UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# OPENFLOW PROTOCOL EXTENSION FOR OPTICAL NETWORKS

THESIS

PRESENTED

AS A PARTIAL REQUIREMENT

FOR THE MASTER IN ELECTRICAL ENGINEERING

BY

MAHMOUD MOHAMED BAHNASY

NOVEMBER 2014

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# EXTENSION DU PROTOCOLE OPENFLOW POUR LES RÉSEAUX OPTIQUES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
MAHMOUD MOHAMED BAHNASY

NOVEMBER 2014

## ACKNOWLEDGMENTS

It is my pleasure to thank all those people who made this thesis possible.

At first, I would like to thank my advisor Professor Halima Elbiaze. With her encouragement, inspiration and her great efforts of explaining things clearly and simply; she made the research process as simple as possible throughout my Master period. She provided useful advice, good guiding, perfect company, and she always gave me great freedom to pursue independent work.

I would like to thank my colleagues for providing an activating and exciting environment in which I could learn and enhance my competence.

I wish to thank my entire extended family for all their love and encouragement specially my beloved mother. And most importantly, I wish to thank my lovely wife, Laila, as they always support me in this road and they were encouraging me along this way.

Finally, I would like to thank all the staff members of the Computer Science department at UQAM for their direct and indirect help during my studies at UQAM.

## RÉSUMÉ

Software Defined Networking (SDN) offre la possibilité de contrôler les réseaux en utilisant un logiciel fonctionnant sur un système d'exploitation dans un contrôleur externe, ce qui offre un maximum de flexibilité et de simplicité. OpenFlow ( OF ), une des implémentations SDN les plus utilisée, est présenté comme architecture et plan de contrôle unifié pour les réseaux de paquets et de circuits. Dans ce projet, nous proposons expérimentalement deux solutions basées sur OpenFlow pour contrôler à la fois les réseaux de paquets et les réseaux optiques : (1) OpenFlow Message-Mapping et (2) OpenFlow Extension. OpenFlow Message-Mapping est basée sur l'association de mesages OpenFlow à des actions appropriées. La deuxième solution que nous proposons, OpenFlow Extension, est basée sur l'extension du protocole OpenFlow standard en ajoutant de nouveaux messages pour suporter des informations d'interconnexion optique au lieu d'utiliser des messages OpenFlow standard. Nous avons implémeté ces deux solutions dans un banc d'essai et nous avons effectué deux expériences : (i) la création de canaux optiques de bout en bout. (ii) la restauration de chemin optique. Les mesures prises à partir de ces expériences sont utilisées pour l'implémntation d'un simulateur Java. Ce simulateur simule les performances de ces deux techniques sur deux topologies de réseaux optiques réels et les compare avec le protocole GMPLS standard. Le résultat est représenté sous format de graphiques comparatifs pour déterminer la technique qui a le meilleur temps d'établissement de liens, la plus petite charge de contrôle et le rapport de blocage le plus bas. La faisabilité de ces solutions a été vérifée dans notre banc d'essai et leurs performances sont quantitativement évaluées et comparées dans deux réseaux optiques réels.

## ABSTRACT

Software Defined Network (SDN) affords the possibility to control networks using software running on a network operating system in an external controller, which provides maximum flexibility, simplicity and manageability. OpenFlow (OF), one of the widely used SDN implementations, is presented as a unified control plane and architecture for packet and circuit switched networks. Based on this, in this thesis, we experimentally propose two solutions based on OpenFlow to control both packet and optical networks: (1) OpenFlow Message-Mapping and (2) OpenFlow extension. OpenFlow Message-Mapping is based on mapping the OpenFlow messages into appropriate cross-connect actions. The second solution we propose, OpenFlow Extension, is based on extending the standard OpenFlow protocol by adding new messages to carry the cross-connect information instead of using standard OpenFlow messages. We implemented these two solutions on a testbed and conduct two experiments: (i) End-to-End lightpath establishment. (ii) Backup lightpath restoration. The measurements taken from these experiments are used in writing a custom-built Java event-driven simulator. This simulator simulates the performance of these two techniques on two real optical network topologies and compare them with the standard GMPLS protocol. The result is depicted with comparative graphs to make it easy to determine which technique has the fastest establishment time, lowest control load and lowest blocking ratio. The overall feasibility of these solutions is assessed using our testbed and their performances are quantitatively evaluated and compared on a real optical network.

TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## ABREVIATIONS

| | |
|---|---|
| **AS** | Autonomous System |
| **CLI** | Command Line Interface |
| **Cost239** | European union Ultra-High Capacity Optical Transmission Network |
| **CSA** | Client System Agents |
| **CTC** | Cisco Transport Controller |
| **DRAGON** | Dynamic Resource Allocation via GMPLSnOptical Networks |
| **DWDM** | Dense Wavelength Division Multiplexing |
| **EMS/NMS** | Element and Network Management Systems |
| **FEDERICA** | Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures |
| **flow** | network traffic |
| **GENI** | The Global Environment for Network Innovations |
| **GMPLS** | Generalized Multi-Protocol Label Switching |
| **ITU** | International Telecommunication Union |
| **LSA** | Link State Advertisement |
| **LSP** | Label Switched Path |
| **MPLS** | Multi-Protocol Label Switching |
| **Node** | network node (Switch, router ...) |
| **OADM** | Optical Add-Drop Multiplexer |
| **NSF** | National Science Foundation |
| **OF** | OpenFlow |

| | |
|---|---|
| **OFP** | OpenFlow Protocol |
| **ONF** | Open Networking Foundation |
| **ONS** | Optical Network Switches |
| **OSPF** | Open Shortest Path First |
| **PCE** | Path Computation Element |
| **PSTN** | Public Switched Telephone Network |
| **QoS** | Quality-of-Service |
| **ROADM** | Reconfigurable Optical Add-Drop Multiplexer |
| **RSVP-TE** | Resource Reservation Protocol-Traffic Engineering |
| **SDN** | Software Defined Network |
| **SLA** | Service Level Agreement |
| **Switch** | network node (Switch, router, etc..) |
| **TCO** | Total Cost of Ownership |
| **TDM** | Time Devision Multiplexing |
| **TED** | Traffic Engineering Database |
| **UCP** | Unified Control Plane |
| **VLSR** | Virtual Label Switch Routers |
| **VPN** | Virtual Private Network |
| **WDM** | Wavelength-division Multiplexing |
| **WSON** | Wavelength Switched Optical Network |

# INTRODUCTION

## Overview

The exponential growth of Internet traffic requires network providers to construct efficient networking systems. These large networks need a complex and sophisticated control system especially when it includes two different infrastructures. One solution to manage this problem is to reduce the differences in network structure, for example, most network providers have removed telephony core switches and replaced them by using voice over IP services.

Today's networks are composed of an optical domain (circuit-switched networks) and an electrical domain (packet-switched networks). These two network structures operate on different network layers: circuit-switched networks operates on layer one and two, while packet-switched network operates on layer three and four (Figure 1). However, electrical

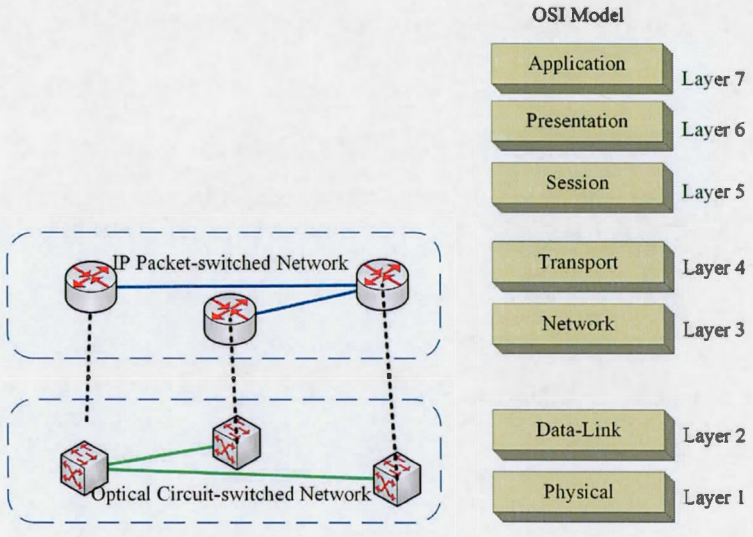



Figure 1: IP and Transport Networks Operating Layers

domain is more flexible and easier to manage, service providers can not replace optical devices with electrical devices because optical network has many benefits over electrical network. Some of these advantages are:

- Optical network equipment support up to 10 times the capacity as electrical equipment support.
- Optical transmission supports very long distances without signal attenuation.
- Optical transmission is interference free which makes it more robust.

Most of infrastructure providers use two different control plane, one for electrical domain and another for optical domain.

The separation between these two networks is because of the different management methodology of establishing a data path. Optical domain which is circuit-switched network operates on layer 1 and 2 of OSI model, while electrical domain is packet-switched network and it operates on layer 3 of OSI model. Another difference between the two domains is packet-switched network meant to be distributed control, each router has its own locally created control strategy, while on the other hand circuit-switched network is mostly centralized control. This separation presents a lack of common control mechanism which supports both network domains.

Most Infrastructure providers use two control mechanisms to operate both networks which is more expensive and inefficient than operating one converged network with a unified control mechanism. Some efforts have been done to unify the control and management of heterogeneous networks. The most mature and widely common example of these efforts is Generalized Multi-Protocol Label Switching (GMPLS)(Mannie, 2004) protocol which is very complicated and not even commercially adapted. Even though it was used, GMPLS did not completely unify the control mechanism. Indeed, it preserves the separation between the two networks.

**Motivation**

Service providers are obliged to own and operate two distinct wide-area networks (packet-switched and circuit-switched networks). For example, traditional service providers like AT&T, Verizon, British Telecom, Deutsche Telekom, NTT and others are all tier 1 and tier 2 ISPs (wikipedia, 2014). These heterogeneous networks require two different design and management teams even within the same organization. For sure, owning and operating two separate networks is inefficient and it causes great management overhead. The coordination

between these two teams is also another challenge to defeat. It also increases the cost of networks management, operating, designing, planning, and maintenance which effect directly the Total Cost of Ownership (TCO).

Networks are built based on closed-systems. Routers and switches from the same vendor have the same private features and services. These features are closed and kept secret inside each vendor's product. This secrecy and closed-box characteristic of network nodes features and services slow down the networks innovation and improvements. Using proprietaries management systems by each vendor creates barriers on face of network development in both IP and transport networks. Thus, it is clear that managing two separate networks operating differently is inefficient.

Software defined networking (SDN) proposes a new architecture capable of managing different networks with different infrastructures even though with different operational layer. This emerging concept, SDN, encourages us to present a common abstract that fits with both types of network and provides a common architecture for controlling both networks.

Some efforts have been done to present SDN-based UCP to control packet and circuit switches using the most commonly known protocol (OpenFlow). Most notably, PAC.C Das et al. (2010) has experimented with alternative approaches. Other papers Liu et al. (2011, 2013, 2012) have presented similar work as PAC.C by providing an experimental study or a Proof-of-Concept to support the use of OpenFlow as a unified control plane. However, Giorgetti et al. (2012) presents a comparison study between OpenFlow and GMPLS solutions based on a simulation. In this work, 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. Finally, we conduct the comparison between the OpenFlow solutions and the GMPLS approach by simulation on two real network topologies.

### Goals

As we discussed before, managing two separate networks operating differently are inefficient. Thus, in this thesis, our main goal is to find a way to manage these heterogeneous networks.

In this thesis, we are working on finding a common control mechanism to manage both networks. These approaches are based on SDN technology which provides a common abstract to fit both networks. This control mechanism should be able to manage both packet-switched network and circuit-switched network. This approach has to provide a simple and efficient method to manage both networks.

### Contribution

In this thesis, two solutions to converge both types of networks is proposed. This proposal is based on the concept of Software Defined Networking (SDN)(open networking foundation, 2013). One of the widely used SDN protocols is OpenFlow protocol. We conducted an experiment of implementing OpenFlow protocol to control both circuit-switched and packet-switched networks. Two techniques of using OpenFlow have been implemented: (1) OpenFlow message-mapping. In this technique, we map the OpenFlow messages into a suitable lightpath setup command (using TL1 command(CISCO, 2012b)). (2) OpenFlow extension. In this technique, we extend the OpenFlow protocol by adding new messages to support the lightpath specification. We used these new messages to carry the requested lightpath information. In both techniques we implemented an OpenFlow agent to translate between OpenFlow messages and the TL1 command and execute it on the hardware switches (Figure 2). In the laboratory, we conducted a simple network which consists of 2 Cisco ONS 15454 DWDM Reconfigurable Optical Add-Drop Multiplexer (ROADM) switches and one ONS 15454 DWDM Optical Add-Drop Multiplexer (OADM) switch connected as in figure 3. Two electrical-optical-converters are connected to each side of the Optical network (to ONS2 and ONS3). Each electrical-optical-converter is connected to an OpenFlow switch. Each OpenFlow switch is connected to a client (Figure 3).

Two experiments have been conducted for each technique: (i) End-to-End lightpath establishment. (ii) Backup lightpath restoration. The measurements taken from these experiments are used in writing a custom-built Java event-driven simulator. The objective of this simulator is to simulate the performance of these two techniques on two real optical network

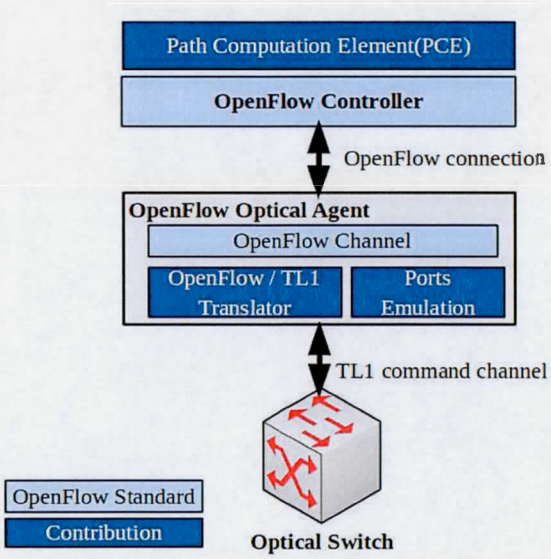



Figure 2: OpenFlow Messages/TL1 Commands Translator Agent

topologies and compare the result with those obtained by simulating the GMPLS protocol. The result is depicted on comparative graphs to make it easy to determine which technique has the fastest establishment time, lowest control load and lowest blocking ratio.

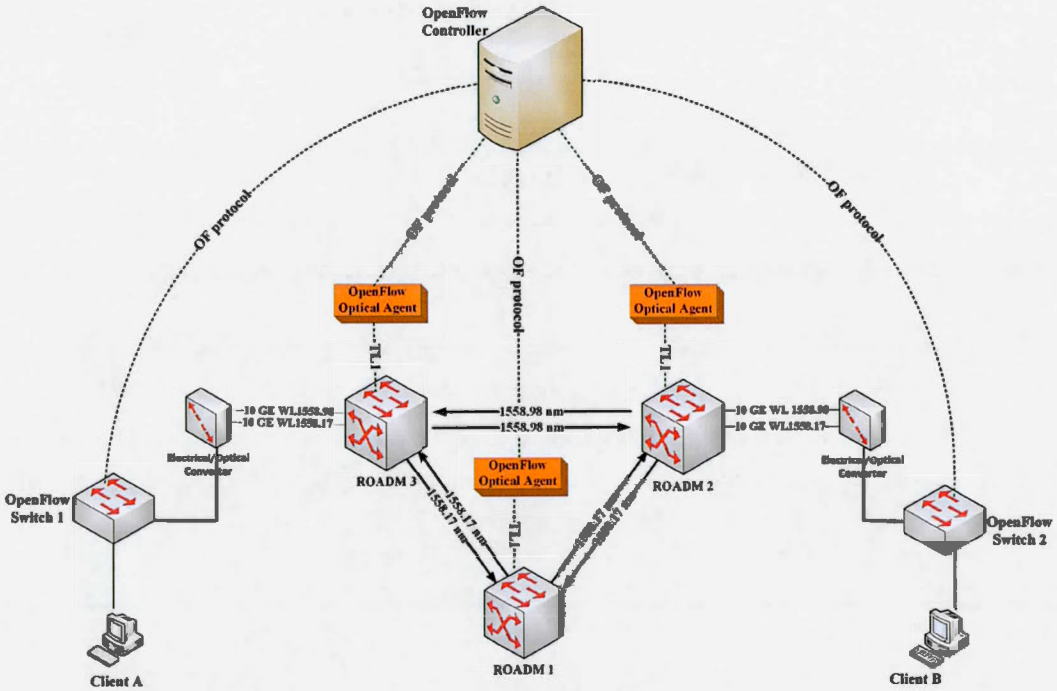



Figure 3: Testbed Architecture

## STRUCTURE OF THIS DISSERTATION

This thesis is laid out as follows.

Chapter 1 introduces the infrastructures of the two networks. Then, it states the problem of managing this heterogeneous network, after that it introduces a general information about the protocol used to manage this problem, and introduces the existing approaches address this problem.

Chapter 2 explains the proposed solution to address this problem explaining each component we created in this research project. Then, it explains how each technique uses the components we created. This chapter also presents a brief explanation of how GMPLS operates, and the messaging types used in this solution for the purpose of comparing it with our proposed solutions.

Chapter 3 discusses the two experiments conducted using our proposed solutions.

Chapter 4 presents the custom-built Java event-driven simulator algorithms for each solution. It presents also the different topologies we used to run the simulator on. Then it discusses the results got from this simulator.

Finally, we present the conclusions of our work and suggest directions for future research in Chapter 5.

# CHAPTER I

# PRINCIPAL CONCEPTS

This chapter presents the principal concepts of the IP network and the transport network. Then it presents the two old approaches to address the controlling of heterogeneous network. Finally, it presents the software-defined network concept which we build our solutions on.

## 1.1 Circuit-switched and Packet-Switched Network

Wide area network is the backbone of the Internet which is IP packet-switched network. Packets are switched hop-by-hop from source to destination through IP nodes. However, the packets may be transported physically on optical circuit switches and fibers (Figure 1.1). In some articles this underlying circuit-switched network is named as the Transport Network. We will take a closer look at the two networks in the following sections.

### 1.1.1 Internet Architecture

Internet components (layers, naming, addressing, protocols etc.) have been widely covered in several books and this thesis is not about the Internet architecture, but we will give a brief introduction about it. Internet is a collection of interconnected IP networks. The networks that compose Internet have independent ownership, administration and management. These networks use special kind of routing protocol capable of advertise IP addresses information between these domains, known as Autonomous Systems (AS)(Wikimedia Foundation, 2003), and capable of choosing routes across routing domains.

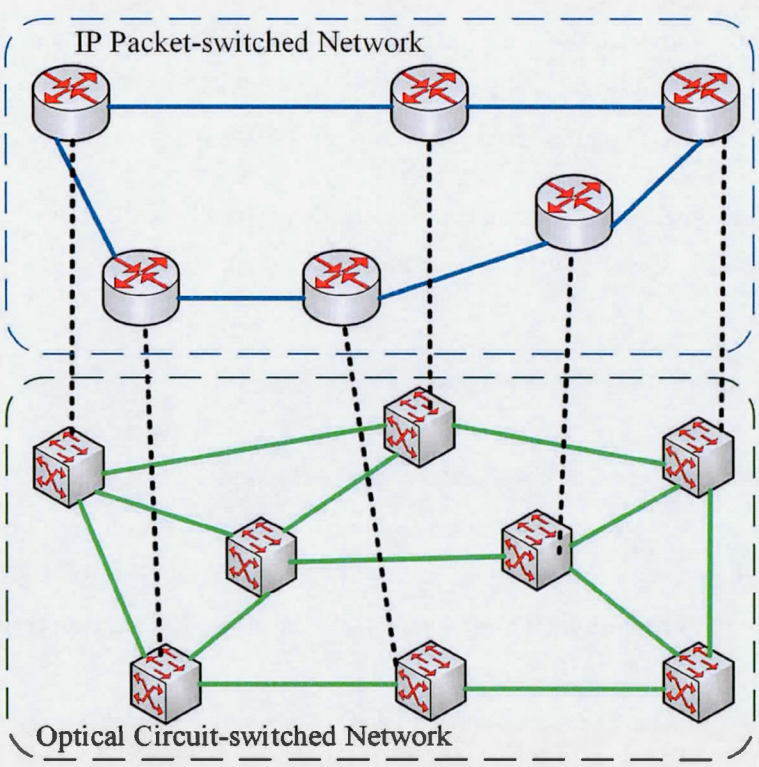



Figure 1.1: IP and Transport Networks

IP networks are based on distributed control mechanisms. These control mechanisms require routing protocols (I-BGP, OSPF etc) and sometimes signaling protocols (LDP, RSVP etc) implemented in each router. Control mechanisms are automated after initial configuration for each node (either manually or using scripts). These automated mechanisms allow network nodes to automatically discover their neighbors, the network topology, exchange routing information, forward packets, learn about failures and re-route packets to avoid this failure and try to guarantee the flow continuity.

Network services or functions in IP networks have a distributed implementation too. Each network-node-vendor implements its features exclusively and nonstandard even though they are using standard control mechanisms.

In case of congestion, IP network performs badly. Even though the public IP networks grant best-effort Quality-of-Service (QoS), some Service Level Agreements (SLAs) and Quality-of-Service (QoS) guarantees are not applicable easily.

IP networks management requires configuration, typically via Command Line Interface (CLI), monitoring, auditing, and maintenance. In general, IP networks are hard to manage.

### 1.1.2 Transport Network Architecture

The main function of a transport network is to provide communication between two geographic locations presented by network nodes. This connection may be established by a time-slotted circuit like Time Division Multiplexing (TDM) or wavelength-circuit Wavelength Division Multiplexing (WDM) figure 1.2. The IP network is an overlay layer on the transport layer.

Transport networks also support several overlay networks or client networks, e.g.: IP net-

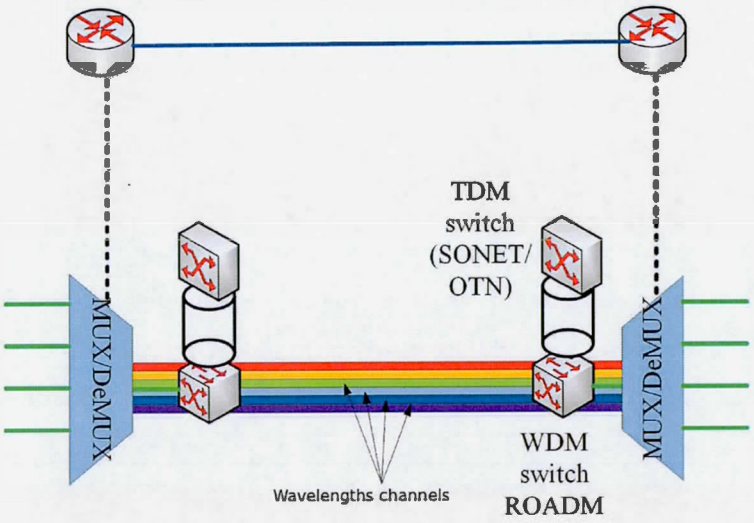



Figure 1.2: IP Network Overlay Transport Networks

works, Public Switched Telephone Network (PSTN), private-networks, etc. (Figure 1.3). More information about transport network architecture is described by the ITU in (ITU,

2000).

