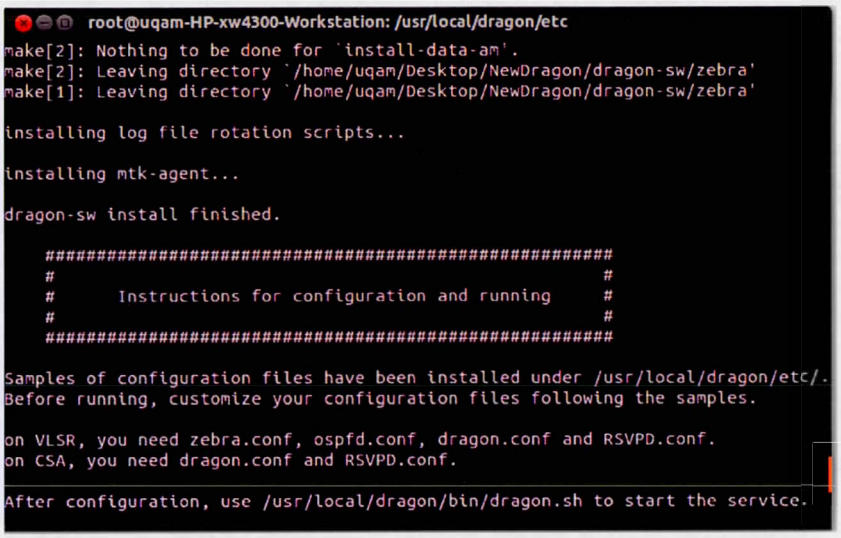


FIGURE 3.2: Message d'installation réussie de VLSR

3.4.4 Configuration

Dans l'exemple de réseau représenté dans la figure 3.1, nous avons besoin de mettre en place deux tunnels GRE (Generic Routing Encapsulation). Le premier du CSA1 vers la machine de contrôle (VLSR) appelé GRE1 et le deuxième de CSA2 vers la machine de

contrôle (VLSR) appelé Gre2. Le routage et les messages de signalisation entre CSA1 et CSA2 peuvent être transmis à travers les tunnels GRE. En plus, nous avons besoin de configurer les différents démons du projet DRAGON (OSPF, RSVP, ZEBRA) dans chaque machine Linux. L'Appendice A contient toutes les configurations que nous avons réalisées pour ce scénario.

3.4.5 Configuration du commutateur Cisco

Le commutateur (qui soutient Net-SNMP) est maintenant connecté aux trois machines. Le VLSR contrôlera le commutateur et il est donc important de le configurer. Le serveur SNMP doit être activé sur le commutateur Cisco. Nous ajoutons donc une communauté nommée «dragon» pour le serveur SNMP. La communauté doit avoir les privilèges de lecture et écriture. Pour le cas de la switch Cisco ME-3400, nous utilisons les commandes suivantes :
#configure
snmp-server community dragon rw
snmp-server enable traps
Pour les commutateurs qui utilisent la méthode de contrôle SNMP mais ne prennent pas en charge la création et la suppression dynamique de VLAN, nous devons créer des VLANs vides qui seront utilisés dans l'approvisionnement futur. Dans notre cas, nous commençons avec deux VLANs (VLAN 100 et VLAN 200).

3.4.6 Test

Pour exécuter les différents démons du DRAGON, nous pouvons utiliser le script suivant dans chaque machine Linux :
/usr/local/dragon/bin/dragon.sh start-vlsr
Dans notre cas, les tunnels GRE sont utilisés pour le plan de contrôle et les deux ports Ethernet 2 de chauqe CSA sont utilisés pour le plan de données. Afin de créer un LSP, du CSA2 vers le CSA1 nous connectons au démon DRAGON via le port 2611 du CSA2 (# telnet localhost 2611) et nous exécutons les commandes suivantes :
1 cnl_host2> edit lsp 5to3-A-
2 cnl_host2(edit-lsp-5to3-A-)# set source ip-address 10.10.23.5 lsp-id 100 destination ip-

address 10.10.23.3 tunnel-id 200

3 cnl_host2(edit-lsp-5to3-A-)# set bandwidth gige swcap l2sc encoding ethernet gpid ethernet

4 cnl_host2(edit-lsp-5to3-A-)# exit

5 cnl_host2> show lsp

La figure 3.3 présente le résultat obtenu. Concernant les commandes exécutées, la première

```
cnl_host2-dragon> show lsp
                        **LSP status summary**
Name        Status     Dir   Source (IP/LSP ID)   Destination (IP/Tunnel ID)
-------------------------------------------------------------------------
5to3-A      Edit       <=>   10.10.23.5          10.10.23.3
                             100                 200
cnl_host2-dragon>
```

FIGURE 3.3: LSP en mode « Edit »

ligne correspond au nom de la LSP que nous voulons créer (lsp 5to3-A-). La ligne deux est pour spécifier les adresses IP source et destination, le LSP-id qui identifie le LSP pour l'hôte source, et le tunnel-id qui identifie le tunnel pour l'hôte de destination. La troisième ligne spécifie le type et la bande passante du LSP demandé, dans notre cas un gige de la bande passante, commutation de couche 2 (l2sc) et le protocole Ethernet. Dans notre cas, nous ne définissons pas le VLAN du LSP et le logiciel DRAGON choisit le premier VLAN disponible (Vlan 100). Une fois nous finissons de définir le nouveau LSP, il est nécessaire de le mettre en fonctionnement avec la commande "commit" et le résultat obtenu est montré dans la figure 3.4 (L'état de LSP change de *Edit* à *In service*) .

1 cnl_host2> commit lsp 5to3-A

2 cnl_host2> show lsp

Dans un autre test, nous avons créé quatre LSPs afin de comprendre comment le logiciel DRAGON choisit les étiquettes et les VLANs des LSPs. La figure 3.5 montre les 4 LSPs crées.

```
cnl_host2-dragon> show lsp
                          **LSP status summary**

Name        Status     Dir   Source (IP/LSP ID)  Destination (IP/Tunnel ID)
-------------------------------------------------------------------------
5to3-A      In service <=>   10.10.23.5          10.10.23.3
                             100                 200
cnl_host2-dragon>
```

FIGURE 3.4: LSP en mode « In service »

```
cnl_host1-dragon> show lsp
                          **LSP status summary**

Name        Status     Dir   Source (IP/LSP ID)  Destination (IP/Tunnel ID)
-------------------------------------------------------------------------
1           In service  =>   10.10.23.5          10.10.23.3
                             100                 200
2           In service  =>   10.10.23.5          10.10.23.3
                             101                 201
3           In service <=>   10.10.23.3          10.10.23.5
                             300                 400
4           In service <=>   10.10.23.3          10.10.23.5
                             301                 401
cnl_host1-dragon>
```

FIGURE 3.5: Création de quatre LSPs

3.4.7 Signalisation GMPLS

En suivant la capture Wireshark du VLSR (Figure 3.6), nous expliquons comment le software DRAGON choisit les étiquettes pour créer les LSPs. Le premier LSP que nous voulons mettre en place était LSP "1". Dans la première ligne, CSA2 envoie un message *RSVP_PATH* pour le VLSR par le GRE2 (Source : 10.40.0.1 / Destination : 10.40.0.2). Dans la deuxième ligne, le VLSR transmet ce message au CSA1 à travers le GRE1 (Source : 10.30.0.2 / Destination : 10.30.0.1). CSA1 répond à cette demande par un message *RSVP_RESV* qui contient l'étiquette (16) au VLSR. Ensuite, dans la ligne 4, le VLSR transmet le message *RSVP_RESV* au CSA2 avec l'étiquette 16. Dans les lignes qui suivent, la même procédure est répétée avec les LSPs "2", "3" et "4". Généralement, les étiquettes sont différentes le long d'un LSP. La valeur de l'étiquette est définie dans l'ordre chronologique des connexions d'entrée que le composant possède. Dans notre cas, le VLSR reçoit quatre connexions d'entrée et il attribue

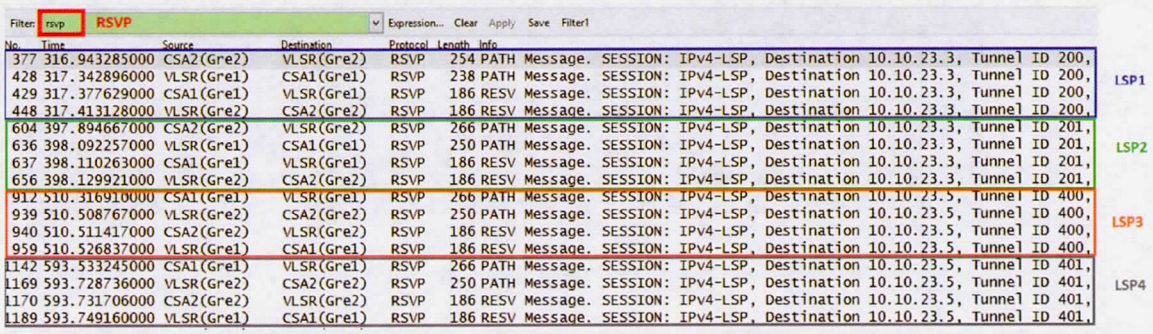


FIGURE 3.6: Capture wireshark à partir du port eth0 du VLSR

donc à chacune une étiquette supérieure à 15 (16, 17, 18 et 19).

3.5 Test du software DRAGON avec deux commutateurs Ethernet commerciaux

3.5.1 Scénario 2 : Infrastructure avancée (VLSR to VLSR)

Pour ce scénario, l'infrastructure de base (Scénario 1) a été étendue avec un deuxième commutateur. Pour contrôler le deuxième commutateur, un autre VLSR est nécessaire. Maintenant, l'infrastructure se compose de quatres machines Linux et deux commutateurs Ethernet. Deux machines fonctionnent comme CSAs, les deux autres fonctionnent comme VLSRs. La figure 3.7 montre une vue d'ensemble de l'infrastructure utilisée pour le scénario 2. Dans cette architecture, le plan de contrôle est composé de deux CSAs et deux VLSRs. Ils sont connectés via le switch hub. Les tunnels GRE ont été créés entre les CSAs et les VLSRs et entre les VLSRs eux même. Les deux commutateurs sont liés au switch hub pour permettre la gestion SNMP par les VLSRs. Les interfaces Ethernet 2 des CSAs font partie du plan de données et sont reliées chacune à l'un des deux commutateurs via le port FastEthernet 0/9. Les deux commutateurs sont connectés par le port GigabitEthernet 0/2.
Les fichiers de configuration pour ce scénario sont ajoutés dans l'Appendice B.

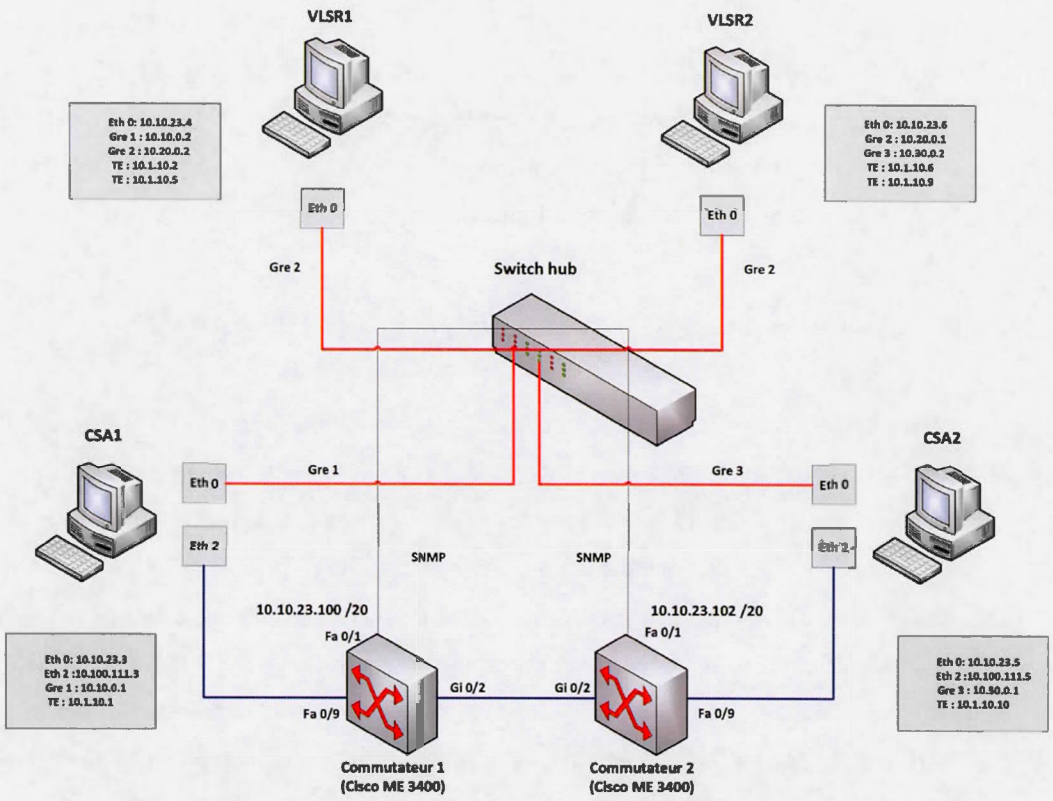


FIGURE 3.7: Vue d'ensemble de l'infrastructure étendue (Scénario 2)

3.5.2 Test

Pour ce scénario, nous avons fait les mêmes tests que le premier scénario. Des LSPs ont été créés entre les deux CSAs (du CSA2 vers le CSA1) en passant par les deux commutateurs Ethernet. Le mécanisme de création de LSP est décrit dans la section suivante.

3.5.3 Signalisation GMPLS

Après avoir exécuté la commande « commit » de validation de LSP, le CSA2 envoie un message *RSVP_PATH* au VLSR2. Ce dernier le transmet au VLSR1, qui a son tour le transmet au CSA1 parce qu'il n'est pas le destinataire du LSP. Quand CSA1 reçoit ce

message, il répond par un message $RSVP_RESV$ et envoie le vers le VLSR1. En recevant ce message, VLSR1 envoie des commandes SNMP au commutateur 1 pour placer les ports dans le VLAN correspondant. Ensuite, il transmet le message vers VLSR2 qui à son tour configure le commutateur 2 en utilisant le protocole SNMP et transmet le message $RSVP_RESV$ vers CSA2. En ce moment, le LSP devient actif et peut être utilisé.

3.6 Adaptation du software DRAGON pour la switch Cisco ONS 15454

Dans cette section, nous expliquons comment nous avons adapté le software DRAGON pour notre switch optique (Cisco ONS 15454). Pour configurer ce genre de switch, nous utilisons les commandes TL1. C'est un protocole de gestion largement utilisé dans les télécommunications, et spécifiquement, pour gérer les réseaux optiques SONET (Synchronous Optical NETwork). Pour le projet DRAGON, le protocole SNMP est utilisé pour configurer les commutateurs. Et donc, pour l'adapter pour la switch Cisco ONS 15454, nous avons développé un «SNMP /TL1 Gateway » dont le rôle principal est la translation des messages SNMP à des commandes TL1 et vice versa. Cette solution est composée de deux modules (Figure 3.8) ;(i) Agent SNMP et (ii) Agent TL1.

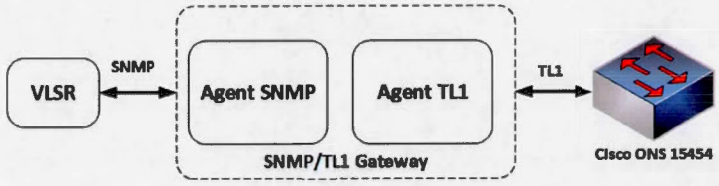


FIGURE 3.8: SNMP /TL1 Gateway

- **Agent SNMP :** Nous avons utilisé la libraire java snmp4j (SNMMP4J, 2003) pour développer un agent SNMP. D'une part, il reçoit les requêtes SNMP provenant du VLSR et les transforme à des commandes TL1. D'autre part, il transforme les commandes TL1 provenant de l'ONS à des réponses SNMP puis les envoie vers le VLSR.
- **Agent TL1 :** Pour développer cet agent, nous avons utilisé l'API java iReasing (TL1API, 2002). Sa fonction principale est d'assurer la communication entre le Cisco ONS 15454 et l'agent SNMP en utilisant les commandes TL1.

L'*Agent SNMP* reste en écoute à la machine VLSR sur le port 161, il reçoit et envoie les messages SNMP en se basant sur un MIB local (Management Information Base). L'*Agent TL1* est également en écoute au Cisco ONS 15454, il traite les commandes TL1 et les transforme à des messages SNMP et vice versa. La figure 3.9 montre les différents messages échangés lors de la création d'un LSP entre CSA2 et CSA1.

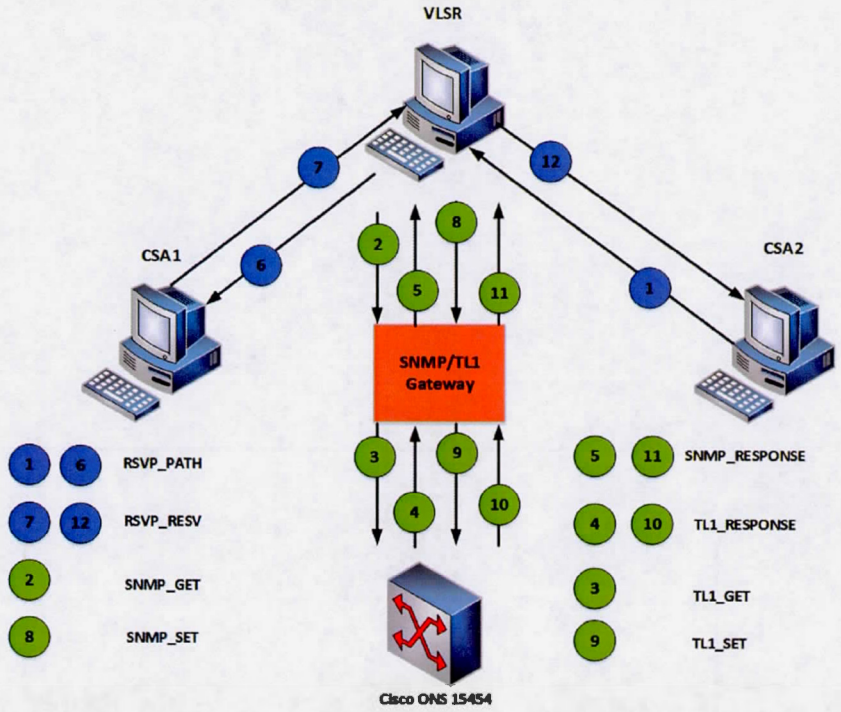


FIGURE 3.9: Messages échangés lors de la création d'un LSP

3.6.1 Scenario 3 : Test du software DRAGON avec une switch Cisco ONS 15454

L'infrastructure de ce scénario est une extension du celle du scénario 1. Dans ce cas, nous avons ajouté deux équipements réseau pour assurer la conversion entre le domaine électrique et optique (Cisco Cat 6504-A et Cisco Cat 6504-B) puisque nous avons introduit une switch optique (Cisco ONS 15454). La machine, ou nous avons implémenté le *SNMP/TL1 Gateway*, contient deux interfaces Ethernet. La première pour se connecter au VLSR et la deuxième

pour le Cisco ONS 15454. La figure 3.10 montre une vue d'ensemble de l'infrastructure utilisée pour ce scénario. Les fichiers de configuration sont les mêmes comme le scénario 1 (Appendice A).

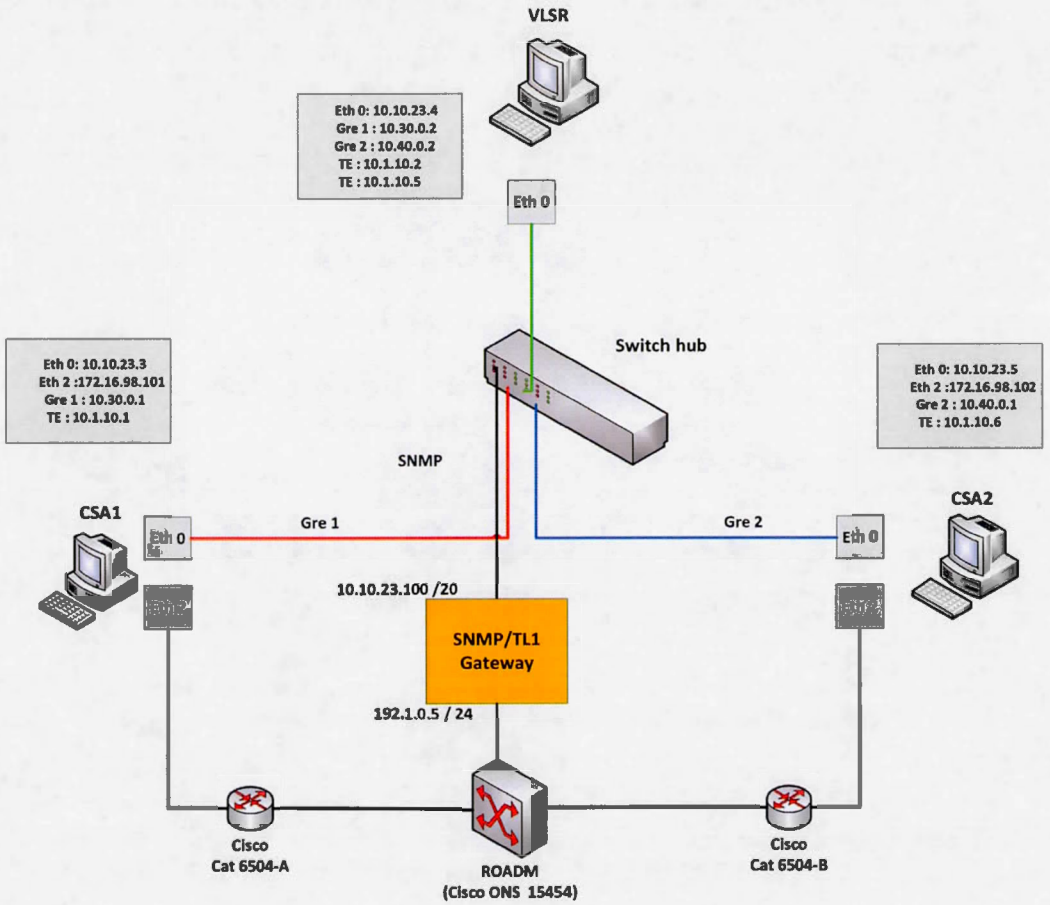


FIGURE 3.10: Test du software DRAGON avec une switch Cisco ONS 15454

3.6.2 Scenario 4 : Test du software DRAGON avec deux switches Cisco ONS 15454

Pour contrôler le deuxième ONS, un autre VLSR est nécessaire. Maintenant l'infrastructure se compose de quatre machines Linux, deux ONSs et un *SNMP/TL1 Gateway* pour

chaque ONS. Deux machines fonctionnent comme CSAs, les deux autres fonctionnent comme VLSRs. Les deux *SNMP/TL1 Gateway* sont installés dans deux machines virtuelles qui se trouvent dans la même machine physique. Chacune reste en écoute à un VLSR sur le port 161 et contrôle un ONS (Figure 3.11). Les fichiers de configuration pour ce scénario sont les mêmes que pour le scénario 2 (Appendice B).

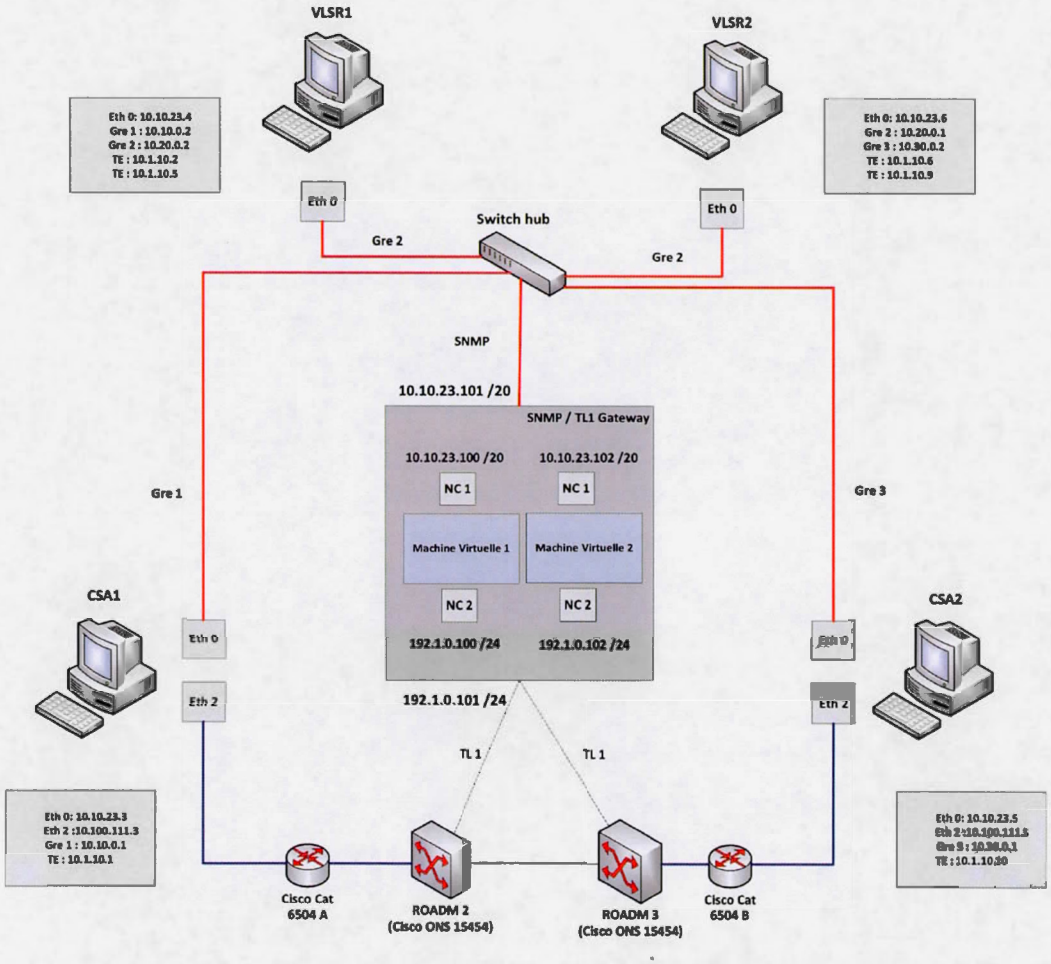


FIGURE 3.11: Test du software DRAGON avec deux switches Cisco ONS 15454

3.7 Conclusion

Dans ce chapitre, nous avons présenté la deuxième solution proposée qui est basée sur le protocole GMPLS. Nous avons rencontré plusieurs problèmes lors de l'installation et le déploiement du software DRAGON du a l'insuffisance de la documentation disponible sur le site officiel du projet DRAGON (surtout au niveau des fichiers de configuration). Finalement, nous avons réussi à réaliser notre objectif. Les quatre scénarios réalisés montrent bien la validité de notre solution.

CHAPITRE IV

COMPARAISON DES DEUX SOLUTIONS

4.1 Introduction

Dans les deux chapitres précédents, nous avons présenté les différents tests et expériences que nous avons réalisés afin de valider nos deux solutions. Pour comparer ces deux solutions, nous avons utilisé un simulateur java [1] pour simuler leurs performances dans deux topologies de réseaux optiques réels. Dans ce chapitre, nous commençons par présenter les résultats expérimentaux pour chaque solution. Ensuite, nous détaillons les résultats de simulation pour les deux topologies étudiées.

4.2 Résulats expérimentaux

Le tableau 4.1 montre le temps (en ms) utilisé par chaque solution pour l'établissement des liens.

Dans le tableau 4.1, Path1 et Path 2 présentent le lightpath primaire et de restauration qui correspondent respectivement aux scénarios A et B de la solution OpenFlow. Les nœuds du Path 1 sont Switch OpenFlow 1 → ROADM2 → ROADM3 → Switch OpenFlow 2 tandis que les nœuds du Path 2 sont Switch OpenFlow 1 → ROADM2 → ROADM1 → ROADM3 → Switch OpenFlow 2. Les nœuds du LSP sont CSA1 → ROADM2 → ROADM3 → CSA2. Les résultats expérimentaux montrent que pour les deux paths (Path 1 et 2), la solution OpenFlow

1. Le simulateur est dévelopé par l'équipe de recherche de notre laboratoire (OTN Lab) de l'UQAM. Il n'est pas le sujet de ce mémoire.

	Solution 1 : Extension du protocole OpenFlow				
	Contrôleur	Temps au niveau des switches			Totale (ms)
		ROADM2	ROADM1	ROADM3	
Path1	16	100	-	100	216
Path2	18	90	30	101	239
	Solution 2 : GMPLS				
	RSVP-TE	Temps au niveau des switches			Totale (ms)
		ROADM2	ROADM1	ROADM3	
LSP	130	110	-	100	340

TABLE 4.1: Temps expérimental d'établissement des liens

offre un temps d'établissement plus petit que la solution GMPLS. Ceci est dû essentiellement à la complexité de la technique GMPLS, surtout lorsqu'elle est utilisée comme un plan de contrôle unifié pour les réseaux IP/DWDM. Le nombre de messages de contrôle utilisés par cette technique est plus important que la solution OpenFlow dont le contrôle est centralisé dans un seul contrôleur externe. GMPLS utilise deux protocoles (OSPF-TE et RSVP-TE) pour l'établissement d'un lightpath. Les messages de signalisation et de réservation sont donc échangés par les différents VLSRs entre la source et la destination. Alors que pour la solution OpenFlow, le calcul se fait au niveau du module PCE et les messages de contrôle sont envoyés simultanément aux switches optiques.