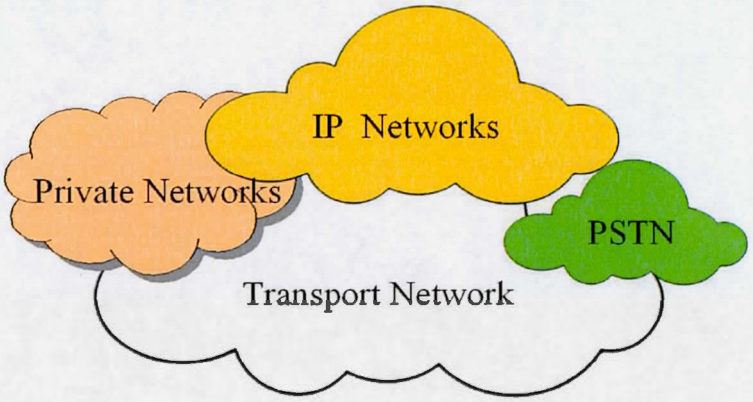



Figure 1.3: The Overlay Networks supported by Transport Network

Transport networks are not like IP networks, they are always intra-domain controlled and not automated-control. The transport networks are divided into partitions called "domains". Each domain is controlled separately and manually. They have Element and Network Management Systems (EMS/NMS) and Operations Support Systems (OSS) which perform all control and management. These systems are not programmatic, vendor proprietary systems, and manually configured (Figure 1.4). Providing services in a transport network is very complicated and long manual procedures. For example, providing a data-path between two end-points requires several steps: First, providing the source and the destination, planning the path from the source to the destination. Then, each provider executes the plan by manually configuring their equipment using their corresponding management systems along the path. Finally the test teams verify the path. Normally, this process takes days or maybe weeks, and the path created is static and stays in place for months or years.

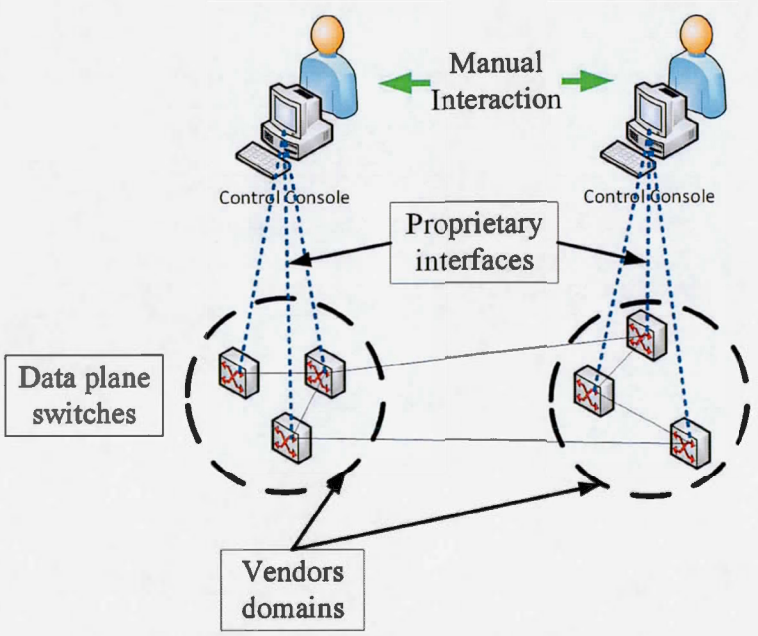



Figure 1.4: Transport Network Control & Management

## 1.2 Existing Approaches

In this section, we discuss two points of view of addressing the IP and transport networks. The first point of view, as we discussed before, is to eliminate circuit switching between network nodes. The second attempts to unify the control and the management for both IP and transport networks.

### 1.2.1 IP over WDM

In this point of view, we can achieve our goal, managing one homogeneous network, by eliminating the circuit switching components in network. As stated before in section 1.1.2, transport network supports many networks as overlay services (Figure 1.3). One example which has almost been eliminated is PSTN by moving traditional voice services to IP network instead of circuit-switched network on both end-user and service provider's core side. Meanwhile, private networks are moving to packet-switched networks based solutions by em-

bracing the Virtual Private Networks (VPNs). It is clear that customers and service providers are moving to eliminate circuit switching components or nodes and trying to find a packet switching substitute for their requirements Computerworld (2000); Das et al. (2010). In this case, in the future the Internet will be the only client uses the transport network Figure (1.5).

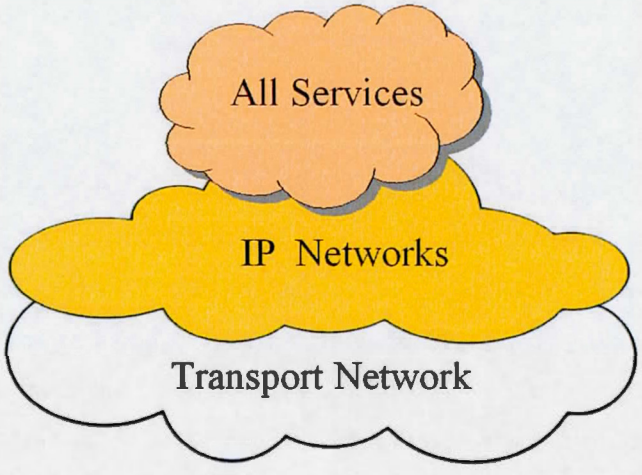



Figure 1.5: IP over WDM scenario for the Future of the Networks

In this scenario, it is logical to ask if circuit-switched underlying transport networks are required or they can be substituted. As we mentioned before in the overview, circuit-switched networks are very useful and have many benefits which make them indispensable. Packet-switched networks are always more expensive than circuit-switched networks because of their complicity of management and the huge capability of optical switches and optical fibers. Circuit switching switches are much more scalable; a circuit switch can switch much higher data rates (about 10 times more than packet switching switch), and consume much less power than an electronic packet switch (about 1/10 times less than packet switching switches). In general, optical circuit-switched networks are faster, simpler and more space efficient. They also have higher capacity, lower cost and lower power consumption than electronic packet-switched network. Therefore, the two networks must work together on a suitable and efficient control system.

### 1.2.2 GMPLS as a Unified Control Mechanism

Using SDN to create a unified method to control packet and circuit switches is not the first approach to control both networks. GMPLS (Mannie, 2004) is the most commonly known as a unified control mechanism. GMPLS has standardized within the IETF (since 2000).

Generalized Multi-Protocol Label Switching (GMPLS) was designed as an extension of MPLS to offer a unified control plane (UCP) for different networks, packet and circuit switching networks. This approach aims to use the capabilities of MPLS as a labeling protocol and extend this feature to work on circuit-switched networks. MPLS had a well-developed control plane based on the IP network. Thus based on this distributed control plane, GMPLS was built as a unified control plane. GMPLS extends distributed methodology, Routing protocol (OSPF-TE) and signaling protocols (RSVP-TE) to control circuit switches (Banerjee et al., 2001; Farrel and Bryskin, 2005; Mannie, 2004).

GMPLS has extended MPLS to include Time-Division Multiplex capabilities, Lambda Switch capabilities or Wavelength-Division Multiplex Switching capabilities as well as the Packet switching capabilities and Layer-2 Switching capabilities inherited from MPLS. Furthermore, GMPLS eliminates the need of an operator, the entire network can be automated.

GMPLS is a very mature protocol and it was standardized more than a decade ago. However, it still was not industrially implemented by equipment vendors because of its complexity. GMPLS has not been seen yet commercially deployed as a unified control plan. In fact, it is not even deployed as a control plane for transport network according to these articles(Das et al., 2012; lightreading.com, 2011). GMPLS is a distributed protocol. This feature reveals many problems with network stability and the control simplicity while most network equipment vendors prefer centralized control solution.

## 1.3 SOFTWARE DEFINED NETWORKING (SDN)

The traditional network node consists of built-in services and protocols. This hierarchy combines the control plane and the datapath in one box (Figure 1.6).

This hierarchy causes a huge limitation of innovation in real-world networks because the enormous installed base equipment and protocols. The unwillingness to experiment with production traffic is also an obstacle for the researchers. This limitation has created a high barrier for new ideas. For example, it is almost impossible to practically experience new routing protocol or alternative to IP protocol. Clearly the result is newest ideas from the network researchers which do not have chance to be tried or tested.

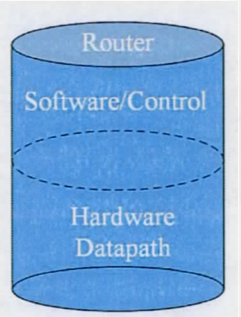



Figure 1.6: The Traditional Network Node hierarchy

Many networking efforts are done on the field of developing programmable networks. This work is based on the isolation of the datapath (data plane) and the applications responsible for controlling this datapath (Figure 1.7).

The slicing in the virtualized programmable networks allows the researchers to try new ideas which increase the rate of innovation McKeown et al. (2008a). Figure 1.8 shows how the production network could be used to carry the experimental flows without interference.

Programming network nodes provide the capabilities of network slicing, virtualization,

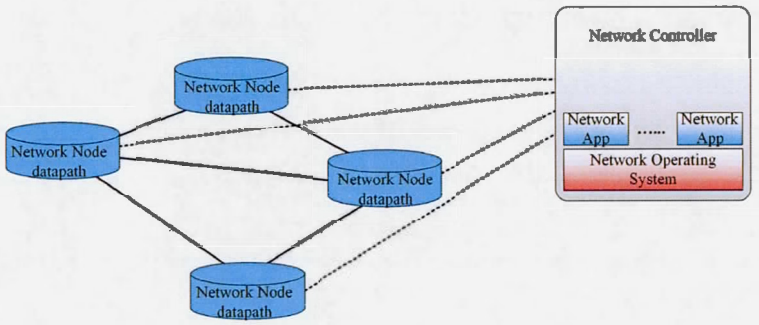



Figure 1.7: The Software-Defined Networking Network Node hierarchy

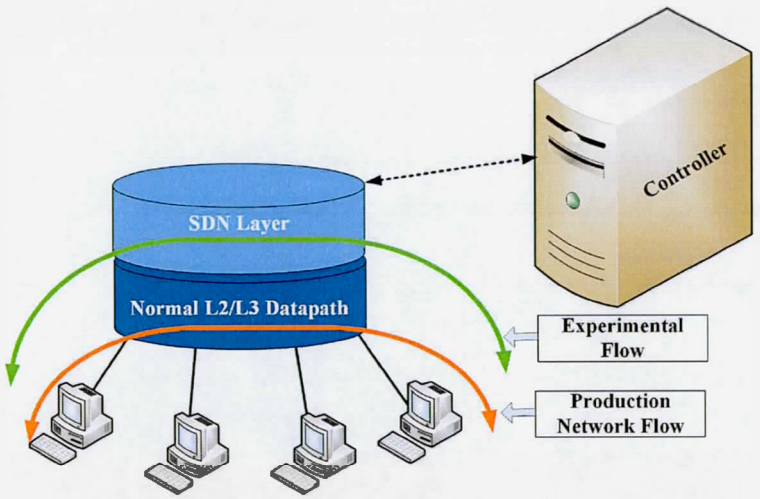



Figure 1.8: Software-Defined Networking Network Node hierarchy

and separation, which accelerate network innovation. Even though, SDN has some obstacles to defeat. Commercial switch and router vendors do not usually provide an open software platform. The network equipment vendors do not accept to open up their boxes, as they have spent many years developing their products and enhancing their products performance. In addition, open systems will lower the barrier for new competitors. A few open software platforms are already existing, for example a PC with several network interfaces and an operating system support packets routing between interfaces which most operating systems do Naous et al. (2008). This model is effective, but the problem is the performance. A PC has limited number of ports to install network interfaces on it, and the packet-processing speed is very limited (PC typically support maximum of 1Gbit/s while closet switches process over

100Gbits/s of data and increasing). Some network equipment providers started to provide equipment with SDN support, for example CISCO, HP, Juniper and NEC.

In brief, in SDN the configurations of network nodes, switches and routers, are done by software (controller) instead of manual involvement of the network administrator. Hence, SDN offers error-free network reconfiguration method, as well as high availability. Indeed, if a problem occurs in the network the automated recovery mechanism is triggered by the software allowing faster convergence compared to the manual approaches. SDN has a centralized knowledge about the network McKeown et al. (2008a), so the convergence process is faster and more accurate than distributed method.

### 1.3.1 OPENFLOW

We briefly outline the main characteristics of OpenFlow. More details and exhaustive documentation are available in the OpenFlow white paper (McKeown et al., 2008a) and in the OpenFlow specification (Consortium et al., 2009).
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 interrupting the existing traffic or network availability (McKeown et al., 2008b). In a traditional network, the data path and the control path occur on the same device (switch, router). Open Flow separates these two functions; OpenFlow switches perform the data plane function and OpenFlow controller implements the control plane intelligence and communicates with the OpenFlow switch via a secure OpenFlow protocol channel (Figure 1.9).

The main goal of SDN is the separation between the control plane and the data plane which OpenFlow algorithm implemented as in figure 1.10 (Consortium et al., 2009; OpenFlow, 2011). Based on this goal the controller and the switch have separated tasks to do. An OpenFlow Switch consists of one or more flow tables and group tables, which perform packet lookups and forwarding, and an OpenFlow Channel that is connected to an external controller. Each Flow table in the Switch contains a set of flow entries; each flow entry con-

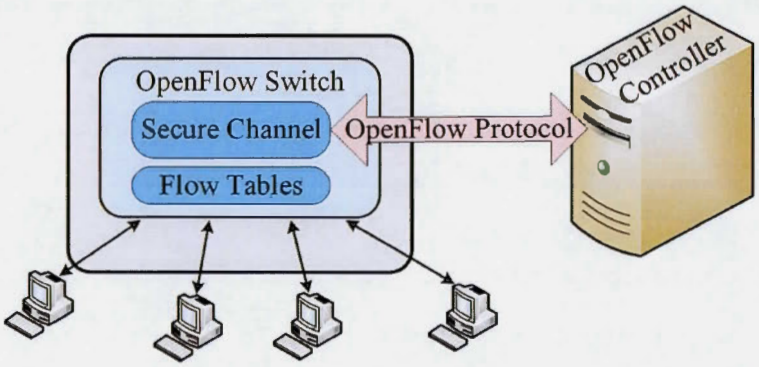



Figure 1.9: OpenFlow network

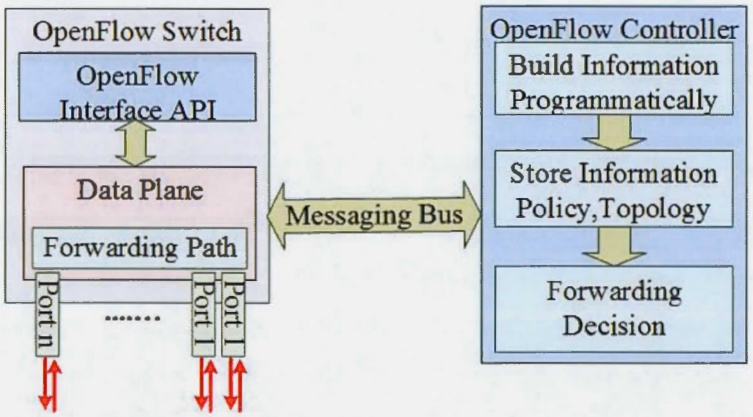



Figure 1.10: The separation between data plane and control plane using OpenFlow

sists of match fields, counters and a set of actions. These actions associated with each flow entry tell the switch what to do with the packets match this flow entry. The most OpenFlow actions basic types are:

- Forward the flow packet to a given port (or ports in case of multicast).
- Encapsulate the packet and forward it to the controller. This happen mostly with the first packet of a new flow, so the controller could decide if the flow should be added to the flow table, or to audit specific flow.
- Drop flow packets. This could be used to limit denial of service attacks.
- Forward the flow packets through the normal processing procedures. This action is

useful for separating the flows which do not belong to OpenFlow traffics.

The basic idea is to use the flow tables that most switches and routers contain. OpenFlow uses this common function (each switch has a flow table) and provides an open protocol to program the flow table by sending flow entries with associated actions to the switch and reading statistics about these flow entries (Figure 1.11).

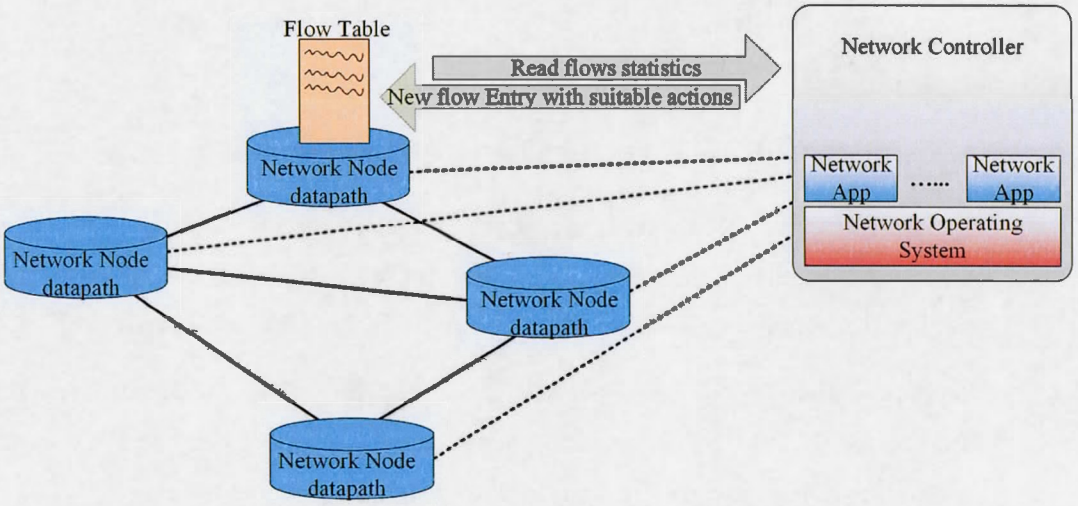



Figure 1.11: Packet flow through an Open Flow switch

When the OpenFlow switch receives a packet, it searches for a match field in its flow table. If it finds a match, first it updates the counters. Then, it fetches the actions associated with this flow entry and executes these actions on this packet. If it did not find a match, it continues with all flow tables. Finally if no match exists in all flow tables, it either drops the packet or sends it to the controller based on the table configurations. This algorithm is depicted in the flow chart in figure 1.12.

OpenFlow protocol messages are restricted in three categories (Consortium et al., 2009); *controller-to-switch*, *asynchronous*, and *symmetric*, each with multiple sub-categories:

- **Controller-to-switch :** These messages are initiated by the controller and may or

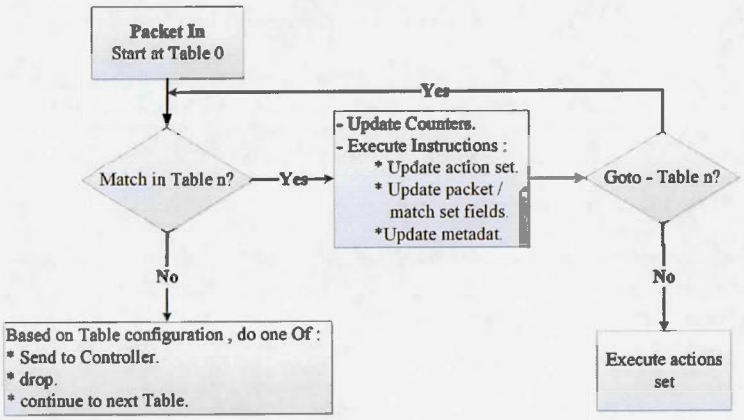



Figure 1.12: Packet flow through an Open Flow switch

may not require a response from the switch. They are used for directly managing or inspecting the state of the switch. The main controller-to-switch message types are: *Features*, *Configuration*, *Modify-State*, *Read-State*, *Packet-Out*, and *Barrier*.

- **Asynchronous :** These messages are sent by the switch without the controller solicitation. Switches send asynchronous messages to the controller to denote packet arrival, switch state change or error. The four main asynchronous message types are: *Packet-in*, *Flow Removed Message*, *Port Status Message*, and *Error Message*.

- **Symmetric :** These messages are initiated by either the switch or the controller and sent without solicitation. The main symmetric messages types are: *Hello*, *Echo Request* and *Echo Reply*.

OpenFlow is an independent protocol and available on currently running networks. These advantages put it at the head list of network virtualization techniques which includes several ambitious work like The Global Environment for Network Innovations (GENI) (geni.net, 2014) and Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures (FEDERICA) (fp7 federica.eu, 2014). The main characteristics of the OpenFlow that make it the best choice are:

- *Separation between control plane and data plane :* The key advantage of the OpenFlow protocol is the separation between data flow and control flow (OpenFlow, 2011),

(Consortium et al., 2009).

- *Centralized :* Most of infrastructure providers prefer centralized solution, which offers them simpler management and easier administration than distributed solution.
- *Simple and Flexible :* Because of the centralized nature of OpenFlow, this protocol is easy to manage and more flexible.

An example of the simplicity and separation attained by the OpenFlow is: if a researcher invents a new routing protocol X-OSPF, for example, and he wants to test it, he can implement his routing protocol on the controller reading the centralized information available at the controller instead of implementing it on each network node, and he only needs to send the flow entries to the network nodes (Routers and Switches).

# CHAPTER II

# PROPOSED SOLUTIONS

In this chapter we present our solutions based on OpenFlow protocol as a unified control plane for both optical and electrical networks. OpenFlow supports the separation of data and control planes for circuit and packet networks. The treatment of L4-L2 flows provide a simple flow abstraction that fits well with both types of networks. Hence, OpenFlow presents a common platform for controlling the underlying switching hardware, these flows of different granularity, while allowing all of the routing, control and management to be defined outside the datapath, in the OpenFlow controller as extended network applications (Figure 2.1).

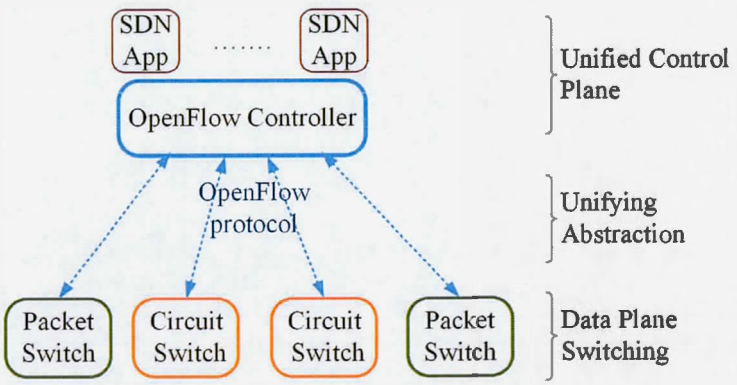



Figure 2.1: Unified architecture of a converged Packet-Circuit network

Two solutions for using OpenFlow protocol as a unified control plane on both optical and electrical domains are presented in this thesis. These techniques is compared with the standard GMPLS technique and presented in this research. The first solution is *OpenFlow*

*message-mapping*. In this solution we map the OpenFlow standard messages (like FLOW-MOD message) into optical domain commands to create or delete the lightpath, and translate the optical switch ports state into OpenFlow FeatureReply message. Otherwise, for the second solution *OpenFlow extension*, new messages is added to the OpenFlow protocol. These messages have the capabilities to carry the L1/L2 switching information explicitly. The added messages to OpenFlow Protocol is explained in details in the OpenFlow Circuit Switch Specification Das (2010)

In both solutions we implemented an OpenFlow agent to translate the OpenFlow messages to its proper TL1 commands (Headquarters, 2003) to be executed on the optical switch using telnet channel. The OpenFlow Controller has been extended by adding a new application we call it path computation element module (PCE). This addition allows the controller to calculate the lightpath for the requests. Then, it sends the appropriate messages to the proper optical switches (Figure 2.2).

In this section, we first explain the OpenFlow channel, the OpenFlow optical agent and

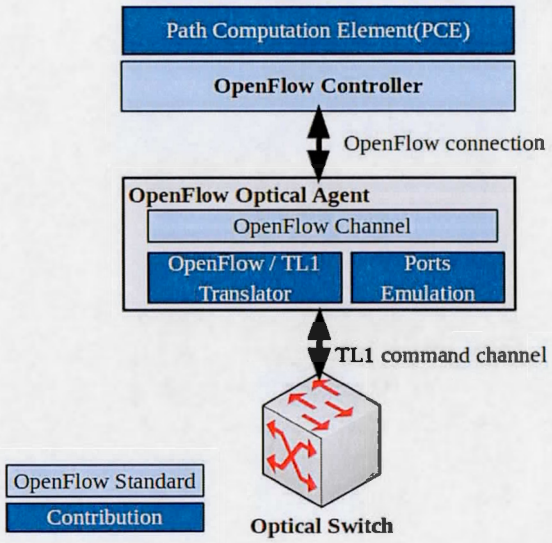



Figure 2.2: OpenFlow Agent

the PCE algorithm. Following that with detailed presentations of our solutions. Finally we

present the GMPLS with PCE lightpath mechanism for the purpose of comparing it with our solutions.

## 2.1 OpenFlow channel

OpenFlow channel is a key part of either the controller or the switches. In our code we used the *(openflow)* message library which is used in Beacon Java-based OpenFlow controller. This Message Library is a Java implementation of the OpenFlow specification (Consortium et al., 2009). This Message Library encodes and decodes OpenFlow messages from Java rich data types into the bytes stream and vice versa (Figure 2.3).

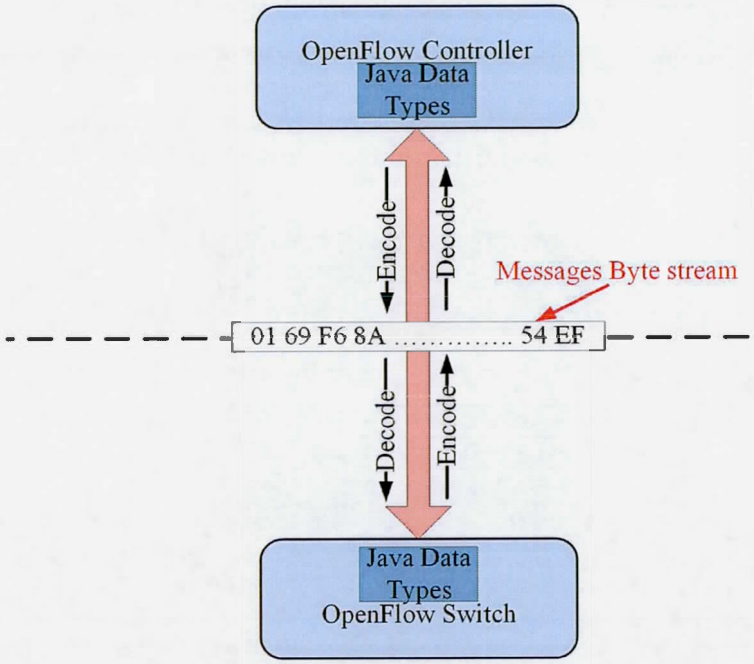



Figure 2.3: OpenFlow Channel

In order to create and encode an OpenFlow message, the application Uses the Message Factory class to create a message of the required type. Then, it encodes this message into a byte-stream using Message Factory class, to be transmitted over the media. The other side

(controller or switch) will receive the byte-stream. Then, it uses the Message Factory class to decode incoming byte-stream into an OpenFlow messages of their Java rich data type form (Figure 2.4).

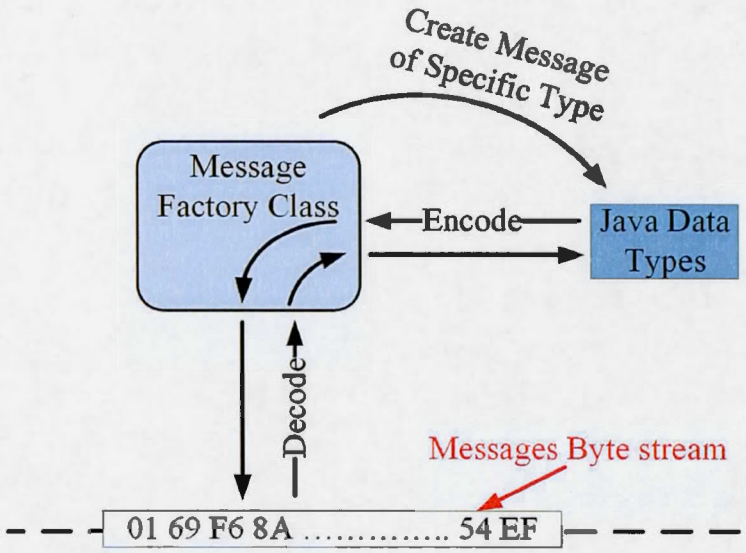



Figure 2.4: OpenFlow Message Factory

Table 2.1 shows a sample code to create a message using the Message Factory class. First, in line 1 we create a message factory instance *(BasicFactory)*. Then, we have this factory create a message of type *OFType.FLOW_MOD*. After that, we set the message properties (line 03 - 20) including the addition of the relevant actions list. Finally, we convert this message into byte-stream by calling the *stream.write()* function at line 22.

```
01    BasicFactory factory = new BasicFactory();
02    OFFlowMod fm = (OFFlowMod) factory.getMessage(OFType.FLOW_MOD);
03    fm.setBufferId(bufferId);
04    fm.setCommand((short) 0);
05    fm.setCookie(0);
06    fm.setFlags((short) 0);
07    fm.setHardTimeout((short) 0);
08    fm.setIdleTimeout((short) 5);
09    match.setInputPort(pi.getInPort());
10    match.setWildcards(0);
11    fm.setMatch(match);
12    fm.setOutPort((short) OFPort.OFPP_NONE.getValue());
13    fm.setPriority((short) 0);
14    OFActionOutput action = new OFActionOutput();
15    action.setMaxLength((short) 0);
16    action.setPort(outPort);
17    List<OFAction> actions = new ArrayList<OFAction>();
18    actions.add(action);
19    fm.setActions(actions);
20    fm.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH
21        + OFActionOutput.MINIMUM_LENGTH));
22    stream.write(fm);
```

Table 2.1: Message Factory Example

## 2.2 OpenFlow Optical Agent

As mentioned above, the main role of this agent is to translate the optical channel requests and OpenFlow messages into TL1 commands to be executed on optical nodes. This agent is added to each optical node and acts as a virtual switch. It consists of an OpenFlow channel to communicate with the OpenFlow controller, OpenFlow/TL1 Translator to convert OpenFlow messages into TL1 commands, Ports-Emulation module to emulate the optical node ports and send this information to the controller to update ports database (Figure 2.2). This information allows the controller to calculate the lightpath.

### 2.2.1 Ports-Emulation Module

This module acts like a virtual switch by creating a list of virtual ports. Each of these virtual ports emulates a physical port of the optical switch (Figure 2.5). This module sends this information to the controller as a way of realization of the optical switch. This module also manages these emulated ports status information[1]. Table 2.2 shows the Java code used to create a *Feature Reply Message* and encode all the virtual ports information. First, it makes the *Message Factory* create a *FeaturesReply* message. Then, it sets the message properties (Line 4-8). After that, it iterates on all physical ports and includes their status into the message (Line 9-19). Finally at line 20, it returns the message.

### 2.2.2 OpenFlow/TL1 Translator

This module is responsible for translating the OpenFlow messages and actions into appropriate commands. Then, it executes these commands on the optical switches. It creates a telnet communication channel with the optical switch to send these TL1 commands through it. The most common operations we use TL1 commands for are creating lightpath, deleting lightpath, retrieving lightpath status, and retrieving port status. The TL1 commands used

[1]Port discovery is out of scope of this research project.

```
01 protected OFMessage createFeatureReplyMsg() {
02  OFMessage featureMsg = factory.getMessage(OFType.FEATURES_REPLY);
03  OFFeaturesReply fReply = (OFFeaturesReply) featureMsg;
04  fReply.setDatapathId(pathId);
05  fReply.setBuffers(100);
06  fReply.setTables((byte) 0x01);
07  fReply.setCapabilities(OFCapabilities.OFPC_FLOW_STATS.getValue());
08  List<OFPhysicalPort> ports = new ArrayList<OFPhysicalPort>();
09  for (int i = 0; i < virtualPorts.length; i++) {
10    \\Iterate on all ports
11    OFPhysicalPort port = new OFPhysicalPort();
12    port.setHardwareAddress(virtualPorts[i].getHardwareAddress());
13    port.setName(virtualPorts[i].getShortName());
14    port.setPortNumber((short) virtualPorts[i].getNumber());
15    port.setCurrentFeatures(OFPortFeatures.OFPPF_FIBER.getValue());
16    port.setState(virtualPorts[i].getOFPortState());
17    ports.add(port);
18  }
19  fReply.setPorts(ports);
20  return fReply;
21 }
```

Table 2.2: Using Message Factory To Create a Feature Reply Message

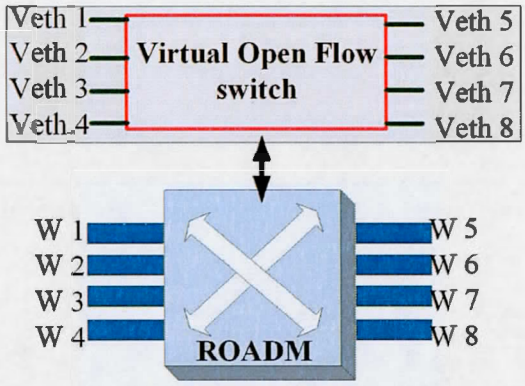



Figure 2.5: Ports-Emulation Module

in our module are discussed in details in Cisco ONS TL1 Command Guide (CISCO, 2012b). Table 2.3 shows a sample of Java code responsible for executing the TL1 create lightpath command. This function receives the ports information from the OpenFlow message. Then, it opens a TL1 session (Line 6). After that, it creates *TL1Command* object and activates the user (Line 7-8). At line 11, it creates the command string. Then at line 13, it sets the command string. At line 14, it sends the command through the TL1 session, created before at line 6, and retrieves the execution result.

```
01 private boolean addLightPath(String portsconnection,short direction){
02  \\ portsconnection =eg=
03  \\ CHAN-1-14-28-RX&LINEWL-1-17-2-RX-1556.55
04  \\ ,LINEWL-1-17-2-TX-1556.55&CHAN-1-13-28-TX
05  try {
06      TL1Session tl1session = new TL1Session(tl1Server, tl1Port);
07      TL1Command req = new TL1Command();
08      req = TL1Command.act_user(tl1Username, tl1Pwd, TID, "300");
09      TL1ResponseMsg msg = (tl1session.send(req))[0];
10      System.out.print("TL1 login mesg:\n" + msg);
11      String commandStr = "ENT-OCHNC:" +TID+":"+portsconnection
12        + ":305::"+direction+"WAY:CKTID=test;";
13      req.setCommand(commandStr);\\ ",CMDMDE=FRCD;"
14      TL1ResponseMsg msg1 = (tl1session.send(req))[0];
15      System.out.print("ENT-OCHNC reply msg:\n" + msg1);
16  } catch (Exception e) {
17      System.err.println("Unable to connect to TL1 agent(server)");
18      e.printStackTrace();
19      return false;
20  }
21  return true;
22 }
```

Table 2.3: Executing TL1 Create Lightpath Command on the Optical Switch

## 2.3 Path Computation Element (PCE)

The objective of this algorithm is to compute a lightpath between source-destination pairs in order to create a fully connected logical topology (Tintor and Radunović, 2012). We have created a Traffic Engineering Database (TED) to save the network topology information. Thanks to the centralized management of the OpenFlow Controller, the TED is always up-to-date. TED will be updated in case of lightpath setup and release (port status changes). Two modules are implemented to achieve our goal; Executor and ONS Adapter (Figure 2.6).

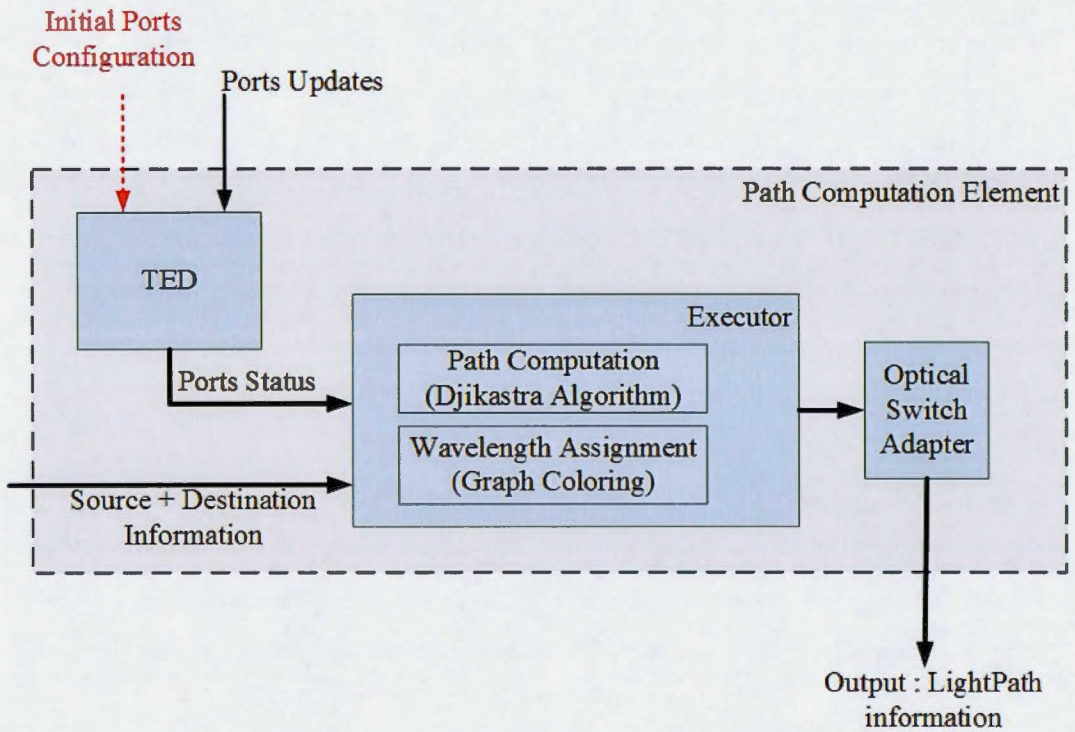



Figure 2.6: Path Computation Element workflow

### 2.3.1 Executor

The objective is to ensure the avoidance of using one wavelength more than one time in the same fiber and the assurance of the wave length continuity through the lightpath. Each

wavelength carries traffic between a source destination pair. Therefore, multiple wavelengths are reserved in a single strand of fiber for establishing multiple lightpaths through one fiber. These connections between the nodes in a WDM networks are done in two steps:

- **Routing:** *Dijkstra Algorithm* is used in order to find the shortest path between each node pair. This algorithm is often used in routing and as a subroutine in other graph algorithms. It solves the single-source shortest path problem for a graph with non-negative edge path costs. In our case, we are interested in a network topology that contains both OpenFlow switches and Optical Network Switches (ONS).

- **Wavelength Assignment :** Once the lightpath routes are determined, the wavelength assignment problem can be represented as a graph coloring problem. In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. 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.

### 2.3.2 ONS Adapter

Each ONS consists of a set of cards and each card contains a set of configured ports (CISCO, 2012a). ONS edges are connected to OpenFlow switches via WSS and DMX cards, whereas ONS core interfaces are interconnected via LINE cards. Two fibers are used for the bidirectional connection between two ONSs. These specifications lead us to add this module.

## 2.4 OpenFlow Message-Mapping Solution

Based on the two units described before, we build our solutions. In the first solution (OpenFlow Message-Mapping), 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. The process of lightpath setup is carried out following these steps:

- The source starts sending data to the destination node.
- Once this flow arrives at an OpenFlow switch, this switch sends a *OFPT_PACKET_IN* message to the controller.
- Upon the receiving of this *OFPT_PACKET_IN* message, the controller requests the PCE unit to compute a lightpath for this packet. The computed lightpath includes the information of the wavelength and the link. This information can not be included in the OpenFlow standard messaging system. In this case, we map the wavelength assignment to a virtual ports as shown before by the OpenFlow agent on section 2.2.
- If the PCE isn't able to compute the path according to lack of resources, it considers this request blocked.
- If the path is computed successfully, the OpenFlow controller sends *OFPT_FLOW_MOD* messages of type (*OFPFC_ADD*) to all the circuit-switched nodes across the lightpath and either a *OFPT_PACKET_OUT* and/or *OFPT_FLOW_MOD* message of type *OFPFC_ADD* to the packet-switched nodes depending on the defined OpenFlow application on the controller.
- When the OpenFlow switch receives the *OFPT_PACKET_OUT* message, it forwards the packet after a considerable delay to be sure that the lightpath is established.
- Once the time-out expires, both the controller and the switches consider the lightpath is established and start to exchange data through the path.

This centralized approach of OpenFlow allows the controller to have the updated nodes and links information stored in the PCE database (TED). The lightpath release mechanism is

fired by the controller using a flow entry time-out timer. This timer is restarted each time this channel is used. When this timeout expires the controller sends a *flow_modification* messages of type (*delete_flow*) to the network nodes causing them to delete this lightpath. The packet-switched nodes will release the path by themselves when the timer expires.

## 2.5 OpenFlow Extension Solution

In this solution, OpenFlow messages are extended and new messages are added. The new messages specification Das (2010) 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*.

In OpenFlow Extension solution the process of lightpath setup is carried out following these steps:

- The source starts sending data to the destination node.
- Once this flow arrives at an OpenFlow switch, this switch sends a *OFPT_PACKET_IN* message to the controller.
- Upon the receiving of this *OFPT_PACKET_IN* message, the controller requests the PCE unit to compute a lightpath for this packet. The computed lightpath includes the information of the wavelength and the link. This information is included in the OpenFlow new structure *OFP_CONNECT* and encapsulated in the new message *OFPT_CFLOW_MOD*. the *OFP_CONNECT* structure could carry the bidirectional lightpath instead of sending one message for each direction.

- If the PCE is not able to compute the path according to lack of resources, it considers this request blocked.
- If the path is computed successfully, The OpenFlow controller sends *OFPT_CFLOW_MOD* messages of type (*OFPFC_ADD*) to all the circuit-switched nodes across the lightpath and either a *OFPT_PACKET_OUT* and/or *OFPT_FLOW_MOD* message of type *OFPFC_ADD* to the packet-switched nodes depending on the defined OpenFlow application on the controller.
- When the OpenFlow switch receives the *OFPT_PACKET_OUT* message, it forwards the packet after a considerable delay to be sure that the lightpath is established.
- Once the time-out expires, both the controller and the switches consider the lightpath is established and start to exchange data through the path.

Similar as the first solution, the lightpath release mechanism is fired by the controller using a flow entry time-out timer. This timer is restarted each time this channel is used. When this timeout expires the controller sends a *OFPT_CFLOW_MOD* messages of type (*OFP_DELETE*) to the network nodes causing them to delete the lightpath. The packet-switched nodes will release the path by themselves when the timer expire.

## 2.6 GMPLS WITH PCE LIGHTPATH SETUP

GMPLS is presented in this thesis for the purpose of comparing it with our proposed solutions. To be able to accomplish this comparison, we have to understand the GMPLS the lightpath establishment procedures. As we mentioned before, GMPLS is a distributed protocol. Using a distributed protocol on large networks makes the path computation process very complex and resources consuming. To address this problem, IETF has introduced a centralized Path Computation Element (PCE) entity in the GMPLS control plane. In this thesis, we implement the GMPLS with PCE. The PCE is a centralized network element responsible for computing the lightpath. In this topology, PCE also assigns wavelength on each link for each request. The PCE is used in GMPLS-controlled Wavelength Switched Optical Network (WSON) (Li et al., 2012; López et al., 2010). PCE uses a messaging protocol

called PCEP to exchange information between GMPLS controller of each node and the PCE. PCE maintains the information of the nodes, links status and wavelength availability in a database called Traffic Engineering Database (TED) The links update is carried out by the OSPF messaging (Link State Advertisements - LSAs). This updates are sent when a new wavelength status change occurs (reserve/release). A full link status update occurs when new node joins or leaves the network. However, to keep the network stable, LSAs are not sent each time update happened. For each link, once an LSA has been generated, a time-out timer starts. During this time-out, no update is sent for this link. Following in detail the message sequence on GMPLS with PCE mechanism to create a lightpath:

- The source node sends a PCEP request message for submitting a path computation request.
- The PCE computes the path requested and assigns a wavelength to this path. Then, the PCE sends this information to the source by using a PCEP PCRep message. Otherwise, if the PCE fails in computing a path or in assigning a wavelength on it, it replies with a PCRep message with NO-PATH reply, and the lightpath request is refused (forward-blocking).
- Upon the reception of PCRep message, the source node sends the Resource Reservation Protocol-Traffic Engineering (RSVP-TE) messages along the computed path to reserve it. The Path reservation message includes the Explicit Route and the Label set. The label set information includes the wavelength assigned by the PCE.
- When a node receives RSVP-TE path reservation message, it performs the wavelength assignment if it is available. Otherwise, another wavelength contained in the Label Set is selected, according to a specific wavelength assignment strategy (e.g., first fit).
- If another request requests the same resource (link and wavelength) on a specific node and this request is accomplished before this one, this will have this node to refuse this request and reply with RSVP refuse message (backward-blocking).
- When the wavelength assigned, the destination node sends back a Resv message to effectively reserve the selected wavelength on each link of the path.