4.3 Simulation

4.3.1 Introduction

Le tableau 4.2 présente les différents protocoles utilisés pour chaque solution.

L'objectif est la comparaison des performances des deux solutions implémentées en utilisant deux topologies de réseaux optiques réelles. La première topologie est le réseau optique de l'Amérique du Nord, NSF (National Science Foundation). La deuxième est celle du réseau optique européen (European Research Project Cost239). Pour cette simulation, nous avons

	Protocole utilisé		
	OpenFlow	OSPF-TE	RSVP-TE
Extension du protocole OpenFlow	Oui	Non	Non
GMPLS	Non	Oui	Oui

TABLE 4.2: Protocoles utilisés pour chaque solution

utilisé le résultat expérimental pour la création du lightpath par un agent OpenFlow (60 ms). Pour chaque nœud optique, on associe un agent OpenFlow et ces agents ont une connexion TCP à un contrôleur OpenFlow à travers un canal OpenFlow sécurisé. L'algorithme exécuté au niveau du contrôleur OpenFlow est présenté par la figure 4.1. Cet algorithme montre les décisions prises par le contrôleur quand il reçoit les différents messages OpenFlow. Pour l'agent OpenFlow, il suit l'algorithme présenté dans la figure 4.2. Cet algorithme montre les décisions à apprendre lors de la réception d'un message OpenFlow. Ces messages sont transformés à des commandes TL1 qui seront exécutées au niveau de la switch optique. Pour la simulation, le temps d'exécution des commandes TL1 sera émulé par le temps expérimental (60 ms).

4.3.2 La Topologie NSF (National Science Fondation)

La première topologie que nous avons utilisée avec notre simulateur est la topologie du réseau optique de l'Amérique du Nord (Figure 4.3). Cette topologie contient 14 nœuds optiques avec 21 liens, chacun possède 32 longueurs d'onde. Pour chaque nœud, nous avons associé un agent OpenFlow. Au total, 1000 requêtes sont générées selon un processus de Poisson et sont réparties uniformément entre toutes les paires de nœuds.
La simulation a été exécutée trois fois pour calculer le temps d'établissement du lightpath (Figure 4.4), le nombre des messages de contrôle au niveau du contrôleur OpanFlow (Figure 4.5), la probabilité de blocage (Figure 4.6) et le nombre de noeuds par requête (Figure 4.7).

La figure 4.4 montre bien que la première solution (Extension du protocole OpenFlow) a un temps d'établissement du lightpath plus petit (Courbe bleue) par rapport à la deuxième

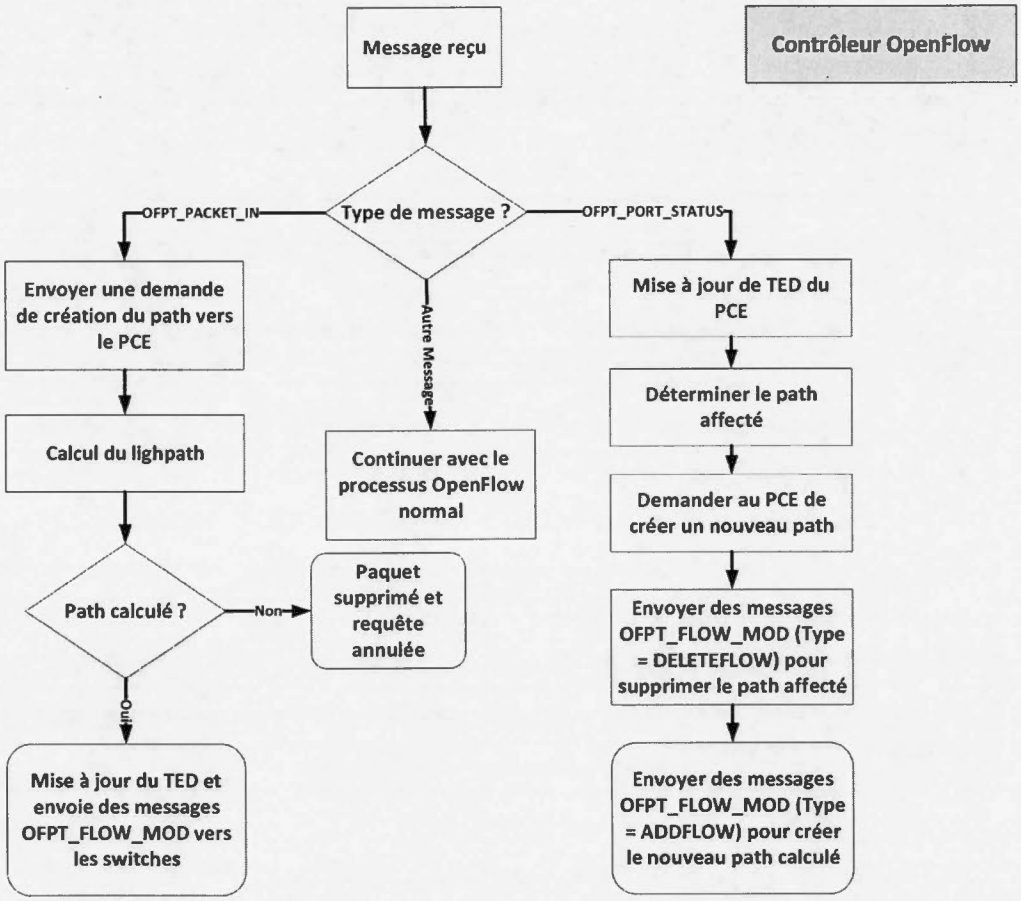


FIGURE 4.1: L'algorithme exécuté au niveau du simulateur pour simuler le contrôleur OpenFlow

solution basée sur GMPLS (Courbe rouge). Ce résultat est dû à la nature distribuée du plan de contrôle de GMPLS. La flexibilité du plan de contrôle basé sur GMPLS est faible car par exemple si on veut créer un lightpath, les messages de signalisation et de réservation doivent être échangés entre tous les VLSRs intermédiaires.

Cependant, GMPLS a tendance à diminuer le temps d'établissement du lightpath quand la charge du réseau augmente. Ceci parce que pour une grande charge de réseau, la longueur moyenne du path est plus petite. Comme le montre la figure 4.7, elle diminue de 3,6 à 2,7 nœud par requête. Le nombre de nœuds par requête diminue aussi pour la solution OpenFlow, mais

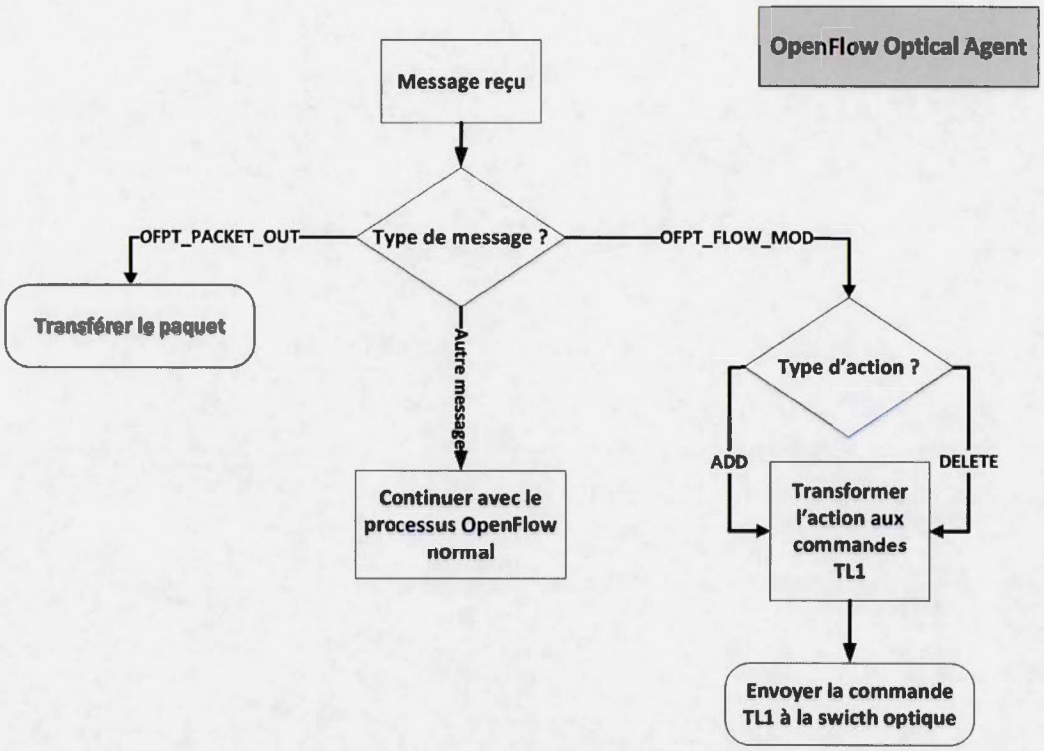


FIGURE 4.2: L'algorithme exécuté au niveau du simulateur pour simuler l'Agent OpenFlow

ceci n'affecte pas l'établissement du lighpath puisque les requêtes sont exécutées en parallèle. La figure 4.5 montre le nombre de messages de contrôle pour chaque solution. On voit bien que la deuxième solution a plus de trafic de contrôle. La différence est importante par rapport à la première solution. Ceci est dû essentiellement au fait que, pour le GMPLS, les messages de signalisation sont envoyés à tous les nœuds et que les messages de mise à jour des LSAs (Link State Advertisement) sont échangés entre eux chaque fois qu'il y'a un changement d'état des liens. Par contre, la solution basée sur OpenFlow offre plus de flexibilité sur le plan de contrôle puisque toutes les fonctionnalités sont intégrées dans un seul contrôleur OpenFlow. La figure 4.6 montre la probabilité de blocage. On remarque que les deux solutions ont presque les mêmes résultats. Cependant, la solution GMPLS présente une probabilité de blocage même avec une faible charge sur le réseau. Ceci est dû à la contention des longueurs d'ondes lorsque plusieurs messages RSVP-TE tentent de réserver la même ressource (lien ou

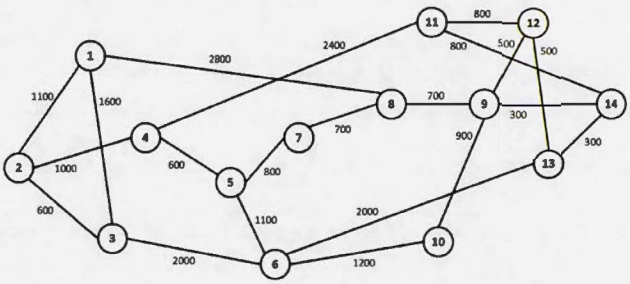


FIGURE 4.3: La topologie NSF

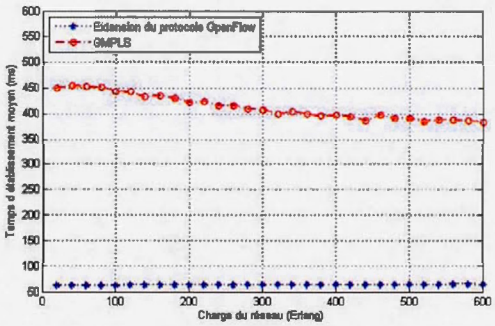


FIGURE 4.4: Topologie NSF : Temps d'établissement du lightpath (ms) vs la charge du réseau (Erlang)

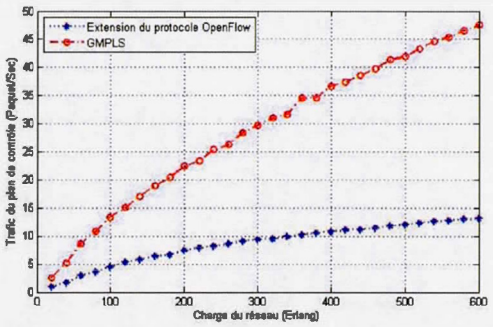


FIGURE 4.5: Topologie NSF : Nombre de messages de contrôle vs la charge du réseau (Erlang)

longueur d'onde).

4.3.3 La Topologie COST239

C'est la deuxième topologie que nous avons simulée (O'Mahony, 1996). Cette topologie contient 11 nœuds optiques avec 26 liens, chacun possède 32 longueurs d'onde (Figure 4.8). Comme dans le premier cas, 1000 requêtes sont générées selon un processus de Poisson et sont réparties uniformément entre toutes les paires de nœuds. La simulation a été exécutée trois fois pour calculer le temps d'établissement du lightpath (Figure 4.9), le nombre des messages de contrôle au niveau du contrôleur OpenFlow (Figure 4.10), la probabilité de blocage (Figure 4.11) et le nombre de noeuds par requête (Figure 4.12).

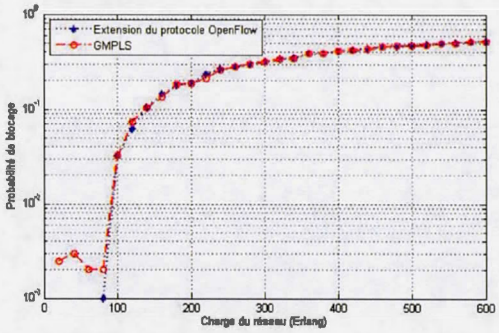


FIGURE 4.6: Topologie NSF : Probabilité de blocage vs la charge du réseau (Erlang)

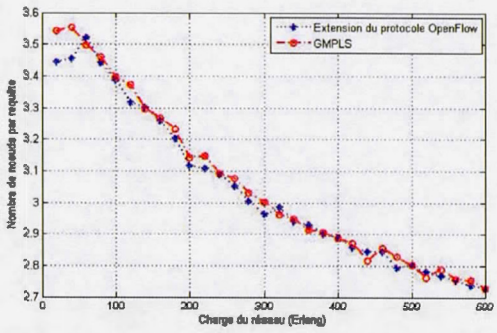


FIGURE 4.7: Topologie NSF : Nombre de noeuds par requête vs la charge du réseau (Erlang)

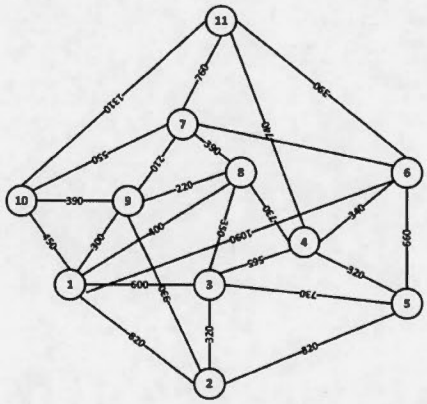


FIGURE 4.8: La topologie COST239

La figure 4.9 montre un résultat similaire à la première topologie. Elle confirme que la solution basée sur OpenFlow offre un temps d'établissement des liens plus faible que celle basée sur GMPLS. Comme dans le cas de la première topologie, pour la solution GMPLS, le temps d'établissement des liens diminue quand la charge du réseau augmente. C'est parce que pour une charge importante du réseau, la longueur moyenne d'un path est plus courte. Elle diminue de 2,8 à 2,3 (Figure 4.12). Pour la solution OpenFlow, le temps d'établissement des liens reste presque constant, car les messages d'établissement d'un lightpath sont envoyés par le contrôleur central en parallèle vers tous les nœuds.

La figure 4.10 montre les messages de contrôle pour chaque solution. Elle confirme le résultat

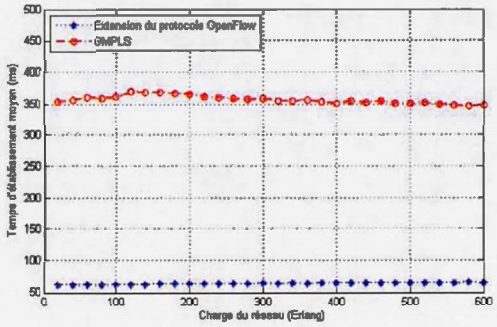


FIGURE 4.9: Topologie COST239 : Temps d'établissement du lightpath (ms) vs la charge du réseau (Erlang)

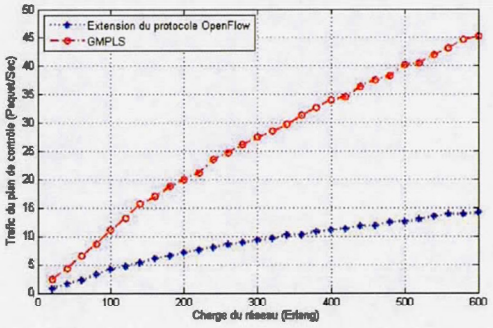


FIGURE 4.10: Topologie COST239 : Nombre de messages de contrôle vs la charge du réseau (Erlang)

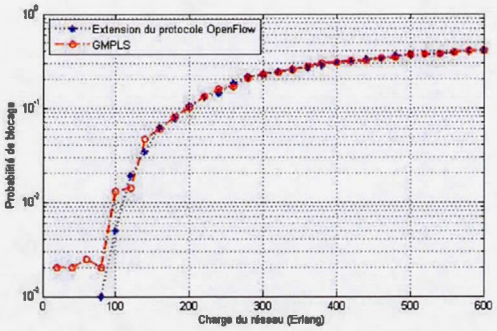


FIGURE 4.11: Topologie COST239 : Probabilité de blocage vs la charge du réseau (Erlang)

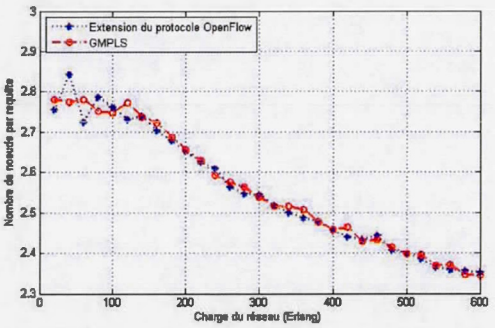


FIGURE 4.12: Topologie COST239 : Nombre de noeuds par requête vs la charge du réseau (Erlang)

obtenu pour la topologie NSF ; La solution OpenFlow a un trafic de contrôle moins important que pour la solution GMPLS.
La figure 4.11 montre la probabilité de blocage et confirme aussi le résultat obtenu pour le premier cas. Les deux solutions ont les mêmes valeurs de probabilité de blocage. Cependant que la solution GMPLS présente un ratio de blocage même avec une faible charge de réseau.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une étude comparative des performances des deux solutions implémentées. Nous avons commencé par les résultats expérimentaux qui montrent le temps d'établissement des liens. Ensuite, nous avons montré les résultats de simulation en se basant sur deux topologies de réseaux réelles. Les résultats obtenus montrent bien que la première solution basée sur OpenFlow permet d'améliorer la performance du plan de contrôle et de réduire le temps d'établissement du lightpath. Cependant elle ne possède pas la fonction de réservation des liens assurée par GMPLS, qui est une propriété importante surtout pour le domaine optique.

CHAPITRE V

CONCLUSION

Nous avons proposé deux solutions d'implémentation d'un plan de contrôle unifié pour gérer les couches IP et optiques simultanément. La première solution est basée sur le protocole OpenFlow, l'une des implémentations SDN la plus utilisée aujourd'hui. Étant donné que ce protocole est seulement mature pour les réseaux de commutation de paquets, nous avons proposé son extension pour qu'il puisse supporter les domaines optiques. Nous avons implémenté un ***OpenFlow Optical Agent*** qui permet d'assurer la communication entre le contrôleur OpenFlow et le nœud optique. Nous avons ajouté aussi le module ***PCE*** au contrôleur OpenFlow pour que ce dernier puisse établir des chemins optiques dans la nouvelle architecture convergente des deux réseaux. Dans la deuxième solution, nous avons utilisé GMPLS. Ce protocole présente une complexité de déploiement commerciale à cause du nombre important d'équipements de réseaux qui ne sont pas compatibles avec lui. Pour résoudre ce problème, nous avons adapté un software open source, ***DRAGON***, pour rendre nos switches optiques capables à opérer dans un réseau GMPLS.

Nous avons validé la faisabilité des deux solutions implémentées en utilisant plusieurs scénarios expérimentaux dans notre banc d'essai. Nous avons également comparé les performances des deux solutions en utilisant une simulation dans deux topologies de réseaux réelles. Les résultats de comparaison montrent bien que la première solution basée sur OpenFlow donne une meilleure amélioration des performances du plan de contrôle et offre un temps d'établissement des liens plus petit. Ceci revient essentiellement à l'utilisation d'un contrôleur centralisé qui a une vision globale sur toute la topologie du réseau. Cependant, l'avantage du protocole GMPLS est la réservation des liens et la possibilité de créer plusieurs LSPs dans une seule connexion physique. En effet, l'évolution technique du GMPLS à OpenFlow est

un processus d'évolution du plan de contrôle d'une architecture entièrement distribuée à une autre totalement centralisée.
Finalement, l'utilisation de ces deux solutions crée une interaction dynamique entre les réseaux de paquets et de transport ce qui revient à gérer un seul réseau convergent avec un plan de contrôle unifié. Ceci permet de réduire les couts d'exploitation, améliorer les performances du plan de contrôle et réduire le temps d'établissement des liens.

Le travail de ce mémoire ouvre de nouvelles voies et des perspectives futures. Un choix logique serait d'introduire une technologie d'interfonctionnement entre les protocoles OpenFlow et GMPLS. Un tel choix permet d'exploiter les avantages de GMPLS pour contrôler efficacement la couche optique et profiter de la flexibilité du contrôleur OpenFlow pour coordonner et faire interagir les couches IP et optiques dynamiquement.

CHAPITRE VI

PUBLICATIONS

6.1 Article accepté dans la conférence ICTON

Enhancing Open Flow to Enable Multilayer Networking

Karim Idoudi, Halima Elbiaze

Université du Québec à Montréal, Case postale 8888, succ. Centre-ville Montréal (QC) H3C3P8, Canada

Tel: (514) 987 3000 (8485), Fax: (514) 987 8477, e-mail: elbiaze.halima@uqam.ca

ABSTRACT

The convergence of packet and circuit networks offers significant advantages, in terms of cost, energy efficiency and network performance. So there have been many attempts to unify the control and management of these networks. Optical Software Defined Networks is revealing new opportunities and challenges for optical equipments vendors and service providers inspired by recent developments in Software Defined Networking. Extending Open Flow to the optical (transport) layer will enable open programmability and pave the way to multidomain and multilayer network integration, which is driven by convergence of packet and optical technologies. In this paper we propose a simple way to unify both types of network using Open Flow. We exploit the fact that Open Flow provides a common API to the underlying hardware, and allows all of the routing, control and management to be defined in software outside the datapath.

Keywords: Software Defined Networking, Open Flow, Optical Networks.

1. INTRODUCTION

Today, IP and transport networks do not interact. The main reason behind this is that they are two very different networks with different architectures, switching technologies, and control mechanisms. The former is automated, dynamic, lightly managed but ultimately best effort, while the latter is largely manual, semi-static, tightly managed and very reliable.

In this paper, we propose unifying architecture and control plane for packet and circuit networks, based on Software Defined Networking (SDN) and Open Flow. Briefly, SDN principles can be summarized as follows: separation of data and control paths in packet and circuit networks; flow based datapath where flows (not packets) are the fundamental unit of control; a rich API called Open Flow(OF) into the switch flow tables for programming and controlling both packet and circuit datapaths; a logically-centralized Controller, running a network operating system, which in-turn provides another API for programming networking applications; and lastly, a means to provide network virtualization by slicing the network and isolating the slices, so that experimental slices can run in parallel to production slices, and backward compatibility is maintained with today's networks.

There have been several attempts [1] [2] and proposals to control both circuit switched and packet switched networks using the Open Flow protocol. The last year, teams from ADVA Optical Networking and the University of Essex unveiled an SDN test bed common Open Flow control capable of dynamically operating both packet and wavelength switches. As part of the OFELIA projects, this was the industry's first glimpse into the potential for true network virtualization across multiple network layers.

Section II provides background on Open Flow architecture [3], describing the separation of data and control planes. In Section III we explain the mechanism to unify packet and circuit networks and we present prototype unified Open Flow architecture. Section IV describes a simple proof-of-concept demonstration of Open Flow unified control of packet and circuit switches.

2. OPEN FLOW

We briefly outline the main characteristics of Open Flow. A more detailed and exhaustive documentation is available in the Open Flow white paper [4] and in the Open Flow specification [5].

In a traditional network, the data path and the control path occur on the same device (switch, router). Open Flow separates these two functions; Open Flow ***switches*** perform the data plane function and Open Flow ***controllers*** implement the control plane intelligence and communicate with the Open Flow switch via the Open Flow protocol which defines the set of messages exchanged. The Controller is responsible of the management and configuration of the Open Flow Switch (Fig. 1).

The basic idea is simple; Open Flow exploits the fact that most modern Ethernet switches and routers contain flow-tables. While each vendor's flow table is different, Open Flow provides an open protocol to program these tables by identifying an interesting common set of functions that run in many switches. This provides a standardized way to allow researchers to run experiments, without requiring vendors to expose the internal working of their network devices

The Open Flow switch communicates with the controller over a secure connection using Open Flow protocol. An Open Flow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an Open Flow Channel that is connected to an external controller. Each Flow table in the Switch

978-1-4799-0683-3/13/$31.00 ©2013 IEEE

contains a set of flow entries; each flow entry consists of match fields, counters and a set of instructions to apply to matching packets.

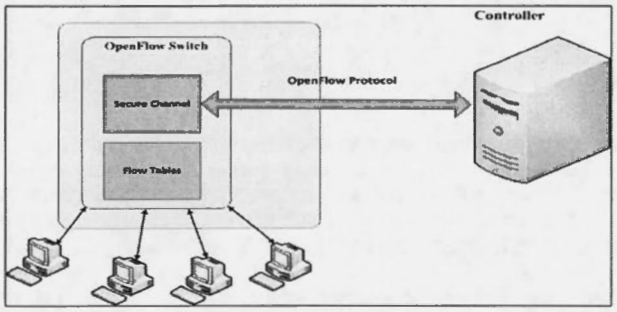


Figure 1. Open Flow network.

2.1 Match field

Match field is a field against which a packet is matched, including packet headers, the ingress port, and the metadata value.

On receipt of a packet, an Open Flow Switch performs the functions shown in Figure2. The Switch starts by performing a table lookup in the first flow table, and, based on pipeline processing, may perform table lookup in other flow tables. Match fields used for table lookups depend on the packet type (Ether dst, Ether type, VLAN id, IPv4 src, IPv4 dst, etc...).

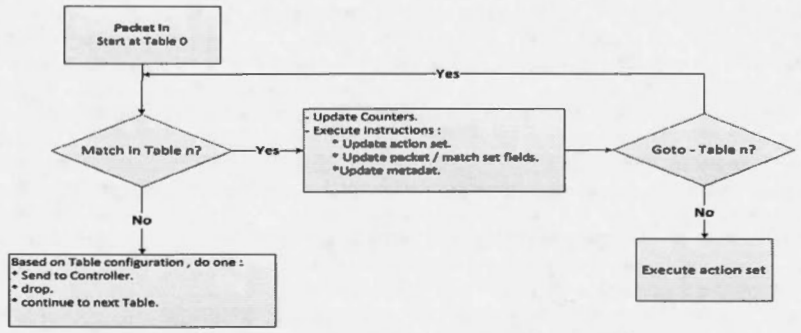


Figure 2. Packet flow through an Open Flow switch.

2.2 Instructions

Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and pipeline processing. Supported instructions include: *Apply - Actions, Clear – Actions, Write – Actions, Write – metadata, Goto – Table*. An action set is associated with each packet. This set is empty by default and it is carried between flow tables. Supported actions include: *copy TTL, pop, push, decrement TTL*, etc...

Counters are used to update for matching packets.

3. OPENFLOW UNIFIED ARCHITECTURE

The current implementations of the Open Flow protocol are mainly addressing packet switched domains (i.e. Ethernet LANs). But, its extension to support circuit and especially optical networks can provide a new framework for evolving carrier grade networks. Enabling SDN via Open Flow at the optical layer will allow decoupling network control and management functionalities from the network elements and their vendors. It will also support the integration of electronic packet and circuit switched networks for access and core network segments.

The Open Flow controller is a server, which has the capabilities to host different network management and control applications to effectively manage the network in a centralized or distributed way. Separation between the control and data plane and the capability to process packet /circuit traffic as flows make Open Flow protocol a single control layer for both packet and circuit networks. Moreover, this control plane integration will promote the convergence of their respective management plane too. A unified Open Flow based control architecture is depicted in Figure 3.