- Once the Resv message reaches the source, the lightpath is established and data can be carried through the path.

Lightpath release is performed in a similar way as the setup process (in a distributed manner through RSVP-TE signaling (Giorgetti et al., 2009)). As the previous description the setup procedure may be blocked during path computation because of lack of resources (forward blocking), or may be blocked due to wavelength contention (backward-blocking). Contentions arrive when two or more RSVP-TE messages attempt to reserve the same resource (link and wavelength). This actually because the link availability database TED may be outdated when the path request reached the PCE.

# CHAPTER III

# CONDUCTED EXPERIMENTS

Two experiments are conducted to demonstrate the efficacy of our proposed solutions. The first experiment is to create end-to-end lightpath while the second is to create a backup restoration lightpath in case of the failure of the primary lightpath.

## 3.1 Testbed Setup

The architecture of our testbed is depicted in figure 3.1. It consists of two clients A and B, which are connected directly to OpenFlow (OF) 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).

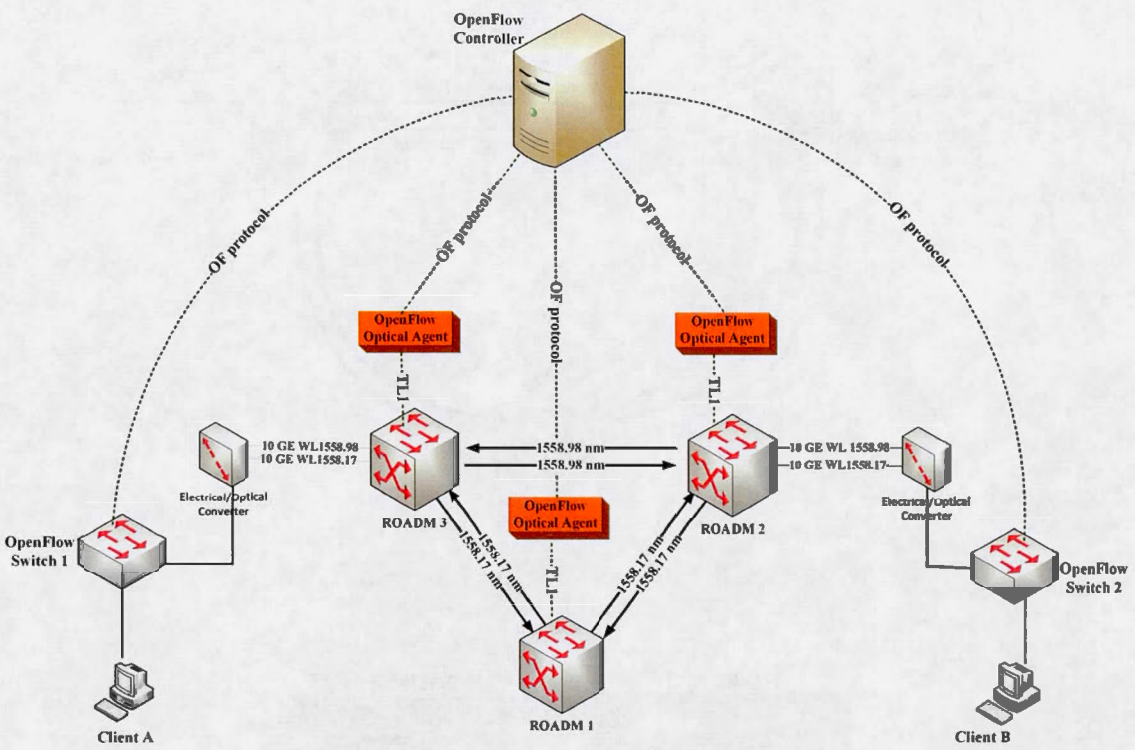



Figure 3.1: Testbed Architecture

The ROADM switches are connected on full mesh topology using optical fiber cables as shown in figure 3.2. Each fiber cable is 10 meter long, and supports 32 channels. Each ROADM is controlled by an OpenFlow Optical Agent. The OpenFlow optical agents and the OpenFlow switches are connected to an OF controller over an OpenFlow channel as shown in figure 3.1.

Figure 3.3 shows a photo of the physical equipments in our lab (Optical Transport Network Laboratory) that is used in our experiments.

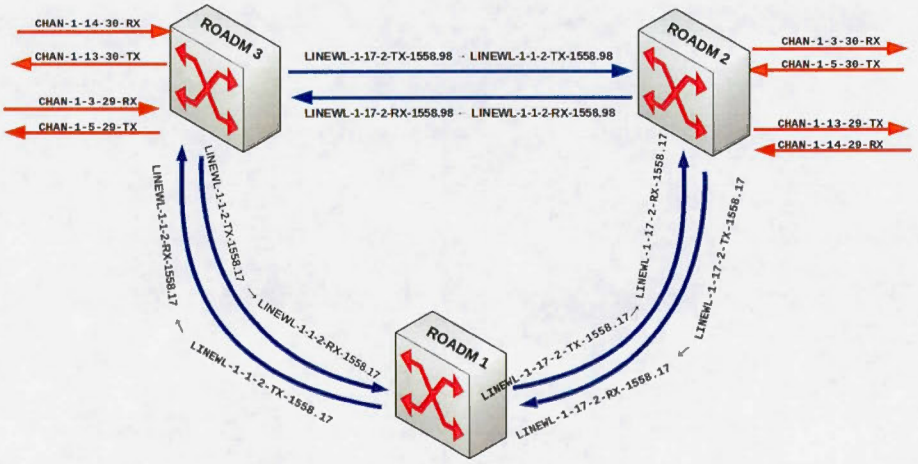



Figure 3.2: Optical domain Interconnection

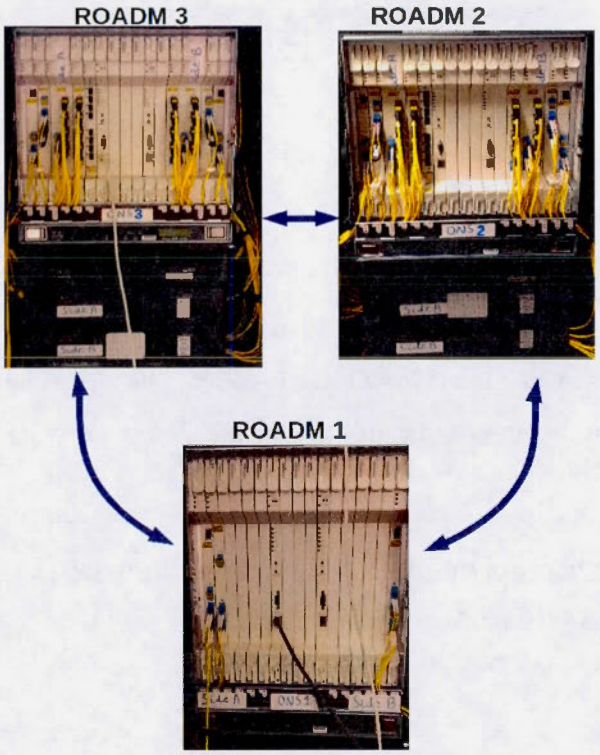



Figure 3.3: physical equipments in the Optical Transport Network Laboratory

### 3.2 Scenario 1: End-to-End Lightpath Setup and Release

The purpose of this experiment is to test the capability of the proposed solution to compute and establish a lightpath when required. As shown in the architecture in figure 3.1, when the optical OpenFlow agent is connected to the controller, it acts as an OpenFlow switch by sending *Hello* message followed by *FeaturesReply* message which is shown in the Wireshark screenshot in Figure 3.4.

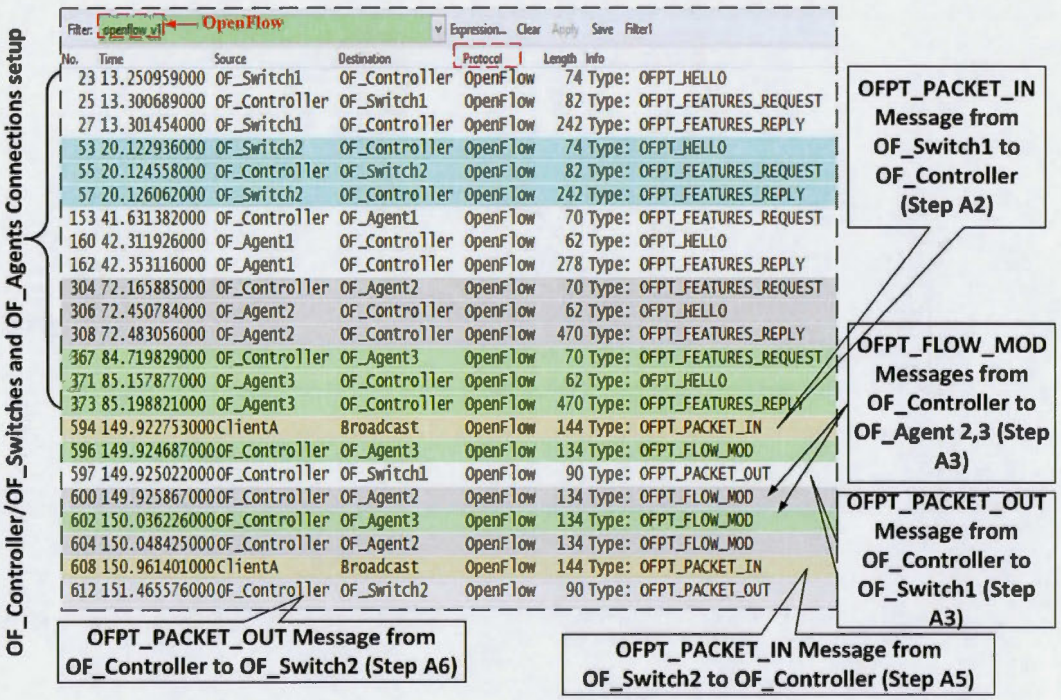



Figure 3.4: Wireshark Screenshot (Lightpath Setup Message Exchange)

As shown in Figure 3.5, 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 3.5). The wireshark screenshot presents the exchanged messages during this scenario (Figure 3.4).

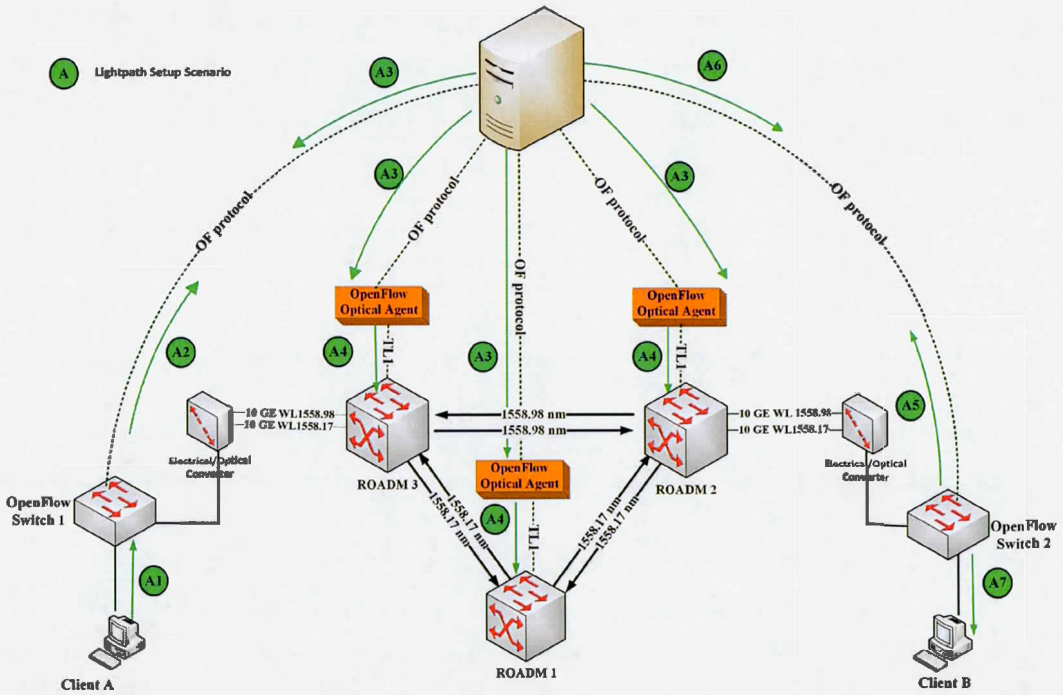



Figure 3.5: Network configuration and message exchange

As shown in figure 3.5 the connection is established between clients, A and B as the following scenario:

Step A1: A data flow sent from client A to client B arrives at OpenFlow switch 1.

Step A2: 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 as shown in Figure 3.4.

Step A3: 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 (Message exchange shown in Wireshark screenshot in figure 3.4).

Step A4: When OpenFlow optical agents receive this message, they translate them into the appropriate TL1 commands and send it to the ROADM switches.

Step A5: After creating the lightpath, the data flow traverses until OpenFlow switch 2. When the flow is received by OpenFlow switch 2, the switch does not find a flow entry in its flow table to forward this flow. Then, it sends a *OFPT_PACKET_IN* message to the controller requesting an action for this flow (as shown in Figure 3.4).

Step A6: The controller sends an *OFPT_PACKET_OUT* message to OpenFlow switch 2 to forward this packet to client B.

Step 7: OpenFlow switch 2 forwards this flow to client B..

Cisco Transport Controller (CTC) screenshot, in the initial state while no lightpath existed, is depicted in figure 3.6. Figure 3.6 depicts Cisco Transport Controller (CTC) screenshot showing the optical channel setup on wavelength 1558.98 nm after lightpath creation.

This scenario is explained by UML diagram in Figure 3.8.

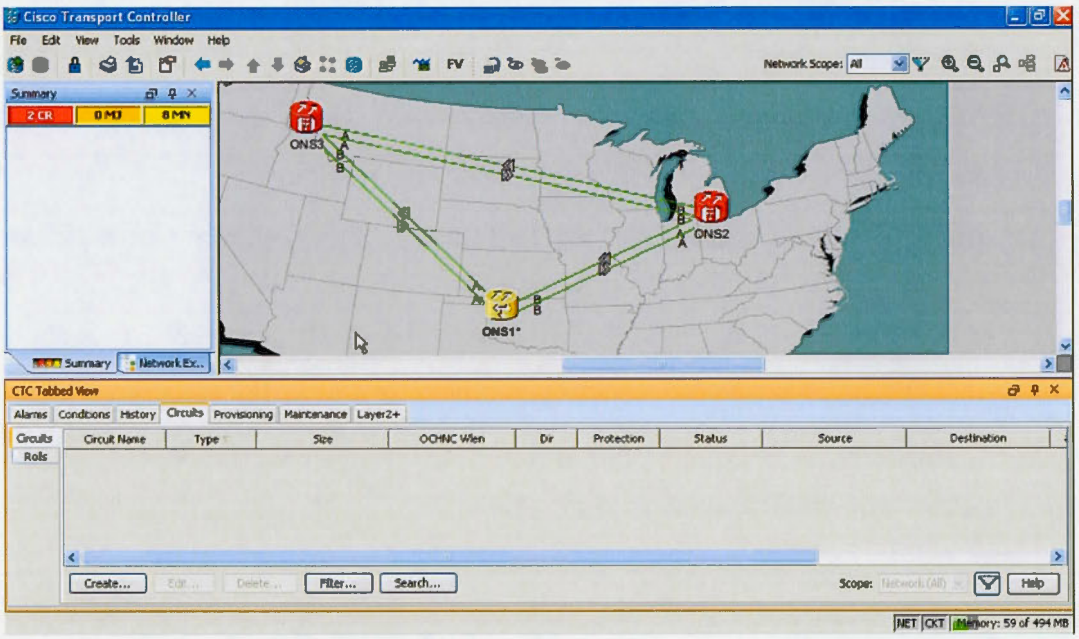



Figure 3.6: Cisco Transport Controller Screenshot (Initial State)

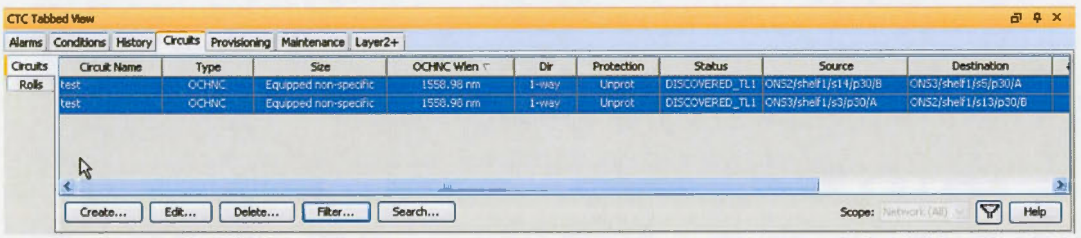



Figure 3.7: Cisco Transport Controller Screenshot (After Lightpath Establishment)

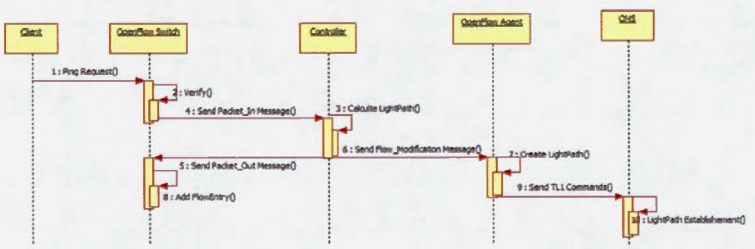



Figure 3.8: UML diagram for lightpath establishment

## 3.3 Scenario 2: Backup lightpath Restoration

This scenario demonstrates how OpenFlow controller acts in case of link failure. Figure 3.9 shows the steps that are executed in this scenario.

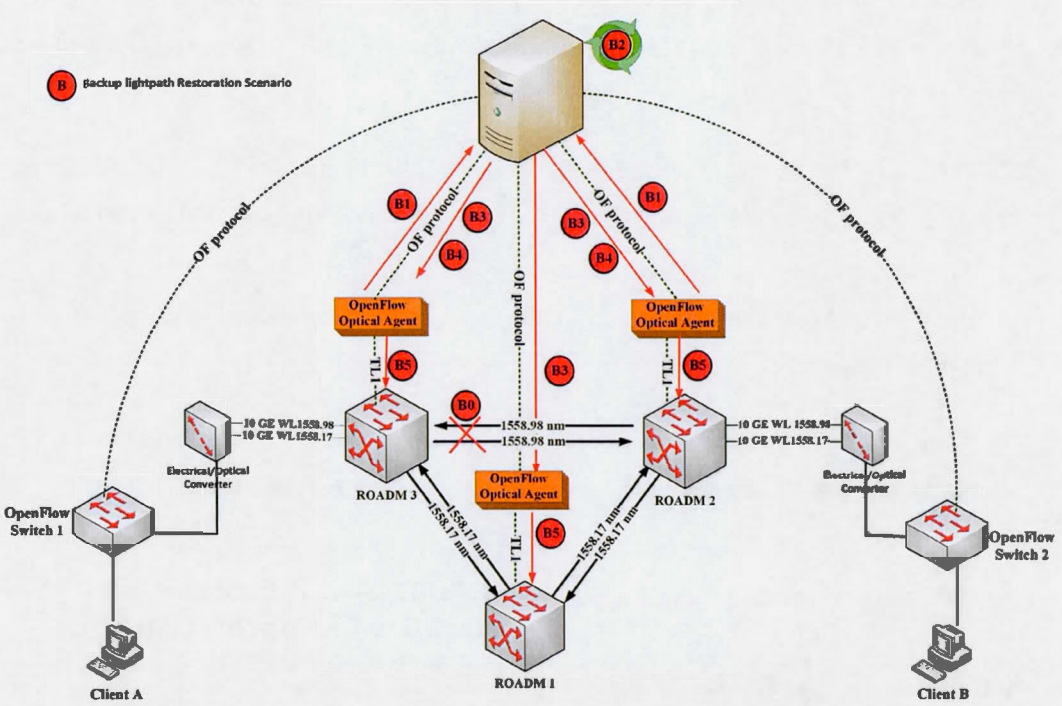



Figure 3.9: Exchanged messages Backup lightpath Restoration Scenario

Step B0: To simulate the link failure, we unplug the fiber cable between ROADM 2 and ROADM 3.

Step B1: When the fiber connection between the ROADM 3 and ROADM 2 fails, both OpenFlow optical agents corresponding to these Optical Switches read the alarms of the optical switches using TL1 *(RTRV-ALM-ALL)* command [1]. Then, they send *OFPT_PORT_STATUS* Messages to the controller about the port status update (Message exchange is shown in figure 3.10).

[1] The mechanism of detecting link failure is out of scope of this work

Step 2B: OpenFlow controller calculates alternative lightpaths for the existing failed lightpaths.

Step 3B: The controller sends *OFPT_FLOW_MOD* (type=OFPT_ADD) messages (OpenFlow Messages Mapping solution) or *OFPT_CFLOW_MOD* messages (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 (1588.17 nm) (Message exchange shown in Wireshark screenshot in figure 3.10).

Step 4B: The controller sends other *OFPT_FLOW_MOD* (Type=OFPFC_DELETE) messages (OpenFlow Messages Mapping solution) or *OFPT_CFLOW_MOD* messages (OpenFlow Extension Solution) to the optical switches which are associated with old lightpath to delete the primary lightpath (Message exchange shown in Wireshark screenshot in figure 3.10).

Step 5B: When the OpenFlow Optical agents receive these messages, they translate them into the appropriate TL1 commands and send them to the optical switches.