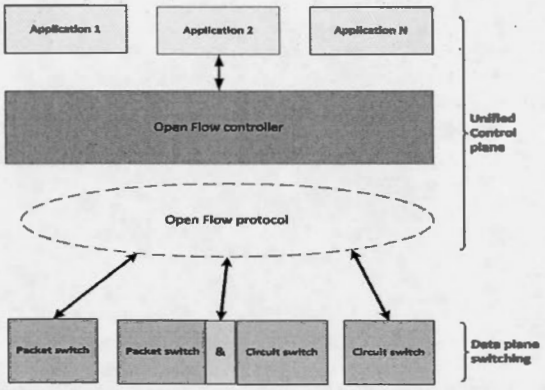


Figure 3. A unified control plane based on Open Flow protocol, controller and network applications.

Open Flow ***protocol***, Open Flow ***controller*** and network ***applications*** build-up a unified control plane. For the optical domain, we can use GMPLS (Generalized Multi Protocol Label Switching) as a control plane framework [6].

GMPLS is used for managing physical path and core tunneling technologies of the Internet and Telecom service providers. The GMPLS architecture extends MPLS (Multi Protocol Label Switching) to encompass control and management of time – division, wavelength, and spatial switching. The GMPLS control plane, due to its support for various optical transport technologies as well as its capability for dynamic and demand provisioning, is widely being considered by operators as the control plane of their next generation core optical networks. Figure 4 shows the architecture of the proposed integrated Open Flow – GMPLS control plane. It utilizes an extended packet switch Open Flow controller integrated with a GMPLS control plane in an overlay model.

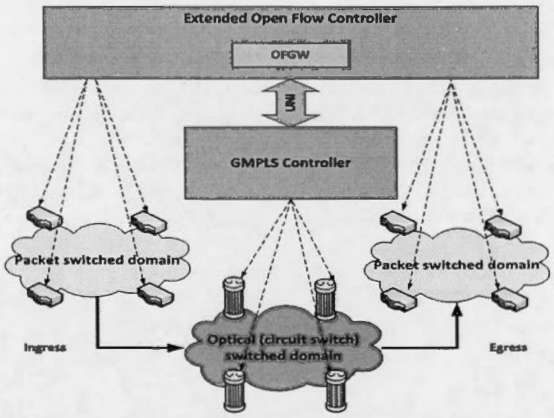


Figure 4. Integrated GMPLS-Open Flow control plane.

In this architecture, the packet switched domain and the GMPLS optical domain are controlled by an extended Open Flow controller. The extended controller provides functionality for requesting optical connectivity from the optical domain. It is assumed that the edge packet switch, which interconnects the optical domain to the packet switched domain, is equipped with tunable WDM interfaces.

Initially the flow tables of the switches are empty. The first packet of the request will be forwarded to the extended Open Flow controller. Based on the destination address of the request, the extended Open Flow controller identifies the end point of the flow. In this context, the Optical flows can be defined as those flows, which are destined to another one through the optical (circuit switch) domain (i.e., from one packet switch domain to another one through the optical core network). In fact of the optical flows, the extended Open Flow controller identifies the end point of the light path in the circuit switch domain and requests (via OFGW) a light path GMPLS control plane. The interface for this request is UNI. Once the light path is established the extended controller updates the flow table of the ingress and egress switches, finalizing the establishment of an end – to – end optical flow path. The extended Open Flow controller also maintains a list of established light paths in terms of light path identifiers. After this process, the new optical flow entry will be inserted to the flow table of the

ingress switch, which instructs it to put the optical flows to the proper port and wavelength in order to traverse the circuit switched domain.

4. EXPERIMENTATION TESTBED

We demonstrate an Open Flow unified control in our testbed with the help of Open Flow enabled packet switches based on the NetFPGA [7] platform and Cisco ONS 15454 optical switches (Figure 5).

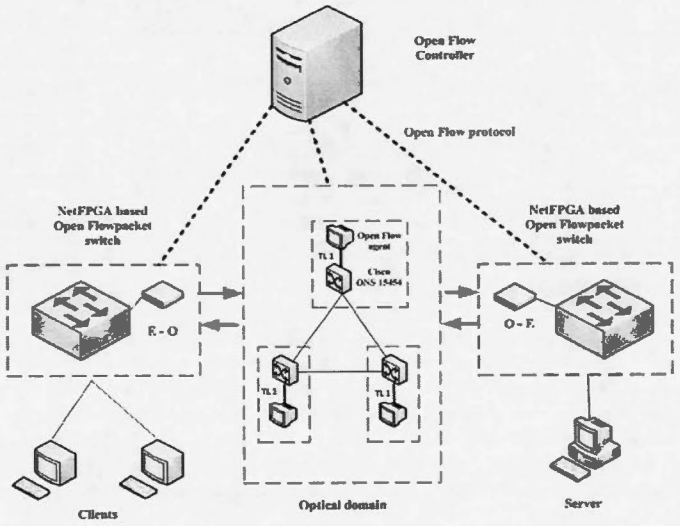


Figure 5. Lab demonstration.

For the optical switch, a Virtual Open Flow Switch (VOFS) is implemented in an embedded Linux PC platform (OpenFlow agent). Virtual Ethernet interfaces (veths) are introduced in the VOFS to reflect the physical ports of the Optical Switch in a one-to-one correspondence, as shown in Figure 6.

By using this approach, the VOFS obtains a virtualized view of the physical structure of the optical switch control channels. This PC use TL1 commands to control the optical switch. Together the PC and the agent and Optical Switch can be regarded as an Open Flow enabled circuit switch [8].

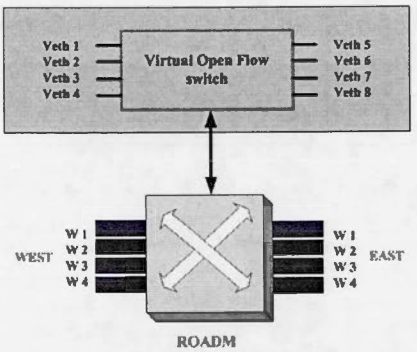


Figure 6. Open Flow-based optical node.

When a new flow arrives to the NetFPGA, the header of the new packet is sent to the controller that will calculate the path based on his routing algorithm and his Traffic Engineering (TE) algorithm. Then the controller will send the flow entries to the optical switches (NetFPGAs and VOFSs) to create the complete path which includes the flow entries in the packet switches and the light-path in the optical switches. Finally, the flow can use this path to reach to their destination.

5. CONCLUSIONS

In this paper we describe how Open Flow could be used to unify the control of packet and circuit networks. We believe the SDN/Open Flow architecture and control plane can create mutually beneficial interaction between IP and Transport networks by enabling new capabilities at the packet-circuit interface.

Our future work will involve expanding the testbed to demonstrate network applications and we also aim to demonstrate unified virtualization of the data plane.

ACKNOWLEDGEMENTS

This work is funded by CFI (Canada Foundation for Innovation).

REFERENCES

[1] S. Das, G. Parulkar, N. McKeown, "*Unifying Packet and Circuit Networks*", Below IP Networking (BIPN), November 2009.

[2] Packet and Circuit Convergence (PAC.C) with OpenFlow wiki: http://www.openflow.org/wk/index.php/PAC.C

[3] N. McKeown, et. al., "OpenFlow: Enabling Innovation in Campus Networks", in IEEE/ACM SIGCOMM CCR, Vol. 38, Issue 2, March 2008.

[4] http://www.openflow.org/documents/openflow-wp-latest.pdf

[5] http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[6] Azodolmolky, S., Nejabati, R., Escalona, E., Jayakumar, R., Efstathiou, N., & Simeonidou, D. "*Integrated OpenFlow-GMPLS control plane: an overlay model for software defined packet over optical networks*". In OSA European Conference and Exposition on Optical Communications., 2011, September.

[7] http://netfpga.org/

[8] Lei Liu; Dongxu Zhang; Tsuritani, T.; Vilalta, R.; Casellas, R.; Linfeng Hong; Morita, I.; Hongxiang Guo; Jian Wu; Martinez, R.; Munoz, R., "Field *Trial of an OpenFlow-Based Unified Control Plane for Multilayer Multigranularity Optical Switching Networks*," in IEEE Journal of Lightwave Technology, vol.31, no.4, pp.506,514, Feb.15, 2013.

6.2 Article accepté dans la conférence IEEE GLOBECOM

Software-Defined DWDM Optical Networks: OpenFlow and GMPLS Experimental Study

M. Bahnasy, K. Idoudi and H. Elbiaze
Université du Québec à Montréal
Email: elbiaze.halima@uqam.ca

***Abstract*—Finding an effective and simple unified control plane (UCP) for IP/Dense Wavelength Division Multiplexing (DWDM) multi-layer optical networks is very important for network providers. Generalized Multi-Protocol Label Switching (GMPLS) has been in development for decades to control optical transport networks. However, GMPLS-based UCP for IP/DWDM multi-layer networks is extremely complex to be deployed in a real operational products because still there are a lot of non-capable GMPLS equipments. DRAGON (Dynamic Resource Allocation via GMPLS Optical Networks) [1] is a software that solves this issue making these equipments capable for working in a GMPLS network. On the other hand, OpenFlow (OF), one of the most widely used SDN (Software Defined Networking) implementations, can be used as a unified control plane for packet and circuit switched networks [2].**

In this paper, we propose and experimentally evaluate two solutions using OpenFlow to control both packet and optical networks (*OpenFlow Messages Mapping* and *OpenFlow Extension*). These two solutions are compared with GMPLS-based UCP. The experimental results show that the *OpenFlow Extension* solution outperforms the *OpenFlow Messages Mapping* and GMPLS solutions.

***Keywords*—Optical Network; GMPLS; DRAGON; Software Defined Networking; OpenFlow**

I. Introduction

Currently, IP and optical layers operate separately without dynamic interaction which leads to high operational cost, low network efficiency, and long processing latency for end-to-end path provisioning. The main reason behind these limitations is that they are two different networks with different architectures, switching technologies, and control mechanisms. Therefore, a unified control plane (UCP) for both IP and optical layers, as one of the key challenges for the network carriers, is very important to address the aforementioned issue. GMPLS, a relatively mature control plane technique for optical transport networks, has been proposed as a solution for UCP [3]. But due to the distributed nature, the number of protocols, and the interactions between different layers, the GMPLS-based UCP is overly complex[4], [5]. Moreover, the implantation of this technology is difficult because still there are a lot of non-capable GMPLS equipments. DRAGON [6], is a software that solves this problem using SNMP (Simple Network Management Protocol) to control these equipments and making them capable for working in a GMPLS network. In this paper, we use this software and adapt it to operate with our optical switch (Cisco ONS 15454 [1]).

On the other hand, we propose SDN [7] as a promising solution for a UCP. Generally, the SDN technology separates the control and data planes so that we can introduce a new functionality by writing a software program, running within an external controller that manipulates the logical map of the network. This provides the maximum flexibility for the operator to control different types of network, and match the carriers preferences. One of the widely used SDN implementations is OpenFlow [8]. OpenFlow protocol is mature for L2/L3 packet switching networks, but still at a starting stage for wavelength-switched optical networks. So, it needs some extensions to be able to support the optical domain.

Some efforts have been done to present OpenFlow-based UCP to control packet and circuit switches. Most notably, PAC.C [2] has experimented with alternative approaches. Other papers [9], [10], [11] have presented similar work as PAC.C by providing an experimental study or a Proof-of-Concept to support the using of OpenFlow as a unified control plane. However, [12] presents a comparison study between OpenFlow and GMPLS solutions based on a simulation. In this paper, we propose two approaches based on OpenFlow protocol to control both optical and electrical networks. Then we experimentally compare these two solutions with a real implementation of GMPLS approach. To the best of our knowledge, this is the first work who considers both OpenFlow and GMPLS UCP solutions, and compare them via testbed experimentation. We conduct a real case study of implementing end-to-end lightpath and a lightpath restoration by establishing a dynamical configured backup lightpath.

The first solution is ***OpenFlow Messages Mapping***; we map the OpenFlow standard messages into equivalent optical channel requests, without modifying the OpenFlow protocol. The second one is ***OpenFlow Extension***; new messages have been added to the OpenFlow protocol in order to support the circuit switching. The proposed solutions are implemented in a testbed to demonstrate their effectiveness, as well as GMPLS-based approach. For both solutions, we implement an **OpenFlow Optical Agent** to translate the OpenFlow messages to be executed on the optical switches. Moreover, a **Path Computation Element (PCE)** module is added to the OpenFlow controller as a network application in order to control the optical domain.

The remaining of this paper is organized as follows; Section II describes how can OpenFlow define a unified control plane for both IP and optical networks and the iplementation of the proposed solutions (***OpenFlow Messages Mapping*** and ***OpenFlow Extension***). Section III presents the GMPLS-based UCP approach and the deployment of this protocol in our testbed. In particular, we explain the adaptation of DRAGON software for our ROADM (CISCO ONS 15454). Section IV

[1]ROADM : Reconfigurable Optical Add-Drop Multiplexer

presents the different experimental scenarios for each solution and the comparative results with GMPLS. Concluding remarks are eventually given in section V.

II. OpenFlow-based Unified Control Plane

A. Overview

We briefly outline the main characteristics of OpenFlow. A more detailed and exhaustive documentation is available in the OpenFlow white paper [13] and in the Open Flow specification [14]. OpenFlow is an open standard that was developed several years ago at Stanford University in order to enable researchers to run experimental new protocols and technologies on real networks, without disrupting the existing traffic or network availability [15]. In a traditional network, the data path and the control path occur on the same device (switch, router). OpenFlow separates these two functions; OpenFlow switches perform the data plane functions and OpenFlow controller implements the control plane intelligence and communicates with the OpenFlow switch via the OpenFlow protocol.

An OpenFlow switch consists of one or more flow tables and group tables, which perform packet lookups and forwarding, and a secure channel that is connected to an external controller. Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters and a set of instructions to apply to matching packets.

OpenFlow advocates the separation of data and control planes for circuit and packet networks, as well as the treatment of packets as part of flows, where a packet flow is defined as any combination of L2/L3/L4 headers. This, together with L1/L0 circuit flows, provides a simple flow abstraction that fits well with both types of networks. Hence, OpenFlow presents a common platform for the control of the underlying switching hardware, that switches flows of different granularities, while allowing all of the routing, control and management to be defined in software outside the datapath, in the OpenFlow controller (Figure 1).