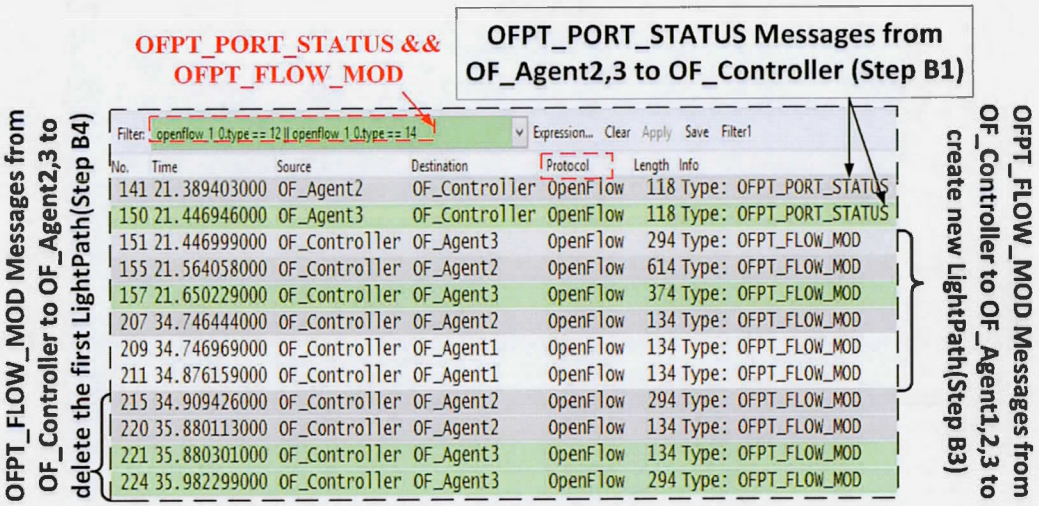



Figure 3.10: Wireshark Screenshot (Lightpath Setup Message Exchange)

Figure 3.11 shows CTC screenshot after the lightpath restoration is done, representing the optical channel setup on a different wavelength 1558.17 nm.

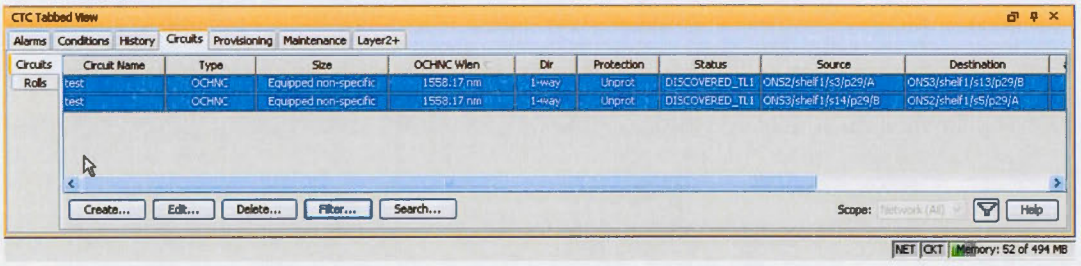



Figure 3.11: Cisco Transport Controller Screenshot (After Lightpath Restoration)

This second scenario is explained by this UML diagram in Figure 3.12.

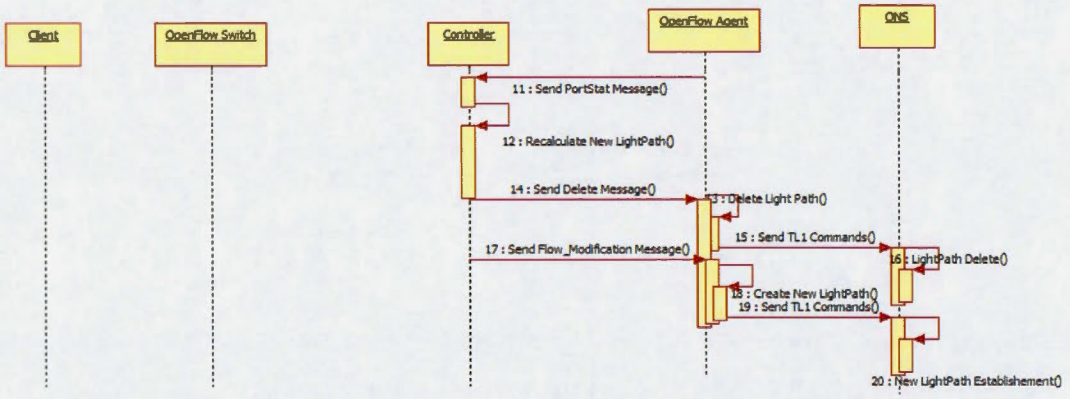



Figure 3.12: UML diagram of lightpath recovery

## 3.4 GMPLS Approach Experiment

Still GMPLS is not deployed commercially. Dynamic Resource Allocation via GMPLSnOptical Networks (DRAGON) software extends the network equipment using SNMP to adapt this equipment to GMPLS control plane. The DRAGON project studies and develops an open source software to enable dynamic provisioning of network resources on an inter-domain basis across heterogeneous network technologies. The project enables the communication between networks of different types through the GMPLS control suite. The extension

of the DRAGON project to support CISCO 15454 ROADM is conducted by a colleague in another research project [2]. Further information about DRAGON project is available in (Lehman et al., 2006a) and (Lehman et al., 2006b). To experiment GMPLS, we construct a transparent optical network testbed with two ROADMs (Figure 3.13). In this infrastruc-

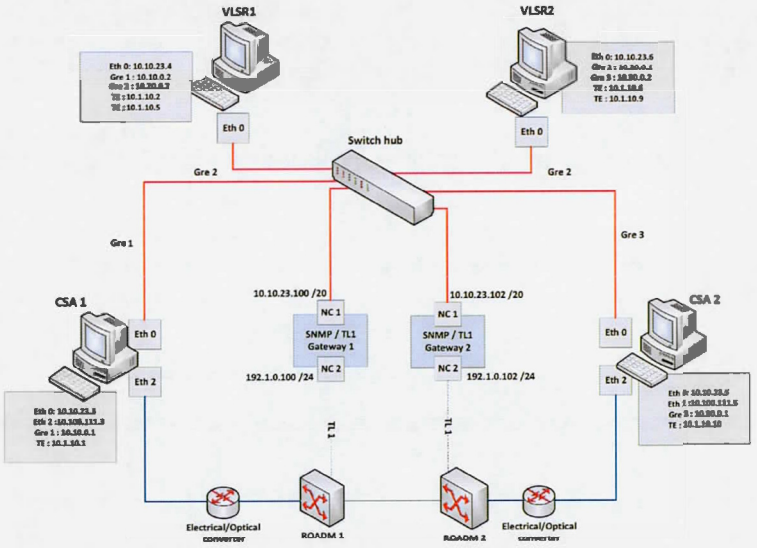
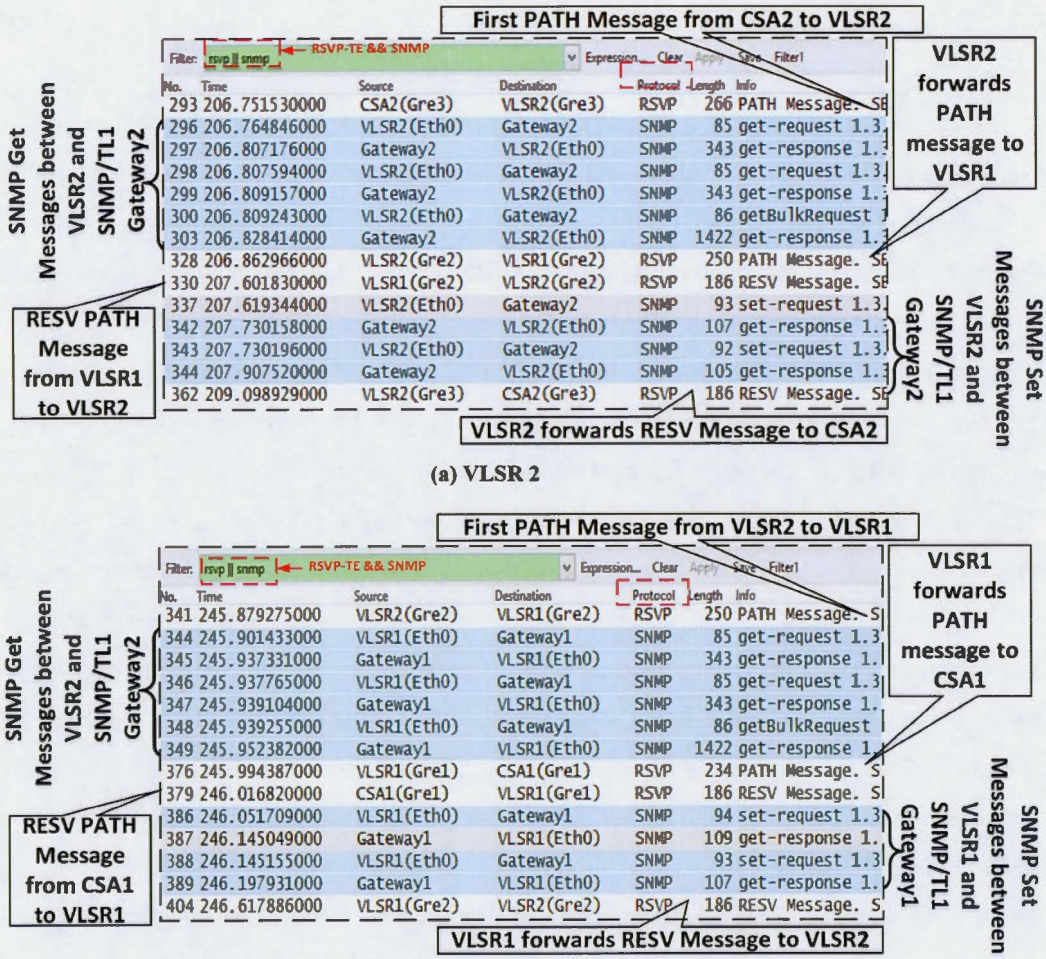



Figure 3.13: GMPLS Experiment Using DRAGON

ture, the control plane consists of two Client System Agents (CSA) and two Virtual Label Switch Routers (VLSR). The CSAs and the VLSRs are connected via a 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. It translates SNMP messages to TL1 commandes in order to configure the ROADMs. In the SNMP/TL1 Gateway machine, we install two machines. Each one listens to a VLSR on port 161 and controls one ROADM. Using wireshark capture in VLSR2 (Figure 3.14 (a)) and VLSR1 (Figure 3.14 (b)), we explain the GMPLS signaling to create a Label Switched Path (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 re-

[2]The Configuration and the mechanism of extending DRAGON to support ROADM is out of scope of this research project.




(a) VLSR 2

(b) VLSR 1

Figure 3.14: GMPLS Scenario : Wireshark screenshot

ceives 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.

### 3.5 Experimentation Results

The experiments setup time (in millisecond) is shown in table 3.1 for OpenFlow solutions (***OpenFlow Messages Mapping*** and ***OpenFlow Extension***) and the GMPLS approach. In this table, Path1 and Path2 refer to the primary and the backup lightpaths, respectively. 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 (with 216 ms setup time) outperforms ***OpenFlow Messages Mapping*** solution (with 227 ms setup time). 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 spans on three nodes, ***OpenFlow Extension*** solution takes 239 ms to create the lightpath while ***OpenFlow Messages Mapping*** takes 269 ms. 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. 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 signaling 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.

| | 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 3.1: The experiments timing

# CHAPTER IV

# SIMULATION STUDY

In this chapter we present a comparative study of the OpenFlow solutions (OpenFlow Messages-Mapping, OpenFlow Extension) and the GMPLS approach. To conduct the comparison, A custom-built Java event-driven simulator is written based on the mechanisms mentioned in chapter 2. The measurements taken from the previously conducted experiments are used in writing a custom-built Java event-driven simulator.

Table 4.1 shows the signaling protocol used by each solution. In this table the signaling protocol is in the first row, and in front of each solution we marked which signaling protocol is used in it.

The simulation is carried out on two real optical network topologies. These network topologies are the physical network topology of United States National Science Foundation (NSF) and the optical network topology of the European Union Ultra-High Capacity Optical

| | messaging protocol | | |
|---|---|---|---|
| | OFP | OSPF-TE | RSVP-TE |
| GMPLS with PCE | NO | YES | YES |
| OpenFlow Message-Mapping solution | YES | NO | NO |
| OpenFlow Extension solution | YES | NO | NO |

Table 4.1: Summary of Simulated Solutions

Transmission Network (European Research Project Cost239). The next section presents the simulation environment, parameters and algorithms. Then, the results for each network are presented in sections 4.2 and 4.3.

## 4.1 The Custom-built Java Event-Driven Simulator

The simulator is a custom-built Java event-driven application. It is written based on the mechanisms mentioned in Chapter 2. The internal optical switch lightpath establishment time is emulated to 60 ms for all solutions. For both topologies the links between nodes are bidirectional. Each link supports 32 wavelengths. The controller and the PCE perform first-fit for assigning wavelengths. Wavelength can not be changed across the path since nodes do not support wavelength conversion. Lightpath requests are generated according to a Poisson process and uniformly distributed among all node pairs. The holding time is fixed to 180 seconds, the average inter-arrival time is varied from 0.3 s to 18 s. This varies the Erlang from 600 to 10.

Algorithm 1 explains how the written application simulates the OpenFlow solutions. The application uses the network topology nodes (G), the connections between them (V), and the simulation end-time as inputs. Then, it starts by generating one event of type create-channel. After that, it reads events one at a time and handles it. Depending on the event type, each event type is treated differently by the algorithm, as explained before. For the create-channel event, it generates a new create-channel event based on the Poisson inter-arrival time, updates the controller's time, calculates the lightpath, finds a free channel (wavelength). Finally, it generates the "create cross-connect" events for each switch through the calculated path (events to be executed by the switches). Unless there is no lightpath available, it declares this request as a blocked request. For the events of type Delete channel, it updates the controller's time. Then, it generates the delete cross-connect events for each switch through the lightpath (events to be executed by the switches). For the event of type "create cross-connect", it generates an event of type delete channel. For both events of type create/delete cross-connect, it updates nodes time (emulating the cross-connect creation time 60 ms). Then, it updates vertex information. The cross-connect creation time is calculated

from the testbed experiments by measuring the time difference between sending the lightpath creation TL1 command and the response received from the optical switch after executing the command.

**Data**: G: Graph, V: vertex, EndTime: Simulation End Time

**Result**: Establishment time, Blocking probability and control traffic

Initialization: Generate one event (using a uniformly distributed source and destination and Poisson inter-arrival time);

**while** *current time* < *EndTime* **do**

read the nearest event;

**switch** *Event Type* **do**

**case** *Create Channel*

Generate new Create Channel event based on Poisson inter-arrival time;

Update the controller's time;

Update the controller's vertex information;

Calculate path using Dijkstra Algorithm;

Find a free channel (wavelength) cross the calculated path;

**if** *Path calculation return false OR no channel available* **then**

Declare Request Blocked;

Continue with the next event;

**else**

Generate "create cross-connect" events for each node through the calculated path (with the information of event time, path and wavelength);

**end**

**end**

**case** *Delete Channel*

Update the controller's time;

Update the controller's vertex information;

Generate delete Cross-Connect events for each node through the calculated path (with the information of event time, path and wavelength);

**end**

**case** *Create Cross-Connect*

Update nodes' time (emulating the cross-connect creation time 60 ms);

Update vertex information;

Generate delete event for the created path (with event time = current time + hold time);

**end**

**case** *Delete Cross-Connect*

Update nodes' time (emulating the cross-connect creation time 60 ms);

Update vertex information;

**end**

**endsw**

GMPLS simulation is divided into three algorithms 2,3 and 4 (Main algorithm and two event handler procedures). These algorithms explain how the written application simulates the GMPLS with PCE approach. In this algorithm the inputs and the initialization are the same as algorithm 1. By traversing all the events depending on their types, each event type is treated differently as explained on algorithms 2, 3 and 4. For the create-channel event, it generates a new create-channel event based on the Poisson inter-arrival time, updates the controller time, calculates the lightpath, finds a free channel (wavelength), finally it generates the "create cross-connect" events for the first switch in the calculated path (event to be executed by the switch). Unless there is no lightpath available, it declares this request as a blocked request. For the events of types Delete channel, it updates the controller's time. Then, it generates the delete cross-connect events for the first switch in the lightpath (event to be executed by the switch). For both events of type create/delete cross-connect, it updates node time (Emulating the cross-connect creation time 60 ms). Then, it updates vertex information. For the event of type "create cross-connect", it verifies if the requested channel is available. If it is not available, it declares this request blocked (Backward Blocking) and it generates delete channel request. If it is available and this is not the last switch in the lightpath, it generates an event of type "create cross-connect" for the next switch in the lightpath, otherwise it generates an event of type delete channel. For both events of type LSA update (create/delete), it updates TED (controller Vertex information).

**Data**: G: Graph, V: vertex, EndTime: Simulation End Time

**Result**: Establishment time, Blocking probability and control traffic

Initialization: Generate one event (using a uniformly distributed source and destination and Poisson inter-arrival time);

**while** *current time* < *EndTime* **do**

  read the nearest event;

  **if** *Event Type* == *Create Channel* **then** Generate one event based on Poisson inter-arrival time ;

  **switch** *Event Type* **do**

    **case** *Create/Delete Channel*

      Call Channel Event-Handler procedure;

    **end**

    **case** *Create/Delete Cross-Connect*

      Call Cross-Connect Event-Handler procedure;

    **end**

    **case** *LSA update (Create/Delete)*

      Update TED (controller Vertex information);

    **end**

  **endsw**

**end**

**Algorithm 2:** GMPLS/PCE Event-Driven Simulator algorithms

```
switch Event Type do
    case Create Channel
        Update the controller's time;
        Calculate path using Dijkstra Algorithm;
        Find a free channel (wavelength) cross the calculated path;
        if Path calculation return false OR no channel available then
            Declare Request Blocked; Continue with the next event;
        else
            Generate "create cross-connect" event for the first node in the calculated path
            (with the information of event time, path and wavelength);
        end
    end
    case Delete Channel
        Update the controller's time;
        Generate delete Cross-Connect event for first node in the calculated path (with the
        information of event time, path and wavelength);
    end
endsw
```

**Algorithm 3:** Channel Event-Handler procedure

```
switch Event Type do
    case Create Cross-Connect
        Update nodes time (emulating the cross-connect creation time 60 ms);
        Update switch's vertex occupation;
        if current switch is the last one in the path then
            Generate delete event for the created path (with event time = current time + hold
            time);
        else
            if channel (wavelength) is available then
                Generate "create cross-connect" event for the next node in the calculated path;
            else
                Declare this request blocked;
                Generate delete channel event
            end
        end
        Generate LAS update (Create) event;
    end
    case Delete Cross-Connect
        Update nodes time (emulating the cross-connect creation time 60 ms);
        Update switch's vertex occupation;
        if current switch is not the last on the path then  Generate delete Cross-Connect
        event for the next node in the calculated path ;
        Generate LAS update (Delete) event;
    end
endsw
```

**Algorithm 4:** Cross-Connect Event-Handler procedure

## 4.2 National Science Foundation (NSF) topology

Figure 4.1 shows the network topology of the National Science Foundation (NSF) topology Foundation (2014). NSF topology consists of 14 nodes and 21 links, each link has 32 channels (wavelengths) (Figure 4.1). The distances between nodes are shown in the figure. Dijkstra algorithm uses these distances to calculate the shortest paths.

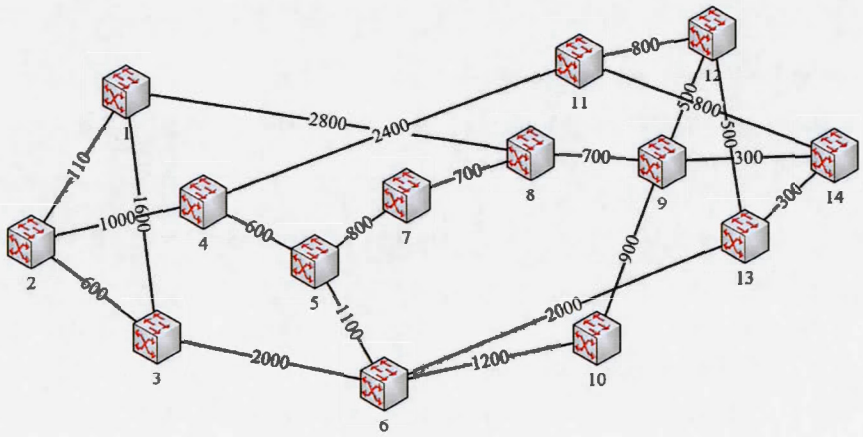



Figure 4.1: NSF topology (14 nodes and 21 links)

The simulation is run for a period of 3000 sec to ensure the stability of the network. Lightpath establishment time, control traffic got into and out of the controller and PCE, and the blocking probability are calculated from the simulation. The results are shown in the graphs : (i) Lightpath establishment time expressed in milliseconds vs. network load (Erlang) (Figure 4.2); (ii) Number of control messages (Controller load) vs. network load (Erlang) (Figure 4.4); (iii) Lightpath blocking probability vs. network load (Erlang) (Figure 4.5).

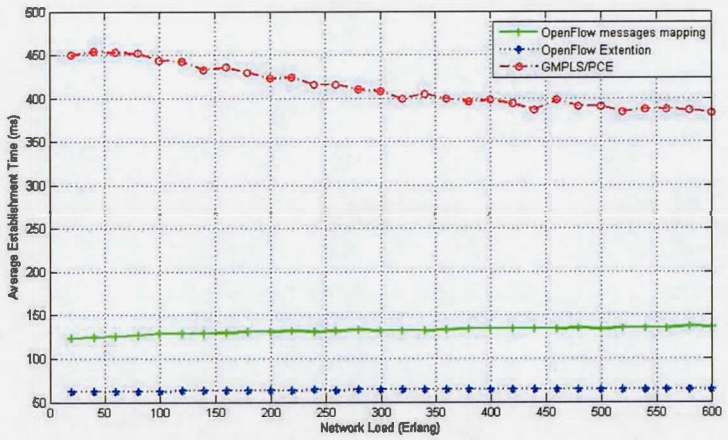



Figure 4.2: Lightpath establishment time [ms] vs. network load (NSF topology)

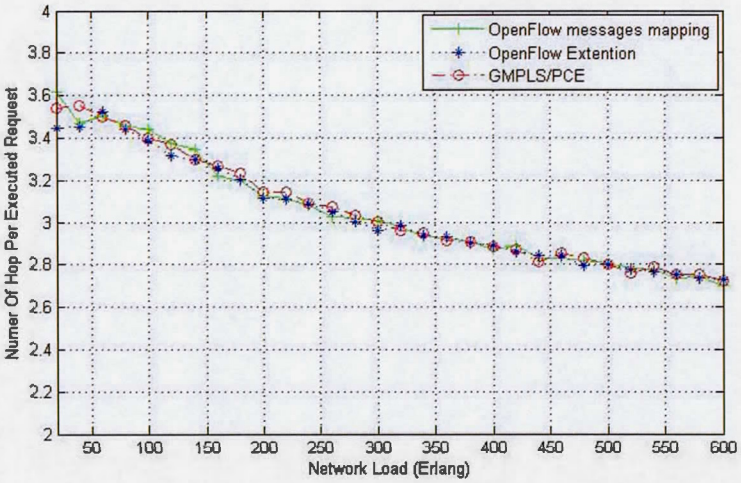



Figure 4.3: Number Of Hop Per Request vs. network load (NSF topology)

Figure 4.2 depicts the establishment time for bidirectional lightpath. It shows that OpenFlow Extension solution experiences the lowest setup time as shown with blue line. Because OpenFlow Message-Mapping uses two *FLOWMOD* messages to establish the lightpath, it is expected that this solution experiences higher time than OpenFlow Extension solution as shown in the figure with the red line. OpenFlow solutions execute the lightpaths on parallel, Hence the establishment time of lightpath is around a fixed value. On the other hand, GMPLS approach executes the lightpaths sequentially, Hence it has higher establishment time. As a result, GMPLS has the highest setup time as shown in the figure with the green line in the range 600- 900 ms for bidirectional lightpaths.