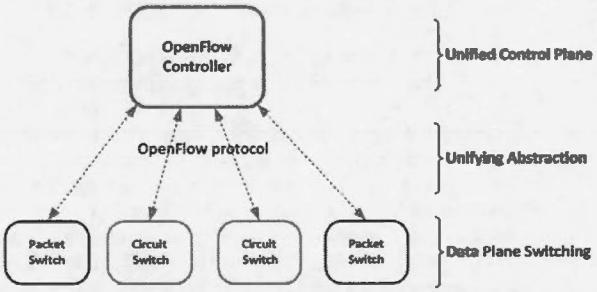


Fig. 1: Unified architecture of a converged Packet-Circuit network

B. OpenFlow Messages Mapping and OpenFlow Extension

This paper proposes two solutions using OpenFlow protocol as a unified control plan for both optical and electrical domains (***OpenFlow Messages Mapping*** and ***OpenFlow Extension***). For both solutions, we implement an OpenFlow Optical Agent to translate the OpenFlow messages to its proper TL1 (Transaction Language 1) commands [16] to be executed on the optical switch using telnet channel. A Path Computation Element (PCE) module is added to the OpenFlow controller as a network application (Figure 2). Upon request arrival, PCE calculates the corresponding lightpath and sends the cross-connection messages to involved ROADMs. In the next sections, we describe ***OpenFlow Messages Mapping*** and ***OpenFlow Extension*** solutions.

1) ***OpenFlow Messages Mapping:*** In this solution, the OpenFlow standard messages are used without any modification. The OpenFlow messages are mapped into optical switch commands. In this approach, the *OFPT_FLOW_MOD* message of type *OFPFC_ADD* is mapped into *ENT-OCHNC* TL1-command to create a lightpath channel. The *OFPT_FLOW_MOD* message of type *OFPFC_DELETE* is mapped into *DLT-OCHNC* TL1-command to delete a lightpath channel. When the agent receives *OFPT_FEATURES_REQUEST* message, it encapsulates the emulated port information into *OFPT_FEATURES_REPLY* message. Finally the agent reads periodically the ROADM events (using *RTRV-ALM-ALL* TL1-command) and if it finds any critical alerts, it creates *OFPT_PORT_STATUS* message and forwards it to the controller.

2) ***OpenFlow Extension:*** In this solution, OpenFlow messages are extended and new messages are added. The new messages specification [17] allows the controller to distinguish between the circuit-switching and the packet-switching networks. For example, *OFPT_FEATURES_REPLY* message is extended by adding extra information about the circuit-switching ports. To send an optical cross-connect information, a new match structure called *ofp_connect* is presented. Multiple ports can be cross-connected by a single structure. This structure is added to the newly defined message called *OFPT_CFLOW_MOD*. Finally when the state of a port changes, the OpenFlow Optical Agent sends a new defined message called *OFPT_CPORT_STATUS*.

C. OpenFlow Optical Agent

As mentioned above, the main role of the OpenFlow Optical Agent is to translate the optical channel requests and OpenFlow messages into TL1 commands to be executed on optical nodes (Figure 2).

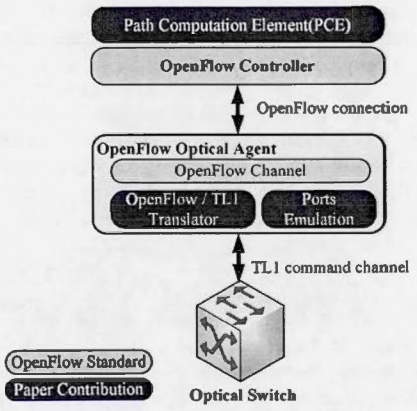


Fig. 2: OpenFlow Optical Agent intercations

This agent is associated to each optical node and acts as a virtual switch. It consists of ; (i) *OpenFlow*

Channel to communicate with the OpenFlow controller, (ii) *OpenFlow/TL1 Translator* to convert OpenFlow messages into TL1 commands, and (iii) *Ports Emulation* module to emulate the optical node ports and send the port status information to the controller. This information is used by the controller to update ports database and to calculate the lightpath [1].

D. Path Computation Element (PCE)

The PCE implements an algorithm to establishe lightpaths between source-destination pairs in order to create a fully connected logical topology [18]. A Traffic Engineering Database (TED) is created to save the network topology information. As the OpenFlow controller has a centralized management, the TED will be updated in case of lightpath creation/release and ports status change. Two modules are proposed to implement the PCE; Executor and Optical Switch Adapter (Figure 3).

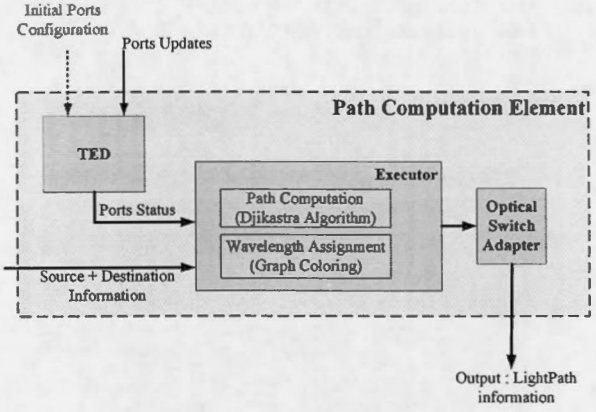


Fig. 3: Path Computation Element workflow

1) Executor: This module ensures the avoidance of using one wavelength more than once in the same fiber. Each wavelength carries traffic between a pair of source and destination. Therefore, multiple wavelengths are reserved in a single strand of fiber for establishing multiple lightpath through one fiber. These connections between source/destination nodes in DWDM networks are performed in two steps:

- **Routing:** We use *Dijkstra Algorithm* in order to find the shortest path between each node pair. In our case, we are interested in a network topology composed of OpenFlow switches and ROADMs.
- **Wavelength Assignment:** Once the lightpath routes are determined, the wavelength assignment problem can be represented as a graph coloring problem. Each lightpath corresponds to a node in wavelength assignment graph, and two nodes are set as neighbors only if the respective lightpaths share at least one common link.

2) Optical Switch Adapter: Each ROADM consists of a set of cards and each card contains a set of configured ports [19]. ROADM edges are connected to OpenFlow switches via WSS (Wavelength Selective Switch) and DMX (Channel Demultiplexer) cards, whereas ROADM core interfaces are interconnected via LINE cards. Two fibers are used for the bidirectional connection between two ROADMs. These specifications lead us to add this module.

[1]Ports discovery is out of scope in this paper

III. GMPLS-based Unified Control Plane

A. Overview

Actually, there are still a lot of non capable GMPLS equipments. DRAGON software solves this problem in the Ethernet networks using SNMP to adapt these equipments to GMPLS control plane. The DRAGON project studies and develops an open source software to enable dynamic provisioning of network resources on an interdomain basis across heterogeneous network technologies. The project enables the communication between networks of different types through the GMPLS control suite. For its implementation, DRAGON deploys the IP network infrastructure and creates a GMPLS capable optical core network to allow dynamic provisioning of deterministic network paths in direct response to end-user requests, spanning multiple administrative domains. Optical transport and switching equipments acting as Label Switching Routers (LSRs) provide deterministic network resources at the packet, wavelength, and fiber cross-connect levels.

B. DRAGON Control Plane Components

DRAGON software is thought to work like control plane within a GMPLS network. The control plane architecture consists of two basic elements [2] : The Client System Agent (CSA) and Virtual Label Switch Router (VLSR).

1) CSA (Client System Agent): The CSA is a software that runs on (or on behalf of) any system which terminates the data plane (traffic engineering) link of the provisioned service. This is the software that participates in the GMPLS protocols to allow for on demand end-to-end provisioning from client system to client system. A CSA can be a host, a router, or any networked device.

2) VLSR (Virtual Label Switch Router): GMPLS has not yet been implemented on large a scale. There are still a lot of non GMPLS capable switches in use. To overcome this limitation, the DRAGON protocol suite uses the VLSR. A VLSR is used to control different kinds of switches like for instance Ethernet, TDM or Optical switches. What a VLSR does besides participating in the GMPLS protocols is translating GMPLS commands into switch specific commands like SNMP. By the use of these commands, a VLSR can control the switch and for example set a switch port in the specific VLAN. To communicate with other VLSRs and CSAs, a VLSR uses the routing protocol OSPF-TE (Open Shortest Path First-Traffic Engineering) and path signaling protocol RSVP-TE (Resource Reservation Protocol-Traffic Engineering). A VLSR uses OSPF-TE to get familiar with the control plane network and to inform the VLSRs and CSAs in the control plane about the TE network links. A VLSR uses the OSPF-TE LSAs (Link State Advertisements) to send information about the TE links.

[2]The informations found in this section is based on the Sara Project documentation produced by the RFC 3945 [20]

Information that could be send over the control plane is information about upcoming and down going LSPs (Label Switched Paths). The OSPF-TE works with two daemons called OSPFD and zebra. Zebra, or GNU Zebra [21], is routing software for managing TCP/IP based routing protocols like RIP, BGP and OSPF. The DRAGON software extends the OSPF routing daemon with Traffic Engineering informations like bandwidth, WDM and TDM used by GMPLS. A VLSR uses RSVP-TE for signaling and setting up LSPs within the GMPLS network. The RSVP-TE protocol originates from the Technische Universitt Darmstadts KOMRSVP [22]. The DRAGON software extends the KOM-RSVP signaling protocol with support for RSVP-TE, GMPLS, Q-Bridge, SNMP and VLAN control.

C. Adapting VLSR for Cisco ONS 15454

The DRAGON software suite is being developed under the GNU General public license [23]. The source code can be viewed, changed for own use. The latest version of the software suite can be downloaded at [24]. In order to install the DRAGON software, the VLSR implementation guide has been followed [25].
By default, the VLSR PC uses SNMP RFC 2674 to communicate with switch. To manage and control the Cisco ONS 15454, we use TL1 commands. Thus, we implement an SNMP/TL1 Gateway that acts as a proxy to adapt the VLSR software with Cisco ONS 15454 specification (Figure 4). As shown in figure

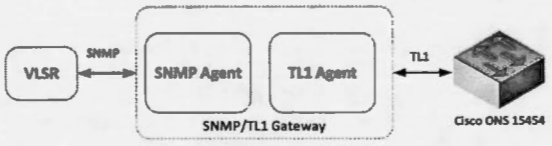


Fig. 4: SNMP/TL1 Gateway

4, the SNMP/TL1 Gateway is composed of two modules:

- **SNMP Agent:** Using *snmp4j* [26] open source Java library, we have developed an SNMP agent. It provides functions to receive and send SNMP PDUs (Protocol Data Unit).
- **TL1 Agent:** Using the *iReasing* [27] TL1 API, we have developed a TL1 based management application that communicates with the Cisco ONS 15454. Its main function is to map the SNMP messages into TL1 commands to set-up configurations in Cisco ONS 15454.

IV. Experimental Setup

In this section, we first present the OpenFlow experiments followed by the GMPLS ones. Then we discuss the experimental results in order to evaluate and compare the OpenFlow solutions with GMPLS.

A. OpenFlow Experiments

Two experiments are conducted to demonstrate the efficacy of our proposed solutions. The first experiment consists of creating end-to-end lightpath while the second experiment performs a backup restoration lightpath when failure occurs on the primary lightpath.

1) Testbed Setup: The architecture of our testbed is depicted by figure 5. It consists of two clients A and B, which are connected directly to OpenFlow switches 1 and 2, respectively. Each switch is connected to an Electrical/Optical converter. These converters are connected to DWDM optical network composed of three Cisco ROADM optical switches (Cisco ONS 15454). Each ROADM is controlled by an OpenFlow Optical Agent. The OpenFlow optical agents and the OpenFlow switches are connected to an OpenFlow controller over an OpenFlow channel.

2) Scenario 1: End-to-End Lightpath Setup: As shown in Figure 5, a data flow sent from Client A to Client B arrives at OpenFlow switch1. When the OpenFlow switch1 does not find any flow entry that matchs with this flow, it encapsulates the first flow packet in an OFPT_PACKET_IN message and forwards it to the Controller. Then the controller uses the PCE to calculate the lightpath, and creates the lightpath by sending *OFPT_FLOW__MOD* message (***OpenFlow Messages Mapping*** solution) or *OFPT_CFLOW__MOD* message (***OpenFlow Extension*** solution) to the switches. The connection is established between the two clients following steps A1, A2, A3, A4, A5, A6, and A7 (Figure 5). The wireshark screenshot presents the exchanged messages during this scenario (Figure 6).

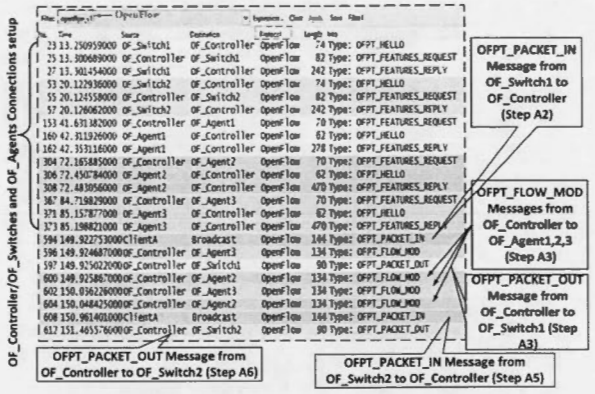
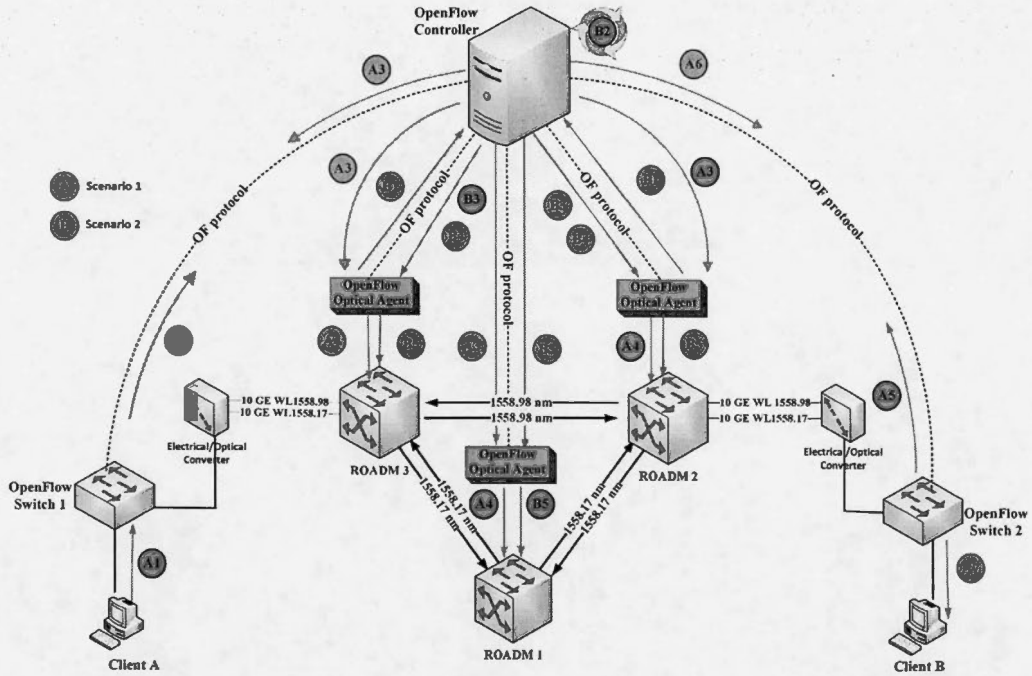


Fig. 6: OpenFlow Scenario1 : Wireshark screenshot

3) Scenario 2: Shared Optical Restoration: This scenario demonstrates how OpenFlow controller acts when a link failure occurs. The path deletion is performed by the controller using *OFPFC_DELETE* message. Figure 5 shows the steps that are executed in this scenario (B1, B2, B3, B4, and B5). The wireshark screenshot presents the exchanged messages during this scenario (Figure 7).