GMPLS has the tendency to decrease the establishment time as the network load increases. Because at high network load the average path length is shorter as shown in figure 4.3 (it decreases from 3.6 to 2.6 nodes per request). Even though the number of hop decreases too on OpenFlow-based solutions, this do not affect the lightpath setup since the requests is executed in parallel.

Figure 4.4 depicts the control traffic for each solution. It shows that both OpenFlow solutions experience low control traffic compared to GMPLS solution as shown by blue and green lines. This difference between the OpenFlow solutions and GMPLS solution due to the

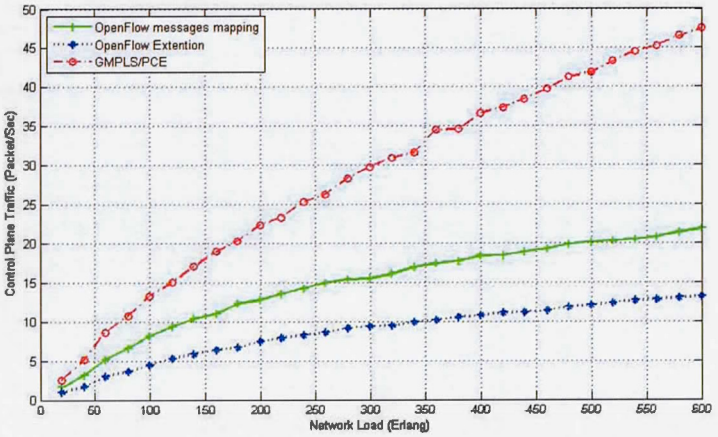



Figure 4.4: Number of control messages vs. network load (NSF topology)

PCEP messaging has to be sent for each node and also because of the LSA update messages which each node has to send back to the controller in case link state changes.

Figure 4.5 depicts the blocking probability. This figure shows that both OpenFlow based solutions have the same blocking probability values which are expected since both techniques use the same Dijkstra algorithm and the same resource Database. On the other hand, GMPLS-based approach experiences the backward-blocking which makes this technique have higher blocking ratio with low network load as shown in the figure with green line. As we mentioned before, the backward-blocking occurs because of wavelength contentions. Contentions arrive when two or more RSVP-TE messages attempt to reserve the same resource (link and wavelength). Indeed, the link state database TED may be outdated when the path request reaches PCE causing this contention.

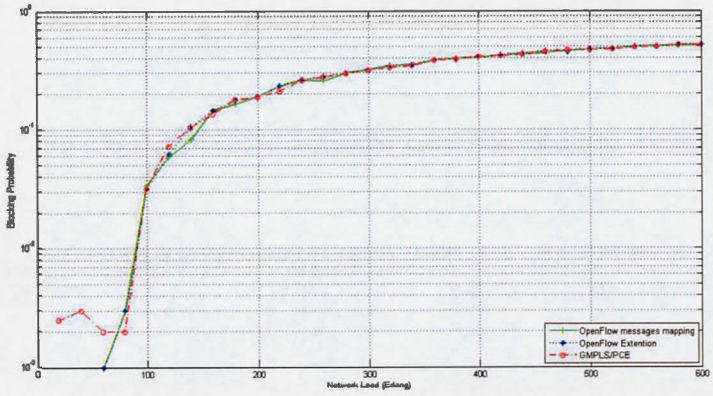



Figure 4.5: Lightpath blocking probability vs. network load (NSF topology)

## 4.3 European Optical Network Topology (COST239)

Ultra-High Capacity Optical Transmission Network (European Re-search Project Cost239) O'Mahony (1996) is the second topology we ran our simulation on. This topology is depicted on Figure 4.6.

COST239 topology consists of 11 nodes and 26 links, each link has 32 channels (wavelengths). The distances between each pairs are shown in the figure. Dijkstra algorithm uses these distances to calculate the shortest paths.

The same simulation steps are followed as the NSF topology. The simulation is run for a period of 3000 sec to ensure the stability of the network. Lightpath establishment time, control traffic got into and out of the controller and PCE, and the blocking probability are calculated from the simulation. The results are shown in the graphs : (i) Lightpath establishment time expressed in milliseconds vs. network load (Erlang) (Figure 4.7); (ii) Number of control messages (Controller load) vs. network load (Erlang) (Figure 4.9); (iii) Lightpath blocking probability vs. network load (Erlang) (Figure 4.10).

The results shown in figure 4.7 support the same result of the NSF topology. It depicts

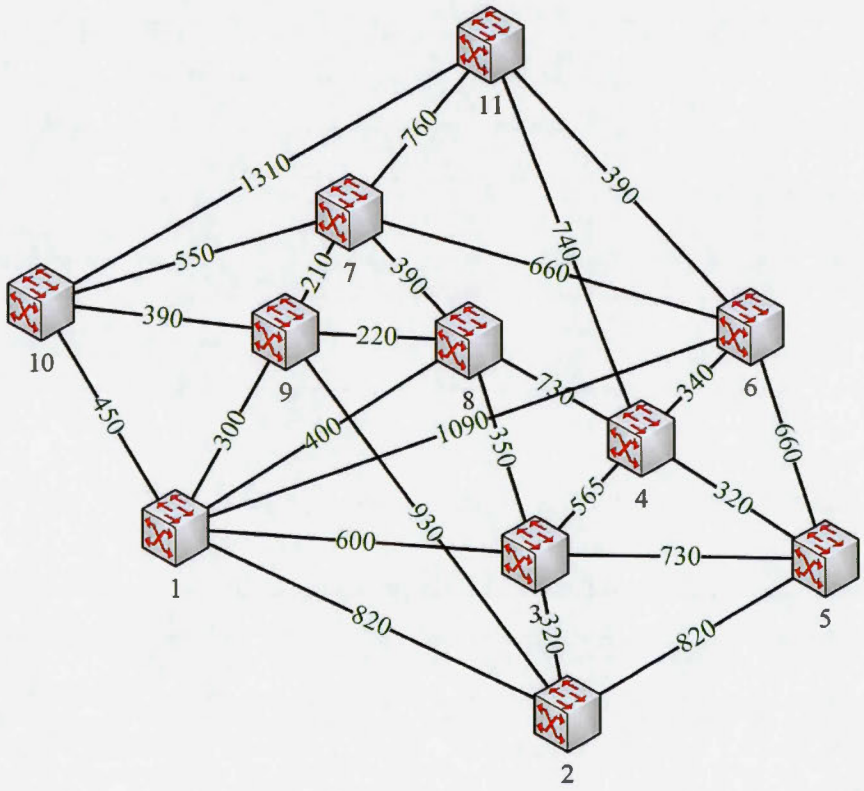



Figure 4.6: COST239 Topology (11 nodes and 26 links)

that OpenFlow Extension solution experiences the lowest setup time as shown with blue line. It depicts also that GMPLS has the highest setup time as shown in the same figure with green line.

As the previous topology, the figure shows that GMPLS lightpath establishment time decreases as the network load increases, because at high network load the average path length is shorter as shown in figure 4.8 (it decreases from 2.77 to 2.34 hop per request).

Figure 4.9 depicts the control messages for each solution. It confirms the result we got on the NSF topology. It shows that OpenFlow solutions experience the lowest control traffic. It depicts also that GMPLS has the highest control traffic as shown in the same figure with the green line.

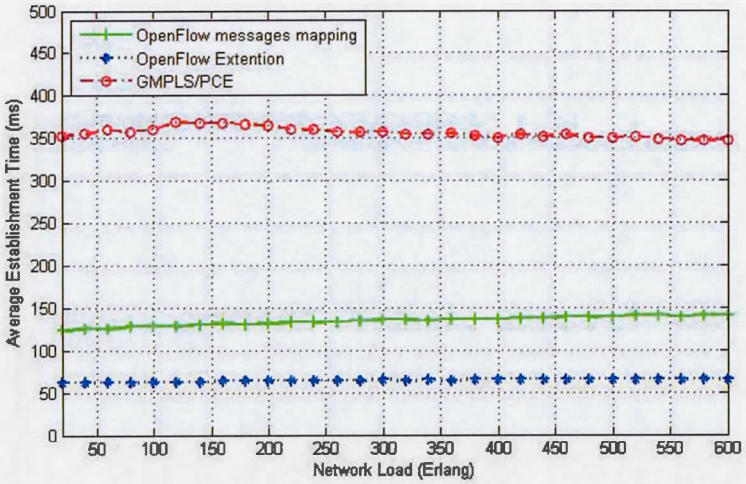



Figure 4.7: Lightpath establishment time [ms] vs. network load (COST239 Topology)

Figure 4.10 depicts the blocking probability and it also confirms the result we got on the NSF topology. This figure shows that both OpenFlow based solutions have almost the same blocking probability values. On the other hand, GMPLS protocol experiences the backward-blocking which makes this technique have higher blocking ratio with low network load as shown in the figure with green line.

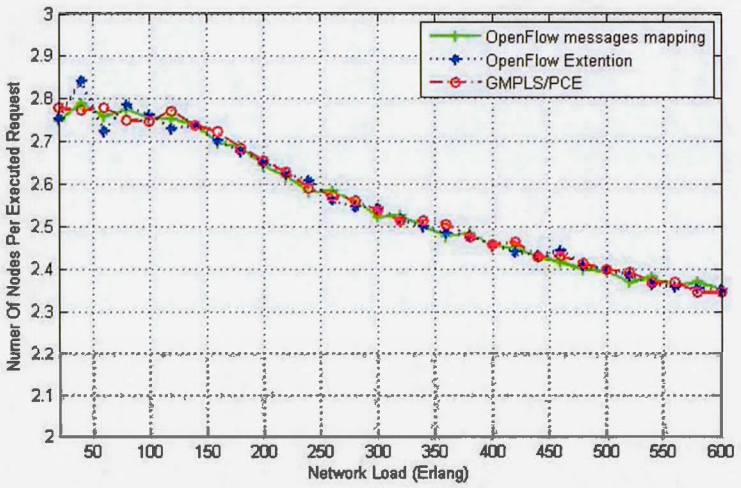



Figure 4.8: Number Of Hop Per Request vs. network load (COST239 Topology)

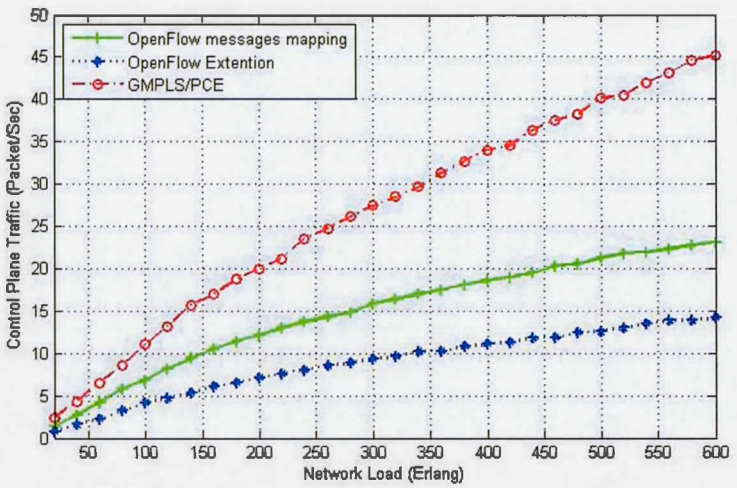



Figure 4.9: Number of control messages vs. network load (COST239 Topology)

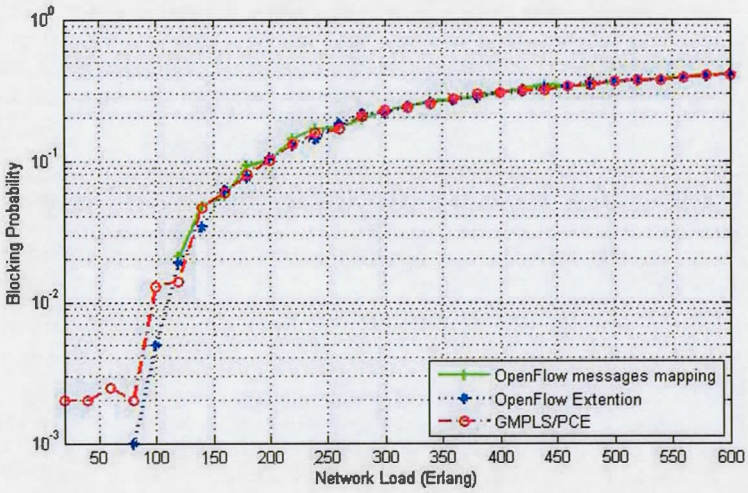



Figure 4.10: Lightpath blocking probability vs. network load (COST239 Topology)

## 4.4 Summary of Simulation Results

The simulations reveals that OpenFlow extension solution outperforms both GMPLS with PCE and OpenFlow message-mapping solution in lightpath establishment time and controle plane traffic. It also outperforms GMPLS in blocking probability, while it has almost the same blocking probability as OpenFlow message-mapping solution. Table 4.2 shows the summary of the simulation results performed on NSF network topology.

| | GMPLS with PCE | OpenFlow Message-Mapping Solution | OpenFlow Extension Solution |
|---|---|---|---|
| Establishment time | Between 450 ms and 380 ms | Around 170 ms | Around 60 ms |
| Control plane traffic | increases till 47 packets/sec at maximum load | increases till 22 packets/sec at maximum load | increases till 13 packets/sec at maximum load |
| Blocking probability | starts at 0.003, and reaches 0.5 | starts at 0, and reaches 0.5 | starts at 0, and reaches 0.5 |

Table 4.2: Summary of NSF topology simulation results

The simulations is repeated using COST239 physical network topology and the results support what we had before using NSF topology (Table 4.3).

In brief, the experiments and the simulations show that the two solutions based on OpenFlow can enhance the lightpath establishment time, reduce the control plane traffic and reduce the blocking probability. They show also that extending OpenFlow protocol by adding new

| | GMPLS with PCE | OpenFlow Message-Mapping Solution | OpenFlow Extension Solution |
|---|---|---|---|
| Establishment time | Around 350 ms | Around 130 ms | Around 60 ms |
| Control plane traffic | increases till 45 packets/sec at maximum load | increases till 23 packets/sec at maximum load | increases till 14 packets/sec at maximum load |
| Blocking probability | starts at 0.002, and reaches 0.4 | starts at 0, and reaches 0.4 | starts at 0, and reaches 0.4 |

Table 4.3: Summary of COST239 topology simulation results

messages to support optical network has a great effect on enhancing the light path establishment time and reducing the control plane traffic. This enhancement of performance makes OpenFlow the best candidate for UCP.

# CHAPTER V

# CONCLUSION

In this thesis, the main contributions are:

**Use Software Defined Network (SDN) to create a unified control plane for both optical circuit-switched and packet-switched networks**

In this thesis, a unified control plane is proposed and conducted using OpenFlow (as an SDN protocol). The control plane is conducted using two techniques and tested on the laboratory on the topology shown before in Figure 3.

**Comparison between these techniques and the standard GMPLS technique**

In this thesis, an experimental comparison conducted between the proposed solutions and GMPLS approach is presented in chapter 3. Additionally, a custom-built Java event-driven simulator is built and run to simulate the performance of our two proposed techniques and compare them with the standard GMPLS protocol on two real optical network topologies (Chapter 4).

As a conclusion from these results in the considered scenario, using SDN/OpenFlow architecture can create mutually beneficial interaction between IP and transport networks by enabling new capabilities at the packet-circuit interface. OpenFlow extension technique can significantly improve the performance of the control plane and the proposed extension is able to significantly reduce the lightpath setup time and the control plane traffic.

# CHAPTER VI

# PUBLICATIONS

## 6.1 Accepted paper at IEEE GLOBECOM 2014 conference

# 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 uscd 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).

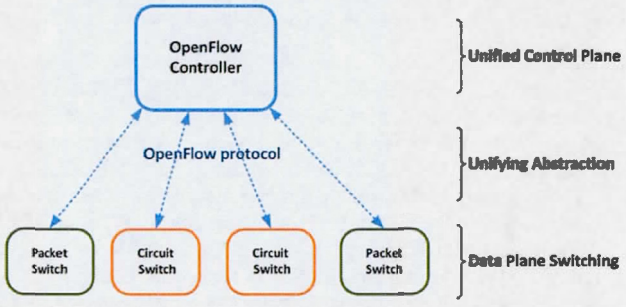



Fig. 1: Unified architecture of a converged Packet-Circuit network

### 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).

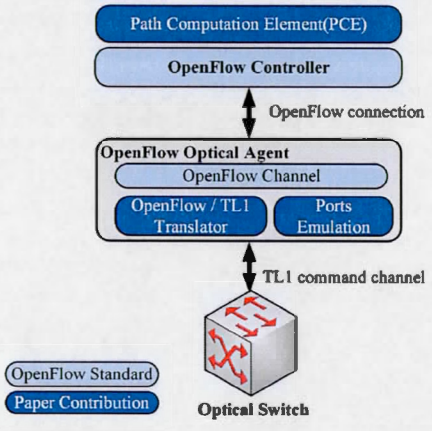



Fig. 2: OpenFlow Optical Agent intercations

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).

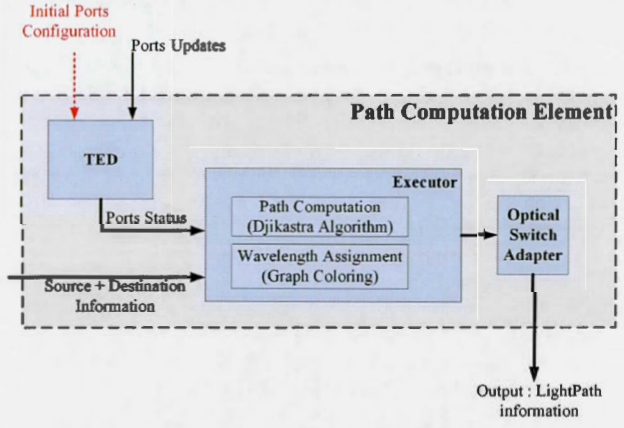



Fig. 3: Path Computation Element workflow

*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

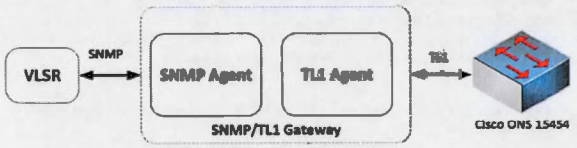



Fig. 4: SNMP/TL1 Gateway

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).

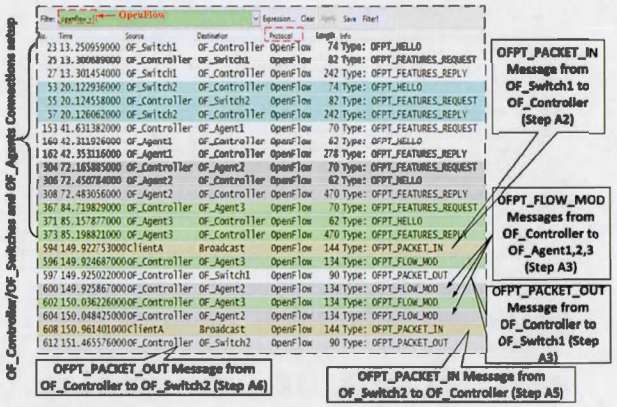
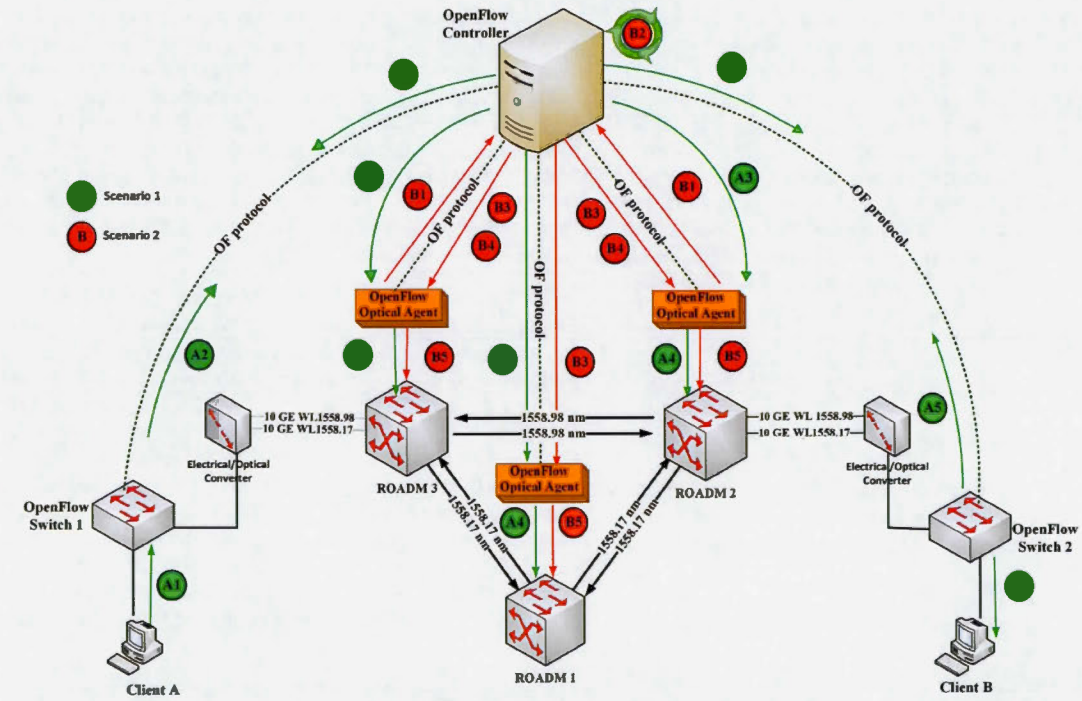



Fig. 6: OpenFlow Scenario1 : Wireshark screenshot

*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.