B. GMPLS Experiments

To experiment GMPLS, we construct a transparent optical network testbed with two ROADMs (Figure 8). In this infrastructure, the control plane consists of two CSAs and two VLSRs. The CSAs and the VLSRs are connected via the switch hub. GRE (Generic Routing Encapsulation) tunnels are created between the CSAs and the VLSRs and between the VLSRs themselves to exchange RSVP-TE and OSPF-TE messages. The SNMP/TL1 Gateway has a connection with the switch hub to allow SNMP management by the VLSRs.



A data flow sent from client A to client B arrives at OpenFlow switch 1.

OpenFlow switch 1 does not find a flow entry in its flow table to forward this flow, so it encapsulates the first flow packet in a OFPT_PACKET_IN message and forwards it to the controller.

The controller calculates the path from Client A to Client B, and sends OFPT_PACKET_OUT message to the OpenFlow switch 1 . The controller sends also OFPT_FLOW_MOD messages (*OpenFlow Messages Mapping* solution) or OFPT_CFLOW_MOD message (*OpenFlow Extension* solution) to the Optical OpenFlow agents in order to create the lightpath.

When OpenFlow optical agents receive this message, they translate it into the appropriate TL1 commands and send it to the ROADM switches.

After creating the lightpath, the data flow traverses until OpenFlow switch 2. When the flow is received by OpenFlow switch 2, if the switch does not find a flow entry in its flow table to forward this packet, it sends a OFPT_PACKET_IN message to the controller requesting an action for this flow.

The controller sends a OFPT_PACKET_OUT message to OpenFlow switch 2 to forward this packet to client B.

OpenFlow switch 2 forwards this packet to client B.

When the interconnection between the ROADM 3 and ROADM 2 fails, both OpenFlow optical agents corresponding to these Optical Switches read the alarms of the optical switches . Then they send OFPT_PORT_STATUS Messages to the controller about the port status update .

OpenFlow controller calculates alternative lightpaths to the existing failed lightpaths.

The controller sends OFPT_FLOW_MOD (type=ADD FLOW) messages (*OpenFlow Messages Mapping* solution) or OFPT_CFLOW_MOD message (*OpenFlow Extension* solution) to the optical switches to create new lightpath. In this case, a new lightpath is established from ROADM 2 to ROADM 3 via ROADM 1 on a different wavelength (1588.17nm).

The controller sends another OFPT_FLOW_MOD (Type=OFPFC DELETE) messages (*OpenFlow Messages Mapping* solution) or OFPT_CFLOW_MOD message (*OpenFlow Extension* Solution) to the optical switches which are associated with old lightpath to delete the primary lightpath.

When the OpenFlow Optical agents receive these messages, they translate it into the appropriate TL1 commands and send it to the optical switches.

Fig. 5: Network configuration and exchanged messages during the OpenFlow experiments

It translates SNMP messages to TL1 commandes in order to configure the ROADMs. In the SNMP/TL1 Gateway machine, we installed two virtual machines. Each one listens to a VLSR on port 161 and controls one ROADM. Using wireshark capture in VLSR2 (Figure 9 (a)) and VLSR1 (Figure 9 (b)), we explain the GMPLS signaling to create an LSP from CSA2 to CSA1.

CSA2 sends RSVP_PATH message to VLSR2 with the destination set to the target CSA1. Both VLSRs forward the path message since they are not the destination. When CSA1 receivs the RSVP_PATH message, it replies to it with RSVP_RESV message and sends it to VLSR1. VLSR1 forwards this message to VLSR2 because again it is not the destination of the message. Finally, VLSR2 forwards the RSVP_RESV message to CSA2. At this point, the LSP is active and can be used. The SNMP/TL1 Gateway translates the SNMP messages sent by the two VLSRs to TL1 commands in order to configure the two ROADMs.

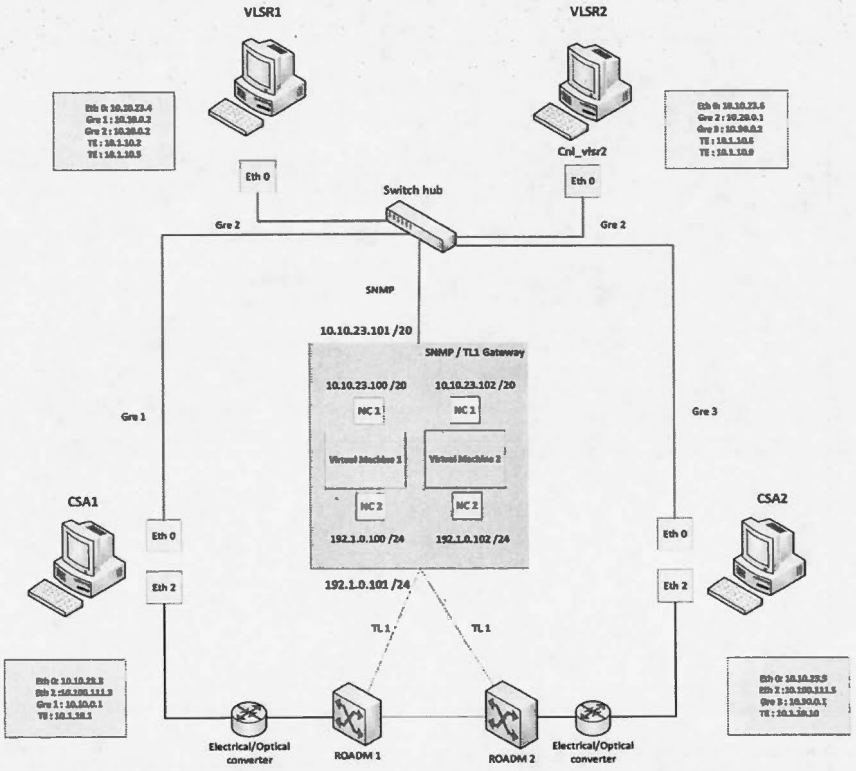


Fig. 8: DRAGON test with two ROADMs

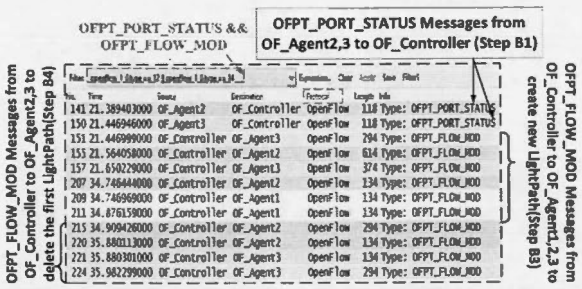


Fig. 7: OpenFlow Scenario2 : Wireshark screenshot

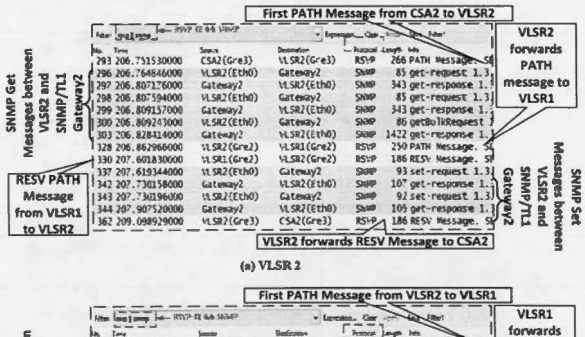


Fig. 9: GMPLS Scenario : Wireshark screenshot

C. Experimentation Results

Table I shows the time (in ms) consumed on each solution (***OpenFlow Messages Mapping*** and ***OpenFlow Extension***) and the GMPLS approach. In this table, Path1 and Path2 refer to the primary and the backup lightpaths. Path1 nodes are OF_Switch1 → ROADM2 → ROADM3 → OF_Switch2, while Path2 nodes are OF_Switch1 → ROADM2 → ROADM1 → ROADM3 → OF_Switch2. LSP on the table refers to Label Switch Path for GMPLS. LSP nodes are CSA1 → ROADM2 → ROADM3 → CSA2. The experiments results show that ***OpenFlow Extension*** solution (216 ms) outperforms ***OpenFlow Messages Mapping*** (227 ms) solution. This result is expected because ***OpenFlow Extension*** solution uses one message to encapsulate bidirectional lightpath information and ***OpenFlow Messages Mapping*** needs two messages. For the backup lightpath (Path2) which span on three nodes, ***OpenFlow Extension*** solution takes 239 ms to create the lightpath while ***OpenFlow Messages Mapping*** takes 269 ms.

	OpenFlow Messages Mapping Solution				
	Controller	Switch establishment			Total (ms)
		ROADM2	ROADM1	ROADM3	
Path1	16	121	-	90	227
Path2	18	110	30	111	269
	OpenFlow Extension Solution				
	Controller	Switch establishment			Total (ms)
		ROADM2	ROADM1	ROADM3	
Path1	16	100	-	100	216
Path2	18	90	30	101	239
	GMPLS Solution				
	RSVP-TE	Switch establishment			Total (ms)
		ROADM2	ROADM1	ROADM3	
LSP	130	110	-	100	340

TABLE I: The experiments timing

On the other hand, GMPLS takes more time (340 ms) to create lightpath than OpenFlow solutions. This is because the GMPLS-based control plane is complicated especially when it is deployed as a unified control plane (UCP) for IP/DWDM multi-layer networks. This is due to its distributed nature, the number of protocols, and the interactions among different layers. The flexibility and manageability of the GMPLS-based control plane is low, because, for example, if we want to create or update an end-to-end lightpath, the signalisation and reservation messages must be updated and exchanged between all the intermediate VLSRs. However, the OpenFlow-based UCP provides the maximum flexibility and manageability for carriers since all the functionalities are integrated into a single OpenFlow controller. More importantly, the OpenFlow-based control plane is a natural choice for a UCP in IP/DWDM multi-layer networks due to its inherent feature, as the procedure shown in Figure 5. Thus, the technical evolution from GMPLS to OpenFlow is a process that the control plane evolves from a fully distributed architecture to a fully centralized one.

V. Conclusion

In this paper, we experimentally present two solutions (***OpenFlow Messages Mapping*** and ***OpenFlow Extension***) for a dynamic wavelength path control in IP/DWDM multi-layer optical networks. The overall feasibility of these solutions is experimentally assessed, and their performance is quantitatively evaluated and compared with GMPLS approach, on an actual transparent network testbed. The comparison reveals that the OpenFlow-based control plane is simpler, more flexible and manageable than the GMPLS-based control plane, especially for an IP/DWDM multi-layer optical network. Finally, the experimental results show that the ***OpenFlow Extension*** solution outperforms the ***OpenFlow Messages Mapping*** and GMPLS solutions. It can significantly improve the performance of the control plane and reduce the lightpath setup time.

References

[1] "DRAGON: Dynamic Resource Allocation via GMPLS Optical Networks," http://dragon.east.isi.edu/twiki/bin/view/DRAGON/WebHome.

[2] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with openflow," in *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, March 2010, pp. 1–3.

[3] E. Mannie, "Generalized multi-protocol label switching (gmpls) architecture," *Interface*, vol. 501, p. 19, 2004.

[4] L. Liu, T. Tsuritani, and I. Morita, "Experimental demonstration of openflow/gmpls interworking control plane for ip/dwdm multi-layer optical networks," in *Transparent Optical Networks (ICTON), 2012 14th International Conference on*. IEEE, 2012, pp. 1–4.

[5] Y. Zhao, J. Zhang, H. Yang, and Y. Yu, "Which is more suitable for the control over large scale optical networks, gmpls or openflow?" in *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013*, 2013, pp. 1–3.

[6] T. Lehman, J. Sobieski, and B. Jabbari, "Dragon: a framework for service provisioning in heterogeneous grid networks," *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 84–90, March 2006.

[7] "SDN: Software Defined Networking," https://www.opennetworking.org/sdn-resources/sdn-definition.

[8] "OpenFlow," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[9] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "Openflow-based wavelength path control in transparent optical networks: a proof-of-concept demonstration," in *Optical Communication (ECOC), 2011 37th European Conference and Exhibition on*. IEEE, 2011, pp. 1–3.

[10] L. Liu, D. Zhang, T. Tsuritani, R. Vilalta, R. Casellas, L. Hong, I. Morita, H. Guo, J. Wu, R. Martínez *et al.*, "Field trial of an openflow-based unified control plane for multilayer multigranularity optical switching networks," *Journal of Lightwave Technology*, vol. 31, no. 4, pp. 506–514, 2013.

[11] L. Liu, D. Zhang, T. Tsuritani, R. Vilalta, R. Casellas, L. Hong, I. Morita, H. Guo, J. Wu, R. Martinez, and R. Munoz, "First field trial of an openflow-based unified control plane for multi-layer multi-granularity optical networks," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2012 and the National Fiber Optic Engineers Conference*, March 2012, pp. 1–3.

[12] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow and pce architectures in wavelength switched optical networks," in *Optical Network Design and Modeling (ONDM), 2012 16th International Conference on*. IEEE, 2012, pp. 1–6.

[13] http://www.openflow.org/documents/openflow-wp-latest.pdf.

[14] O. S. Consortium *et al.*, "Openflow switch specification version 1.0. 0," 2009.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[16] C. Headquarters, "Tl1 command reference for the cisco ons 15808 dwdm system," 2003.

[17] S. Das, "Extensions to the openflow protocol in support of circuit switching," *Addendum to OpenFlow protocol specification (v1. 0)Circuit Switch Addendum v0*, vol. 3, 2010.

[18] V. Tintor and J. Radunović, "Multihop routing and wavelength assignment algorithm for optical wdm networks," *International Journal of Networks and Communications*, vol. 2, no. 1, pp. 1–10, 2012.

[19] "Cisco ONS 15454 DWDM Reference Manual, Release 9.2," http://www.cisco.com/en/US/docs/optical/15000r9_2/dwdm/reference/guide/454d92_ref.html/, 2012.

[20] "RFC 3945 Generalized Multi-Protocol Label Switching (GMPLS) Architecture," http://www.ietf.org/rfc/rfc3945.txt.

[21] "GNU Zebra," http://www.gnu.org/software/zebra/.

[22] "KOMRSVP Engine," http://www.kom.tu-darmstadt.de/en/downloads/software/komrsvp- engine/ .

[23] "GNU General Public License," http://www.gnu.org/copyleft/gpl.html.

[24] "DRAGON Source Code," http://Dragon.maxgigapop.net/public/Dragon-swvlsr-daily.tar.gz.

[25] http://dragon.east.isi.edu/twiki/pub/DRAGON/VLSR/dragon-vlsr-implement-v2.1b.pdf.

[26] "SNMP4J API," http://www.snmp4j.org/.

[27] "iReasoning TL1 API," http://ireasoning.com/tl1api.shtml.

APPENDICE A : SCÉNARIO 1 ET 3

A.1 Fichiers de configuration au niveau du CSA1

****

Script Linux pour créer les GRE

****

#/sbin/modprobe ip_gre

/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 10.30.0.1/30 dev gre1

/sbin/ip route add 10.30.0.2 dev gre1

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cln_host1-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_host1-ospf

password karim

log file /usr/local/dragon/sbin/ospfd.log

interface gre1

description GRE tunnel between cnl_host1 and cnl_vlsr

ip ospf network point-to-point

router ospf

ospf router-id 10.10.23.3

network 10.30.0.0/30 area 0.0.0.0

ospf-te router-address 10.10.23.3

ospf-te interface gre1

level gmpls

data-interface ip 10.1.10.1

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

exit

line vty

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre1 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_host1-zebra

password karim

enable password karim

interface lo

interface gre1

line vty

log file /var/log/zebra.log

A.2 Fichiers de configuration au niveau du CSA2

****

Script Linux pour créer les GRE

****

```
#/sbin/modprobe ip_gre
/sbin/ip tunnel add gre2 mode gre remote 10.10.23.4 local 10.10.23.5 ttl 255
/sbin/ip link set gre2 up
/sbin/ip addr add 10.40.0.1/30 dev gre2
/sbin/ip route add 10.40.0.2 dev gre2
```

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

```
hostname cln_host2-dragon
password karim
```

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

```
hostname cnl_host2-ospf
password karim
interface gre2
description GRE tunnel between cnl_host2 and cnl_vlsr
ip ospf network point-to-point
router ospf
ospf router-id 10.10.23.5
network 10.40.0.0/30 area 0.0.0.0
ospf-te router-address 10.10.23.5
ospf-te interface gre2
level gmpls
```

data-interface ip 10.1.10.6

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

exit

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre2 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_host2-zebra

password karim

enable password karim

interface lo

interface gre2

line vty

log file /var/log/zebra.log

A.3 Fichiers de configuration au niveau du VLSR

****

Script Linux pour créer les GRE

****

#/sbin/modprobe ip_gre

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 10.30.0.2/30 dev gre1

sudo /sbin/ip route add 10.30.0.1 dev gre1

sudo /sbin/ip tunnel add gre2 mode gre remote 10.10.23.5 local 10.10.23.4 ttl 255

sudo /sbin/ip link set gre2 up

sudo /sbin/ip addr add 10.40.0.2/30 dev gre2

sudo /sbin/ip route add 10.40.0.1 dev gre2

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cnl_vlsr-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_vlsr-ospf

password karim

log file /usr/local/dragon/sbin/ospfd.log

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 10.10.23.4

network 10.30.0.0/30 area 0.0.0.0

network 10.40.0.0/30 area 0.0.0.0

ospf-te router-address 10.10.23.4

ospf-te interface gre1

level gmpls

data-interface ip 10.1.10.2 protocol snmp switch-ip

10.10.23.100 switch-port 3

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

metric 10

exit

ospf-te interface gre2

level gmpls

data-interface ip 10.1.10.5 protocol snmp switch-ip

10.10.23.100 switch-port 5

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

metric 10

exit

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre1 tc none mpls

interface gre2 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_vlsr-zebra

password karim

enable password karim

interface lo

interface gre1

interface gre2

line vty

log file /var/log/zebra.log

APPENDICE B : SCÉNARIO 2 ET 4

B.1 Fichiers de configuration au niveau du CSA1

****

Script Linux pour créer les GRE

****

#/sbin/modprobe ip_gre

/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

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cln_host1-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_host1-ospf

password karim

log file /usr/local/dragon/sbin/ospfd.log

interface gre1

description GRE tunnel between cnl_host1 and cnl_vlsr1

ip ospf network point-to-point

router ospf

ospf router-id 10.10.23.3

network 10.10.0.0/30 area 0.0.0.0

ospf-te router-address 10.10.23.3

ospf-te interface gre1

level gmpls

data-interface ip 10.1.10.1

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

exit

line vty

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre1 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_host1-zebra

password karim

enable password karim

line vty

log file /var/log/zebra.log

B.2 Fichiers de configuration au niveau du CSA2

****

Script Linux pour créer les GRE

****

#/sbin/modprobe ip_gre

/sbin/ip tunnel del gre3

/sbin/ip tunnel add gre3 mode gre remote 10.10.23.6 local 10.10.23.5 ttl 255

/sbin/ip link set gre3 up

/sbin/ip addr add 10.30.0.1/30 dev gre3

/sbin/ip route add 10.30.0.2 dev gre3

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cln_host2-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_host2-ospf

password karim

interface gre3

description GRE tunnel between cnl_host2 and cnl_vlsr2

ip ospf network point-to-point

router ospf

ospf router-id 10.10.23.5

network 10.30.0.0/30 area 0.0.0.0

ospf-te router-address 10.10.23.5

ospf-te interface gre3

level gmpls

data-interface ip 10.1.10.10

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

exit

line vty

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre3 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_host2-zebra

password karim

enable password karim

interface lo

interface gre2

line vty

log file /var/log/zebra.log

B.3 Fichiers de configuration au niveau du VLSR1

****

Script Linux pour créer les GRE

****

#/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

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cnl_vlsr1-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_vlsr1-ospf

password karim

interface gre1

description GRE tunnel between cnl_vlsr1 and cnl_host1

ip ospf network point-to-point

interface gre2

description GRE tunnel between cnl_vlsr1 and cnl_vlsr2

ip ospf network point-to-point

router ospf

ospf router-id 10.10.23.4

network 10.10.0.0/30 area 0.0.0.0

network 10.20.0.0/30 area 0.0.0.0

ospf-te router-address 10.10.23.4

ospf-te interface gre1

level gmpls

data-interface ip 10.1.10.2 protocol snmp switch-ip 10.10.23.100 switch-port 9

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

metric 10

exit

ospf-te interface gre2

level gmpls

data-interface ip 10.1.10.5 protocol snmp switch-ip 10.10.23.100 switch-port 2

swcap l2sc encoding ethernet

max-bw 125000000

max-rsv-bw 125000000

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

metric 10

exit

line vty

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre1 tc none mpls

interface gre2 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_vlsr1-zebra

password karim

enable password karim

interface lo

interface gre1

interface gre2

line vty

log file /var/log/zebra.log

B.4 Fichiers de configuration au niveau du VLSR2

****

Script Linux pour créer les GRE

****

#/sbin/modprobe ip_gre

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 10.20.0.1/30 dev gre2

sudo /sbin/ip route add 10.20.0.2 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

****

Configuration du démon DRAGON (/usr/local/dragon/etc/dragon.conf)

****

hostname cnl_vlsr2-dragon

password karim

****

Configuration du démon zebra ospfd (/usr/local/dragon/etc/ospfd.conf)

****

hostname cnl_vlsr2-ospf

password karim

interface gre2

description GRE tunnel between cnl_vlsr1 and cnl_vlsr2

```
ip ospf network point-to-point
interface gre3
description GRE tunnel between cnl_vlsr2 and cnl_host2
ip ospf network point-to-point
router ospf
ospf router-id 10.10.23.6
network 10.20.0.0/30 area 0.0.0.0
network 10.30.0.0/30 area 0.0.0.0
ospf-te router-address 10.10.23.6
ospf-te interface gre2
level gmpls
data-interface ip 10.1.10.6 protocol snmp switch-ip 10.10.23.101 switch-port 2
swcap l2sc encoding ethernet
max-bw 125000000
max-rsv-bw 125000000
max-lsp-bw 0 125000000
max-lsp-bw 1 125000000
max-lsp-bw 2 125000000
max-lsp-bw 3 125000000
max-lsp-bw 4 125000000
max-lsp-bw 5 125000000
max-lsp-bw 6 125000000
max-lsp-bw 7 125000000
metric 10
exit
ospf-te interface gre3
level gmpls
data-interface ip 10.1.10.9 protocol snmp switch-ip 10.10.23.101 switch-port 9
swcap l2sc encoding ethernet
max-bw 125000000
max-rsv-bw 125000000
```

max-lsp-bw 0 125000000

max-lsp-bw 1 125000000

max-lsp-bw 2 125000000

max-lsp-bw 3 125000000

max-lsp-bw 4 125000000

max-lsp-bw 5 125000000

max-lsp-bw 6 125000000

max-lsp-bw 7 125000000

metric 10

exit

line vty

****

Configuration du démon RSVP (/usr/local/dragon/etc/RSVPD.conf)

****

interface gre2 tc none mpls

interface gre3 tc none mpls

api 4000

****

Configuration du démon zebra (/usr/local/dragon/etc/zebra.conf)

****

hostname cnl_vlsr2-zebra

password karim

enable password karim

interface lo

interface gre2

interface gre3

line vty

log file /var/log/zebra.log

BIBLIOGRAPHIE

Azodolmolky, S., Wieder, P., and Yahyapour, R. (2013). Sdn-based cloud computing networking. In *Transparent Optical Networks (ICTON), 2013 15th International Conference on*, pages 1–4.

Berge, C. (1958). La theorie des graphes. *Paris, France.*

Channegowda, M., Nejabati, R., Fard, M. R., Peng, S., Amaya, N., Zervas, G., Simeonidou, D., Vilalta, R., Casellas, R., Martínez, R., noz, R. M., Liu, L., Tsuritani, T., Morita, I., Autenrieth, A., Elbers, J., Kostecki, P., and Kaczmarek, P. (2013a). Experimental demonstration of an openflow based software-defined optical network employing packet, fixed and flexible dwdm grid technologies on an international multi-domain testbed. *Opt. Express*, 21(5) :5487–5498.

Channegowda, M., Nejabati, R., and Simeonidou, D. (2013b). Software-defined optical networks technology and infrastructure : Enabling software-defined optical network operations [invited]. *Optical Communications and Networking, IEEE/OSA Journal of*, 5(10) :A274–A282.

CiscoONS (2014). http ://www.cisco.com/c/en/us/products/optical-networking/ons-15454-series-multiservice-provisioning-platforms/index.html.

Das, S., Parulkar, G., McKeown, N., Singh, P., Getachew, D., and Ong, L. (2010). Packet and circuit network convergence with openflow. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pages 1–3.

DRAGON (2006). https ://wiki.maxgigapop.net/twiki/bin/view/dragon.

Erickson, D. (2013). The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM.

Farrel, A. (2010). A unified control plane : dream or pipedream. In *6th International Conference on IP+ Optical Network (IPOP 2010), Tokyo, Japan*, pages 10–11.

Gringeri, S., Bitar, N., and Xia, T. (2013). Extending software defined network principles to include optical transport. *Communications Magazine, IEEE*, 51(3) :32–40.

Gudla, V., Das, S., Shastri, A., Parulkar, G., McKeown, N., Kazovsky, L., and Yamashita, S. (2010). Experimental demonstration of openflow control of packet and circuit switches. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pages 1–3.

Headquarters, C. (2003). Tl1 command reference for the cisco ons 15808 dwdm system.

Ji, P. (2012). Software defined optical network. In *Optical Communications and Networks (ICOCN), 2012 11th International Conference on*, pages 1–4.

Kumaki, K. (2008). Interworking requirements to support operation of mpls-te over gmpls networks.

Lehman, T., Sobieski, J., and Jabbari, B. (2006). Dragon : a framework for service provisioning in heterogeneous grid networks. *Communications Magazine, IEEE*, 44(3) :84–90.

Liu, L., Tsuritani, T., and Morita, I. (2012a). Experimental demonstration of openflow/gmpls interworking control plane for ip/dwdm multi-layer optical networks. In *Transparent Optical Networks (ICTON), 2012 14th International Conference on*, pages 1–4.

Liu, L., Tsuritani, T., and Morita, I. (2012b). From gmpls to pce/gmpls to openflow : How much benefit can we get from the technical evolution of control plane in optical networks ? In *Transparent Optical Networks (ICTON), 2012 14th International Conference on*, pages 1–4.

Liu, L., Zhang, D., Tsuritani, T., Vilalta, R., Casellas, R., Hong, L., Morita, I., Guo, H., Wu, J., Martínez, R., and noz, R. M. (2013). Field trial of an openflow-based unified control plane for multilayer multigranularity optical switching networks. *J. Lightwave Technol.*, 31(4) :506–514.

Mannie, E. (2004). Generalized multi-protocol label switching (gmpls) architecture. *Interface*, 501 :19.

O'Mahony, M. (1996). Results from the cost 239 project. ultra-high capacity optical transmission networks. In *Optical Communication, 1996. ECOC'96. 22nd European Conference on*, volume 2, pages 11–18. IEEE.

ONF (2014). https ://www.opennetworking.org/.

Papadimitriou, D., Berde, B., Martinez, R., Ordas, J., Theillaud, R., and Verbrugge, S. (2006). Generalized multi-protocol label switching (gmpls) unified control plane validation. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 6, pages 2717–2724.

Patel, A., Ji, P., and Wang, T. (2013). Qos-aware optical burst switching in openflow based software-defined optical networks. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pages 275–280.

Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., and Rao, N. (2013). Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7) :36–43.

Shiomoto, K. (2008). Framework for mpls-te to gmpls migration.

Shirazipour, M., John, W., Kempf, J., Green, H., and Tatipamula, M. (2012). Realizing packet-optical integration with sdn and openflow 1.1 extensions. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6633–6637.

Simeonidou, D., Nejabati, R., and Azodolmolky, S. (2011). Enabling the future optical internet with openflow : A paradigm shift in providing intelligent optical network services. In *Transparent Optical Networks (ICTON), 2011 13th International Conference on*, pages 1–4.

SNMMP4J (2003). http ://www.snmp4j.org/.

TL1API (2002). http ://www.ireasoning.com/tl1api.shtml.

Wang, J., Yan, Y., and Dittmann, L. (2013). On the design of energy efficient optical networks with software defined networking control across core and access networks. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pages 47–52.

Zang, H., Jue, J. P., Mukherjee, B., et al. (2000). A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical Networks Magazine*, 1(1) :47–60.