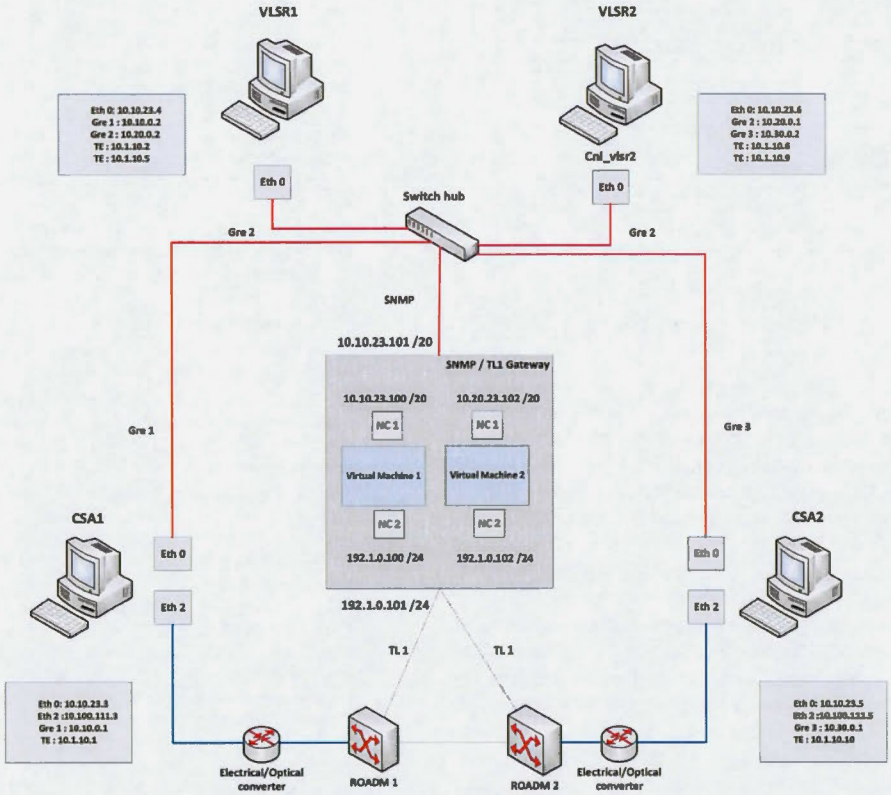



Fig. 8: DRAGON test with two ROADMs

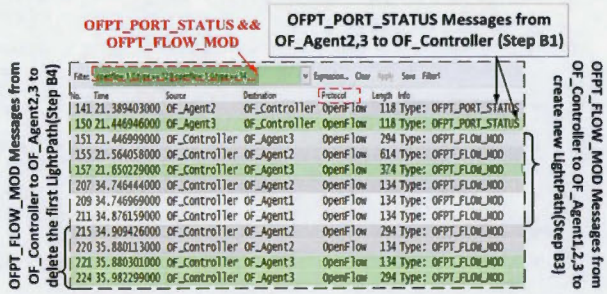



Fig. 7: OpenFlow Scenario2 : Wireshark screenshot

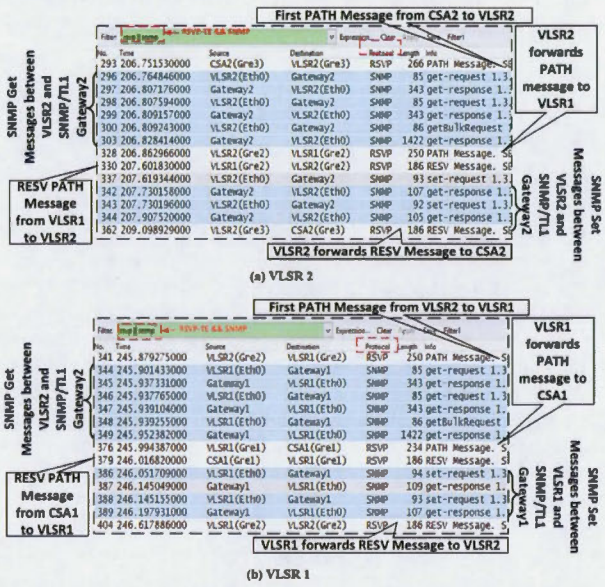



Fig. 9: GMPLS Scenario : Wireshark screenshot

### 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.

## 6.2 Submitted paper at OpticsInfoBase journal (2014)

# OpenFlow and GMPLS Unified Control Planes: Testbed Implementation and Comparative Study

Mahmoud Bahnasy, Karim Idoudi, and Halima Elbiaze

***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. 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 [1].**

**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*). The overall feasibility of these solutions is assessed, and their performance is evaluated and compared with GMPLS approach, using a custom-build simulator. Simulation results show that the *OpenFlow Extension* solution outperforms the *OpenFlow Messages Mapping* and GMPLS solutions.**

***Index Terms*—Optical Network; Software Defined Networking; OpenFlow; GMPLS; Testbed**

## I. Introduction

CURRENTLY, IP and optical layers operate separately without dynamic interaction which leads to high operational cost, low network efficiency, and long latency for end-to-end path provisioning. The main reason behind these limitations is that IP-based and optical-based networks have 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 [2]. The GMPLS protocol suite has been developed decades ago to fully operate in a distributed fashion. It is considered as the reference control plane for IP/Dense Wavelength Division Multiplexing (DWDM) multi-layer optical networks. But due to its distributed nature, the number of protocols, and the interactions between different layers, the GMPLS-based UCP is overly complex[3], [4]. Moreover, the implantation of this technology is difficult because still there are a lot of non-capable GMPLS equipments. DRAGON [5], [6] (Dynamic Resource Allocation via GMPLS Optical Networks), 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 new functionalities by writing software programs, 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 [1] has been experimented with alternative approaches. Other works [9], [10], [11] have presented similar proposition to 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 comparative study between OpenFlow and GMPLS solutions based only on simulations. 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 considering 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 named ***OpenFlow Messages Mapping***. It maps the OpenFlow standard messages into equivalent optical channel requests, without modifying the OpenFlow protocol. The second one is named ***OpenFlow Extension*** where 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, an **Open Flow-Path Computation Element (OF-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 defines 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

H. Elbiaze is with the Department of Electrical and Computer Engineering, Université du Québec à Montréal, Québec, Canada (e-mail: elbiaze.halima@uqam.ca).

[1]ROADM : Reconfigurable Optical Add-Drop Multiplexer

adaptation of DRAGON software for our ROADM (CISCO ONS 15454). Section IV presents the different experimental scenarios for each solution and the comparative results with GMPLS. In Section V, we present our custom-built Java event-driven simulator and the different algorithms and topologies used in order to compare the performances of the proposed solutions. Concluding remarks are eventually given in Section VI.

## 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[2], 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 as shown in figure 1.

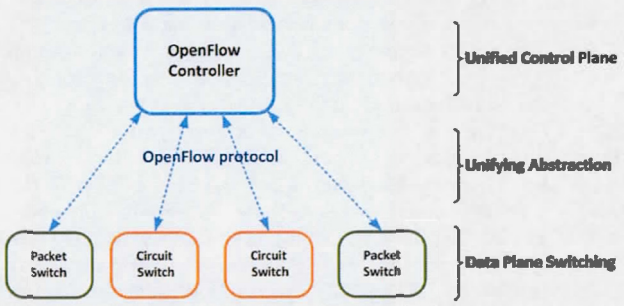



Fig. 1. Unified architecture of a converged Packet-Circuit network

[2]Match Field: a field on which packet could be matched, including packet headers, the ingress port, and the metadata value.

### 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 separately the two solutions.

*1)* ***OpenFlow Messages Mapping:*** In this solution, OpenFlow standard messages are used without any modification. OpenFlow messages are mapped into optical switch commands. Hence, 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).

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].

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

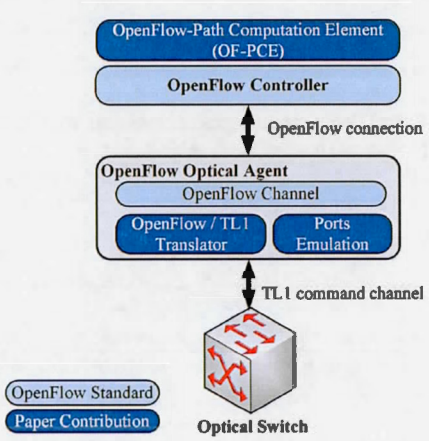



Fig. 2. OpenFlow Optical Agent intercations

### D. OpenFlow-Path Computation Element (OF-PCE)

The OF-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).

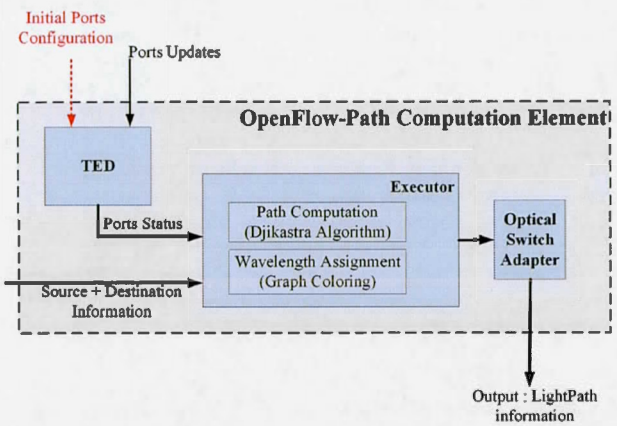



Fig. 3. OpenFlow-Path Computation Element workflow

*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.

## III. GMPLS-based Unified Control Plane

### A. Overview

It is easy to guess that GMPLS comes from MPLS. MPLS was introduced in the nineties and its best characteristics are that it could set up multiple tunnels and apply traffic engineering properties to them and also with MPLS had found a way to make two opposing Technologies coexist next to each-other and establish end-to-end paths in both packet-based and cell-based networks. At the beginning of the new millennium appears GMPLS to put together all the current networking technologies. The GMPLS is an extension of MPLS that solves some problems and adds new features. GMPLS has a set of five interfaces such as a Time-Division Multiplex capable, Lambda Switch capable or Fiber Switched capable interfaces as well as the Packet switch capable and Layer-2 Switch capable interfaces inherited from MPLS. Furthermore, of the diversity of networking technologies the GMPLS supports, it eliminates the need of an operator, the entire network can be automated and no human interference will be required in the tunneling process. Using a distributed protocol on large networks makes the path computation process very complex and resources consuming. To address this problem, Internet Engineering Task Force (IETF) has introduced a centralized Path Computation Element (PCE) entity in the GMPLS control plane.

### B. GMPLS WITH PCE SIGNALING

Because of the complexity of the GMPLS protocol, a centralized approach is presented using a PCE. The PCE is a centralized network element responsible for computing the lightpath. In this topology PCE also assign wavelength on each link for each request. The PCE is used in GMPLS-controlled Wavelength Switched Optical Network (WSON) [20], [21]. PCE uses a messaging protocol called PCEP to exchange information between GMPLS controller of each node and the PCE. PCE maintains the information of the nodes, links status and wavelength availability in a database called Traffic Engineering Database (TED) The links update is carried out by the OSPF messaging (Link State Advertisements - LSAs). This updates are sent when a new wavelength status change occurs (reserve/release). A full link status update occurs when new node joins or leaves the network. Following in detail the message sequence on GMPLS with PCE mechanism to create a lightpath:

- The source node sends a PCEP request message for submitting a path computation request.

- The PCE computes the path requested and assigns a wavelength to this path. Then the PCE sends this information to the source by using a PCEP PCRep message. Otherwise, if the PCE fails in computing a path or in assigning a wavelength on it, it replies with a PCRep message with NO-PATH reply, and the lightpath request is refused (forward-blocking).
- Upon the reception of PCRep message, the source node sends the Resource Reservation Protocol-Traffic Engineering (RSVP-TE) messages along the computed path to reserve it. The Path reservation message includes the Explicit Route and the Label set. The label set information include the wavelength assigned by the PCE.
- When a node receives RSVP-TE path reservation message, it performs the wavelength assignment if it is available. Otherwise, another wavelength contained in the Label Set is selected, according to a specific wavelength assignment strategy (e.g., first fit).
- If another request requested the same resource (link and wavelength) on a specific node and this request is accomplished before this one, this will have this node to refuse this request and reply with RSVP refuse message (backward-blocking).
- When the wavelength assigned, the destination node sends back a Resv message to effectively reserve the selected wavelength on each link of the path.
- Once the Resv message reaches the source, the lightpath is established and data could be carried through the path.

Lightpath release is performed in a similar way as the setup process (in a distributed manner through RSVP-TE signaling [22]). As the previous description the setup procedure may be blocked during path computation because of lack of resources (forward-blocking), or may be blocked due to wavelength contentions (backward-blocking). Contentions arrive when two or more RSVP-TE messages attempt to reserve the same resource (link and wavelength). This actually because the link availability database TED may be outdated when the path request reached PCE.

### C. DRAGON

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. In this paper, we use this software and adapt it to operate with our optical switch (Cisco ONS 15454).

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.

*1) 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 [3] : The Client System Agent (CSA) and Virtual Label Switch Router (VLSR).

*a) 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.

*b) 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. 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 [24], 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 [25]. The DRAGON software extends the KOM-RSVP signaling protocol with support for RSVP-TE, GMPLS, Q-Bridge, SNMP and VLAN control.

*2) Adapting VLSR for Cisco ONS 15454:* The DRAGON software suite is being developed under the GNU General public license [26]. The source code can be viewed, changed for own use. The latest version of the software suite can be downloaded at [27]. In order to install the DRAGON software, the VLSR implementation guide has been followed [28].

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

- **SNMP Agent:** Using *snmp4j* [29] open source Java library, we have developed an SNMP agent. It provides functions to receive and send SNMP PDUs (Protocol Data Unit).

[3]The informations found in this section is based on the Sara Project documentation produced by the RFC 3945 [23]

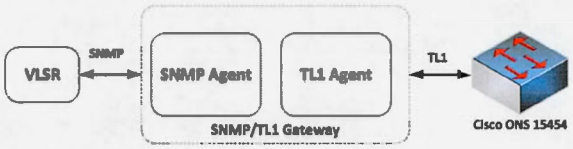



Fig. 4. SNMP/TL1 Gateway

- **TL1 Agent:** Using the *iReasing* [30] 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 A: 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 OF-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).

*3) Scenario B: 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).

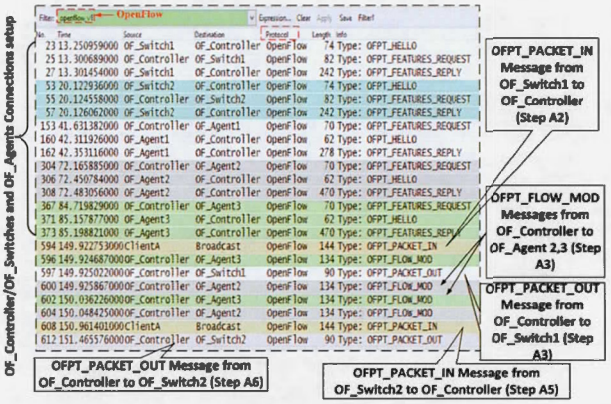



Fig. 6. OpenFlow Scenario A: Wireshark screenshot

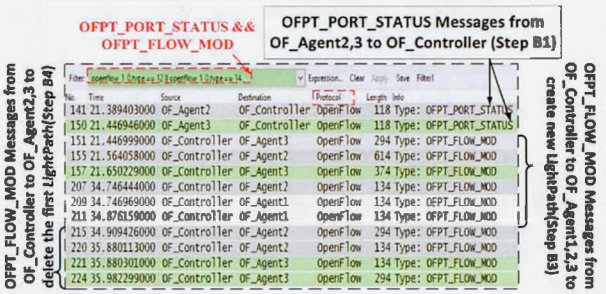
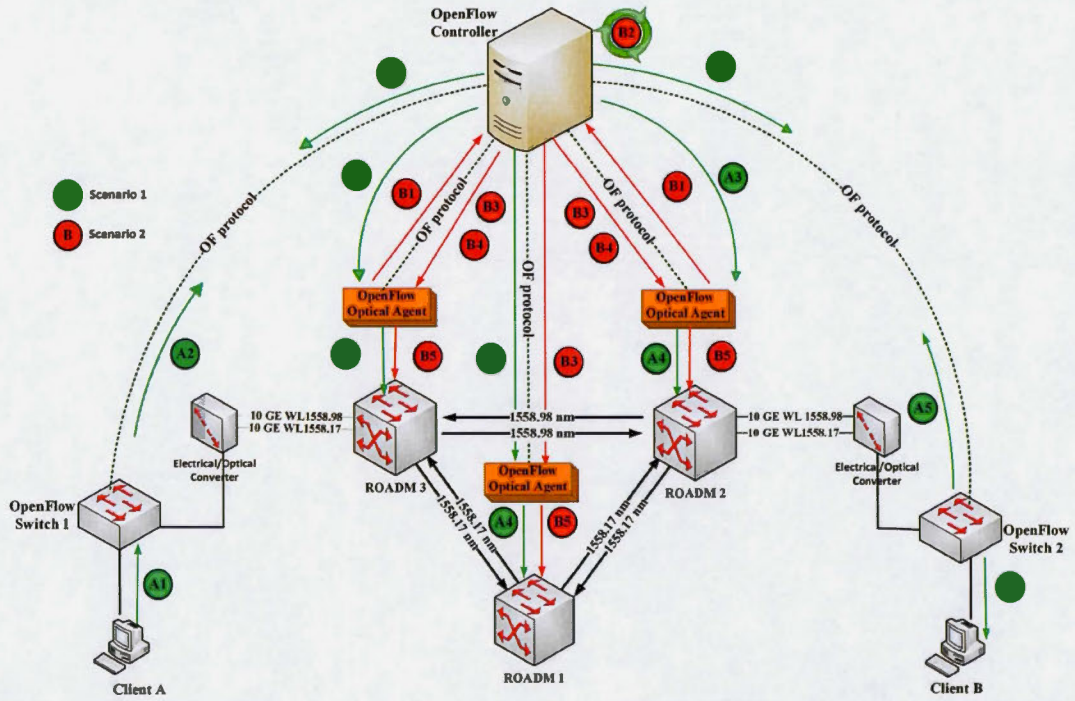



Fig. 7. OpenFlow Scenario B: Wireshark screenshot

### 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. 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 receives 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.




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

### 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. 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

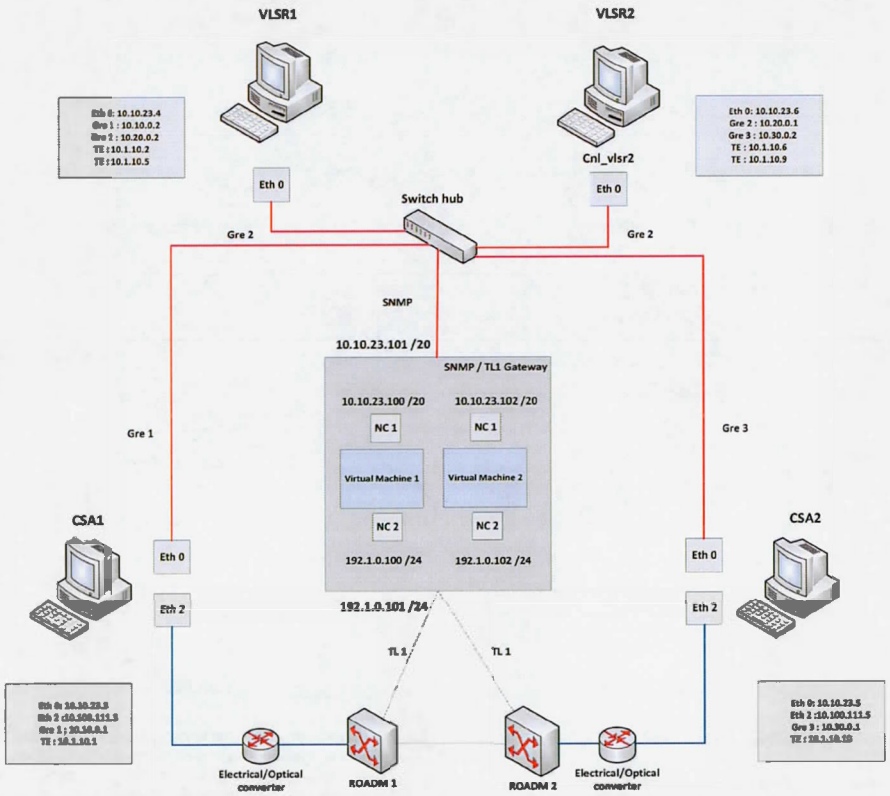



Fig. 8. DRAGON test with two ROADMs

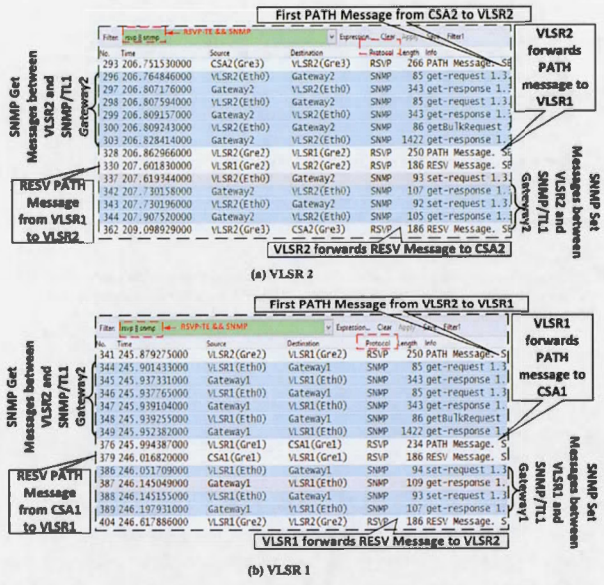



Fig. 9. GMPLS Scenario : Wireshark screenshot

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,

| | 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

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.

| | messaging protocol | | |
|---|---|---|---|
| | OFP | OSPF-TE | RSVP-TE |
| GMPLS with PCE | NO | YES | YES |
| OpenFlow Message-Mapping | YES | NO | NO |
| OpenFlow Extension | YES | NO | NO |

TABLE II
SUMMARY OF SIMULATED SOLUTIONS

## V. SIMULATION STUDY

In this chapter we present a simulation comparative study of the OpenFlow solutions (OpenFlow Messages-Mapping, OpenFlow Extension) and the GMPLS approach. To conduct the comparison, a custom-built Java event-driven simulator is written based on the mechanisms mentioned in III-B. The measurements taken from the previously conducted experiments are used in writing the simulator.

Table II shows the signaling protocol used by each solution.

The simulation is carried out on two real optical network topologies. These network topologies are the optical network topology of America National Science Foundation (NSF) and the optical network topology of the European union Ultra-High Capacity Optical Transmission Network (European Re-search Project Cost239). The next section presents the simulation environment, parameters and algorithms. Then, the results for each topology is presented in sections V-B and V-C.

### A. The Custom-built Java Event-Driven Simulator

The simulator is a custom-built Java event-driven application. It is written based on the mechanisms mentioned in chapter III-B. The internal optical switch lightpath establishment time is emulated to 60 ms for all solutions. For both topologies, the links between nodes are two directions. Each link supports 32 wavelengths. The controller and the PCE perform first-fit for assigning wavelengths. Wavelength can not be changed across the path since nodes do not support wavelength conversion. Lightpath requests are generated according to a Poisson process and uniformly distributed among all node pairs. The holding time is fixed to 180 seconds, the average inter-arrival time is varied from 0.3 s to 18 s. This varies the Erlang from 600 to 10.

The first algorithm explains how the written application simulates the OpenFlow solutions. The application uses the network topology nodes (G:Graph), the connections between them (V:Vertex), and the simulation end-time as inputs. Then, it starts by generating one event of type create-channel. After that, it reads events one at a time and handles it. Depending on the event type, each event type is treated differently as explained on the algorithm. For the create-channel event, it generates a new create-channel event based on the Poisson inter-arrival time, updates the controller's time, calculates the lightpath, finds a free channel (wavelength). Finally, it generates the create cross-connect events for each switch through the calculated path (Events to be executed by the switches). Unless there is no lightpath available, it declares this request as a blocked request. For the events of type Delete channel, it updates the controller's time. Then, it generates the delete cross-connect events for each switch through the lightpath (Events to be executed by the switches). For the event of type create cross-connect, it generates an event of type delete channel. For both events of type create/delete cross-connect, it updates nodes time (Emulating the cross-connect creation time 60 ms). Then, it updates vertex information.

```
Data: G: Graph, V: vertex, EndTime: Simulation End Time
Result: Establishment time, Blocking probability and
        control traffic
Initialization: Generate one event (using a uniformly
distributed source and destination and Poisson inter-arrival
time);
while current time < EndTime do
    read the nearest event;
    switch Event Type do
        case Create Channel
            Generate new Create Channel event based on
            Poisson inter-arrival time;
            Update the controller's time;
            Update the controller's vertex information;
            Calculate path using Dijkstra Algorithm;
            Find a free channel (wavelength) cross the
            calculated path;
            if Path calculation return false OR no channel
            available then
                Declare Request Blocked;
                Continue with the next event;
            else
                Generate create Cross-Connect events for
                each node through the calculated path (with
                the information of event time, path and
                wavelength);
            end
        end
        case Delete Channel
            Update the controller's time;
            Update the controller's vertex information;
            Generate delete Cross-Connect events for each
            node through the calculated path (with the
            information of event time, path and wavelength);
        end
        case Create Cross-Connect
            Update nodes' time (emulating the cross-connect
            creation time 60 ms);
            Update vertex information;
            Generate delete event for the created path (with
            event time = current time + hold time);
        end
        case Delete Cross-Connect
            Update nodes' time (emulating the cross-connect
            creation time 60 ms);
            Update vertex information;
        end
    endsw
end
```

**Algorithm 1:** OpenFlow Event-Driven Simulator algorithms

GMPLS simulation is shown in algorithm 2. The algorithm explains how the written application simulates the GMPLS with PCE approach. In this algorithm, the inputs and the initialization are the same as algorithm 1. By traversing all the events depending on their types, each event type is treated differently. For the create-channel events, it generates a new create-channel event based on the Poisson inter-arrival time, updates the controller time, calculates the lightpath, finds a free channel (wavelength), finally it generates the create cross-connect event for the first switch in the calculated path (Event to be executed by the switch). Unless there is no lightpath available, it declares this request as a blocked

request. For the events of types Delete channel, it updates the controller's time. Then, it generates the delete cross-connect event for the first switch in the lightpath (Event to be executed by the switch). For both events of type create/delete cross-connect, it updates node time (Emulating the cross-connect creation time 60 ms). Then, it updates vertex information. For the event of type create cross-connect, it verifies if the requested channel is available. If it is not available, it declares this request blocked (Backward Blocking) and it generates delete channel request. If it is available and this is not the last switch in the lightpath, it generates an event of type create cross-connect for the next switch in the lightpath, otherwise it generates an event of type delete channel. For both events of type LSA update (create/delete), it updates TED (controller Vertex information).

### B. National Science Foundation (NSF) topology

The first topology we ran our simulation on is the National Science Foundation (NSF) topology [31].
NSF topology consists of 14 nodes and 21 links, each link has 32 channels (wavelength) (Figure 10). The distances between each pairs are shown in the figure. Dijkstra algorithm uses these distances to calculate the shortest path.

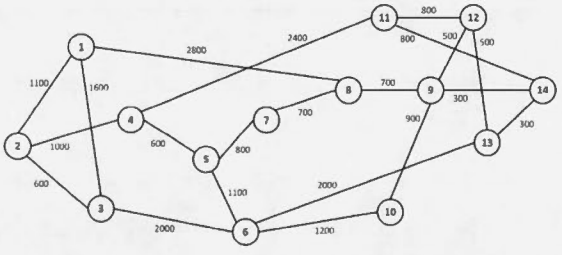



Fig. 10. NSF topology (14 nodes and 21 links)

The Simulation is run for a period of 3000 sec to ensure the stability of the network. Lightpath establishment time, control traffic gotten into and out of the controller and PCE, and the blocking probability are calculated from the simulation. The results are shown in the graphs : (i) Lightpath establishment time expressed in millisecond vs. network load (Erlang) (Figure 11); (ii) Number of control messages (Controller load) vs. network load (Erlang) (Figure 13); (iii) Lightpath blocking probability vs. network load (Erlang) (Figure 14).

Figure 11 depicts the establishment time for bidirectional lightpath. It shows that OpenFlow Extension solution experiences the lowest setup time as shown with blue line. Because OpenFlow Message-Mapping uses two *FLOWMOD* messages to establish the lightpath, it is expected that this solution experiences higher time than OpenFlow Extension solution as shown in the figure with the red line. OpenFlow solutions execute the lightpath on parallel, hence the establishment time of lightpath is around a fixed value. On the other hand, GMPLS approach executes the light path sequentially. As a result, it has the highest setup time as shown in the figure with the green line in the range 600-900 ms for bidirectional lightpath.

**Data**: G: Graph, V: vertex, EndTime: Simulation End Time
**Result**: Establishment time, Blocking probability and control traffic
Initialization: Generate one event (using a uniformly distributed source and destination and Poisson inter-arrival time);
**while** *current time* < *EndTime* **do**
  read the nearest event;
  **if** *Event Type* == *Create Channel* **then** Generate one event based on Poisson inter-arrival time ;
  **switch** *Event Type* **do**
    **case** *Create Channel*
      Update the controller's time;
      Calculate path using Dijkstra Algorithm;
      Find a free channel (wavelength) cross the calculated path;
      **if** *Path calculation return false OR no channel available* **then**
        Declare Request Blocked; Continue with the next event;
      **else**
        Generate create Cross-Connect event for the first node in the calculated path (with the information of event time, path and wavelength);
      **end**
    **end**
    **case** *Delete Channel*
      Update the controller's time;
      Generate delete Cross-Connect event for first node in the calculated path (with the information of event time, path and wavelength);
    **end**
    **case** *Create Cross-Connect*
      Update nodes time (emulating the cross-connect creation time 60 ms);
      Update switch's vertex occupation;
      **if** *current switch is the last one in the path* **then**
        Generate delete event for the created path (with event time = current time + hold time);
      **else**
        **if** *channel (wavelength) is available* **then**
          Generate create Cross-Connect event for the next node in the calculated path;
        **else**
          Declare this request blocked;
          Generate delete channel event
        **end**
      **end**
      Generate LAS update (Create) event;
    **end**
    **case** *Delete Cross-Connect*
      Update nodes time (emulating the cross-connect creation time 60 ms);
      Update switch's vertex occupation;
      **if** *current switch is not the last on the path* **then**
      Generate delete Cross-Connect event for the next node in the calculated path ;
      Generate LAS update (Delete) event;
    **end**
    **case** *LSA update (Create/Delete)*
      Update TED (controller Vertex information);
    **end**
  **endsw**
**end**

**Algorithm 2:** GMPLS/PCE Event-Driven Simulator algorithms

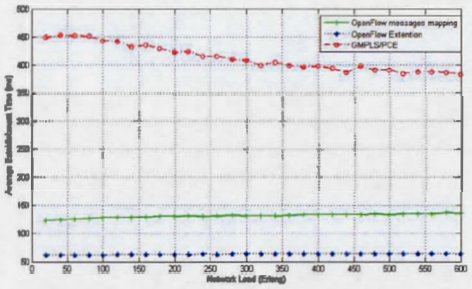



Fig. 11. Lightpath establishment time [ms] vs network load (NSF Topology)

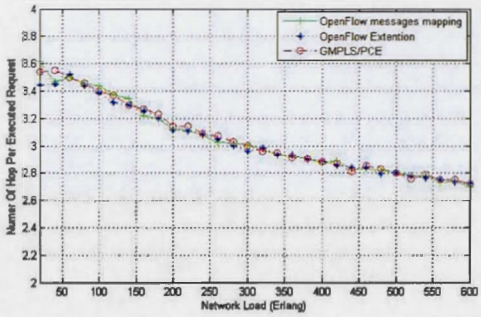



Fig. 12. Number Of Hop Per Request vs network load (NSF Topology)

GMPLS has the tendency to decrease the establishment time as the network load increases. Because at high network load, the average path length is shorter as shown in figure 12 (it decreases from 3.6 to 2.6 nodes per request). Even though the number of hop decreases too on OpenFlow-based solutions, this do not affect the lightpath setup since the request is executed on parallel.

Figure 13 depicts the control traffic for each solution. It shows that both OpenFlow solutions experience low control traffic compared to GMPLS solution as shown by blue and green lines. This difference is due to the PCEP messaging which has to be sent for each node and also because of the LSA update messages which each node has to send back to the controller in case link state changes.

Figure 14 depicts the blocking probability. This figure shows that both OpenFlow based solutions have the same blocking probability values which are expected since both techniques use the same Dijkstra algorithm and the same resource Database. On the other hand, GMPLS-based approach experiences the backward-blocking which makes this technique have higher blocking ratio with low network load as shown in the figure with green line. As we mentioned before, the backward-blocking occurs because of wavelength contentions. Contentions arrive when two or more RSVP-TE messages attempt to reserve the same resource (link and wavelength). Indeed, the link state database TED may be outdated when the path request reaches PCE causing this contention.

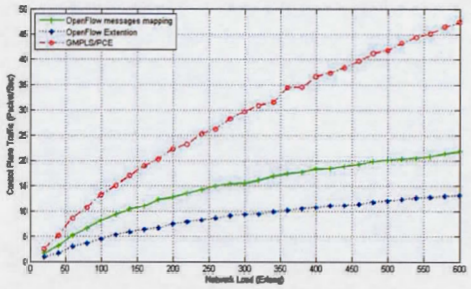



Fig. 13. Number of control messages vs network load (NSF Topology)

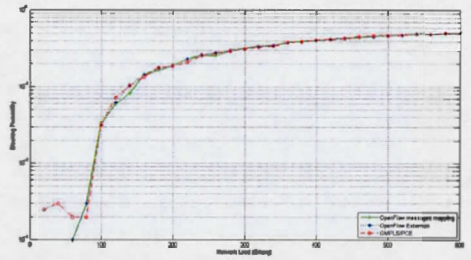


Fig. 14. Lightpath blocking probability vs network load (NSF Topology)

### C. European Optical Network Topology (COST239)

Ultra-High Capacity Optical Transmission Network (European Re-search Project Cost239) [32] is the second topology we ran our simulation on. This topology is depicted on Figure 15.

COST239 topology consists of 11 nodes and 26 links, each link has 32 channels (wavelength). The distances between each pairs are shown in the figure. Dijkstra algorithm uses these distances to calculate the shortest path.

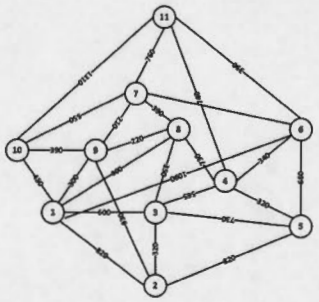


Fig. 15. COST239 Topology (11 nodes and 26 links)

The same simulation steps are followed as the NSF topology. The Simulation is run for a period of 3000 sec to ensure the stability of the network. Lightpath establishment time, control traffic gotten into and out of the controller and PCE, and the blocking probability are calculated from the simulation. The results are shown in the graphs : (i) Lightpath establishment time expressed in millisecond vs. network load (Erlang) (Figure 16); (ii) Number of control

messages (Controller load) vs. network load (Erlang) (Figure 18); (iii) Lightpath blocking probability vs. network load (Erlang) (Figure 19).

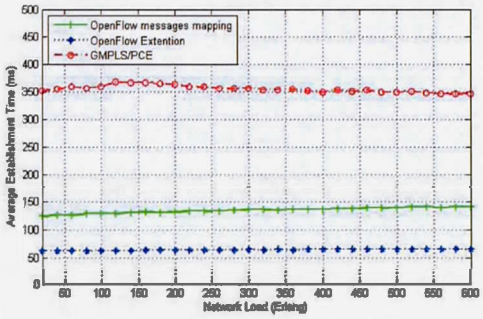



Fig. 16. Lightpath establishment time [ms] vs network load (COST239 Topology)

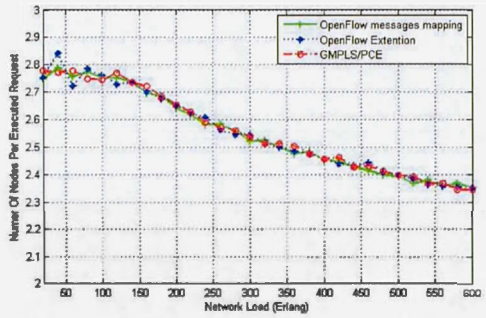



Fig. 17. Number Of Hop Per Request vs network load (COST239 Topology)

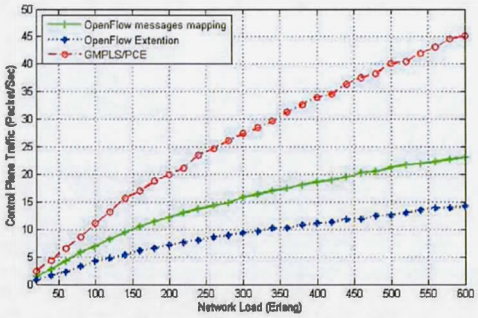



Fig. 18. Number of control messages vs network load (COST239 Topology)

The results shown in figure 16 support the same result of the NSF topology. It depicts that OpenFlow Extension solution experiences the lowest setup time as shown with blue line. It depicts also that GMPLS has the highest setup time as shown in the same figure with green line.

As the previous topology, the figure shows that GMPLS lightpath establishment time decreases as the network load increases, because at high network load the average path

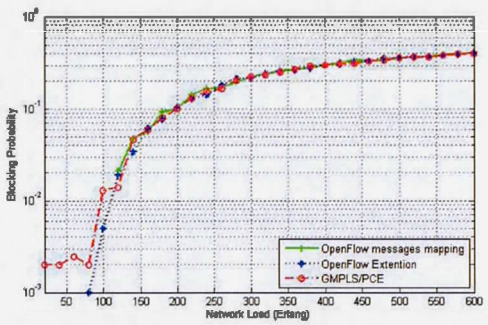



Fig. 19. Lightpath blocking probability vs network load (COST239 Topology)

length is shorter as shown in figure 17 (it decreases from 2.77 to 2.34 hop per request).

Figure 18 depicts the control messages for each solution. It confirms the result we got on the NSF topology. It shows that OpenFlow solutions experience the lowest control traffic. It depicts also that GMPLS has the highest control traffic as shown in the same figure with the green line.

Figure 19 depicts the blocking probability and it also confirms the result we got on the NSF topology. This figure shows that both OpenFlow based solutions have almost the same blocking probability values. On the other hand, GMPLS protocol experiences the backward-blocking which makes this technique have higher blocking ratio with low network load as shown in the figure with green line.

## VI. Conclusion

In this paper, we present a comparative study between two OpenFlow solutions (OpenFlow Messages Mapping, OpenFlow extension) and GMPLS approach. The overall feasibility of these solutions is experimentally assessed, and their performance is evaluated and compared with GMPLS approach, using a custom-build simulator. The simulation results show that the OpenFlow Extension solution outperforms the OpenFlow Messages Mapping and GMPLS solutions since it experience lower end-to-end lightpath setup time and lower blocking ratio and control traffic compared by GMPLS.

## References

[1] 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.

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

[3] 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.

[4] 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.

[5] "DRAGON: Dynamic Resource Allocation via GMPLS Optical Networks," http://dragon.east.isi.edu/twiki/bin/view/DRAGON/WebHome.
[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] "ONF: Open Networking Foundation," https://www.opennetworking.org/.
[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 multi-granularity 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] D. Li, G. Bernstein, G. Martinelli, and Y. Lee, "A framework for the control of wavelength switched optical networks (wsons) with impairments," 2012.
[21] V. López, B. Huiszoon, J. Fernández-Palacios, O. Gonzalez de Dios, and J. Aracil, "Path computation element in telecom networks: Recent developments and standardization activities," in *Optical Network Design and Modeling (ONDM), 2010 14th Conference on*. IEEE, 2010, pp. 1–6.
[22] A. Giorgetti, N. Sambo, I. Cerutti, N. Andriolli, and P. Castoldi, "Label preference schemes for lightpath provisioning and restoration in distributed gmpls networks," *Journal of Lightwave Technology*, vol. 27, no. 6, pp. 688–697, 2009.
[23] "RFC 3945 Generalized Multi-Protocol Label Switching (GMPLS) Architecture," http://www.ietf.org/rfc/rfc3945.txt.
[24] "GNU Zebra," http://www.gnu.org/software/zebra/.
[25] "KOMRSVP Engine," http://www.kom.tu-darmstadt.de/en/downloads/software/komrsvp- engine/ .
[26] "GNU General Public License," http://www.gnu.org/copyleft/gpl.html.
[27] "DRAGON Source Code," http://Dragon.maxgigapop.net/public/Dragon-swvlsr-daily.tar.gz.
[28] http://dragon.east.isi.edu/twiki/pub/DRAGON/VLSR/dragon-vlsr-implement-v2.1b.pdf.
[29] "SNMP4J API," http://www.snmp4j.org/.
[30] "iReasoning TL1 API," http://ireasoning.com/tl1api.shtml.
[31] N. S. Foundation. (2014) National science foundation. [Online]. Available: http://www.nsf.gov/
[32] M. O'Mahony, "Results from the cost 239 project. ultra-high capacity optical transmission networks," in *Optical Communication, 1996. ECOC '96. 22nd European Conference on*, vol. 2, Sept 1996, pp. 11–18 vol.2.

## BIBLIOGRAPHY


Banerjee, A., Drake, J., Lang, J., Turner, B., Awduche, D., Berger, L., Kompella, K., and Rekhter, Y. (2001). Generalized multiprotocol label switching: an overview of signaling enhancements and recovery techniques. *Communications Magazine, IEEE*, 39(7):144–151.

CISCO (2012a). Cisco ons 15454 dwdm reference manual, release 9.2.

CISCO (2012b). Tl1 cisco : Tl1 command guide.

Computerworld (2000). Packet-switched vs. circuit-switched.

Consortium, O. S. et al. (2009). Openflow switch specification version 1.0. 0.

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

Das, S., Parulkar, G., and McKeown, N. (2012). Why openflow/sdn can succeed where gmpls failed. In *European Conference and Exhibition on Optical Communication*, pages Tu–1. Optical Society of America.

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.

Farrel, A. and Bryskin, I. (2005). *GMPLS: architecture and applications*. Academic Press.

Foundation, N. S. (2014). National science foundation.

fp7 federica.eu (2014). Federica: Federated e-infrastructure dedicated to european researchers innovating in computing network architectures.

geni.net (2014). Geni: Global environment for network innovations.

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

Giorgetti, A., Sambo, N., Cerutti, I., Andriolli, N., and Castoldi, P. (2009). Label preference schemes for lightpath provisioning and restoration in distributed gmpls networks. *Journal of Lightwave Technology*, 27(6):688–697.

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

ITU (2000). ITU-T Recommendation G.805: Generic functional architecture of transport networks. Technical report, International Telecommunication Union.

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

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

Li, D., Bernstein, G., Martinelli, G., and Lee, Y. (2012). A framework for the control of wavelength switched optical networks (wsons) with impairments.

lightreading.com (2011). Packet-optical stays out of control.

Liu, L., Tsuritani, T., Morita, I., Guo, H., and Wu, J. (2011). 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*, pages 1–3. IEEE.

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

Liu, L., Zhang, D., Tsuritani, T., Vilalta, R., Casellas, R., Hong, L., Morita, I., Guo, H., Wu, J., Martinez, R., and Munoz, R. (2012). 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*, pages 1–3.

López, V., Huiszoon, B., Fernández-Palacios, J., Gonzalez de Dios, O., and Aracil, J. (2010). Path computation element in telecom networks: Recent developments and standardization activities. In *Optical Network Design and Modeling (ONDM), 2010 14th Conference on*, pages 1–6. IEEE.

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

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008a). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008b). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Naous, J., Erickson, D., Covington, G. A., Appenzeller, G., and McKeown, N. (2008). Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9. ACM.

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 vol.2.

open networking foundation (2013). Software-defined networking (sdn) definition.

OpenFlow (2011). Openflow.

Tintor, V. and Radunović, J. (2012). Multihop routing and wavelength assignment algorithm for optical wdm networks. *International Journal of Networks and Communications*, 2(1):1–10.

Wikimedia Foundation, I. (2003). Autonomous system (internet).

wikipedia (2014). Tier 1 and tier 2 isps.