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

ARTICLE *in* JOURNAL OF OPTICAL COMMUNICATIONS AND NETWORKING · APRIL 2015

Impact Factor: 2.06 · DOI: 10.1364/JOCN.7.000301

3 AUTHORS, INCLUDING:

Mahmoud Bahnasy

École de Technologie Supérieure

4 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Halima Elbiaze

Université du Québec à Montréal

69 PUBLICATIONS **161** CITATIONS

SEE PROFILE

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 multilayer 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, it is extremely difficult to deploy in real operational products, as there is still much non-GMPLS-capable equipment. On the other hand, OpenFlow (OF), one of the most widely used software defined networking implementations, can be used as a UCP for packet and circuit switched networks ["Packet and circuit network convergence with OpenFlow," in *OFC/NFOEC*, Mar. 2010]. In this paper, we propose and experimentally evaluate two solutions using OF 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 the GMPLS approach, using a custom-built simulator. Simulation results show that the OpenFlow Extension solution outperforms the OpenFlow Messages Mapping and GMPLS solutions.**

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

I. Introduction

Currently, IP and optical layers operate separately without dynamic interaction, which leads to high operational cost, low network efficiency, and high 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 is a key challenge for network carriers and it is important to address this issue.

Generalized multi-protocol label switching (GMPLS), a relatively mature control plane technique for optical transport networks, is proposed as a solution for UCP [1]. The GMPLS protocol suite was developed decades ago to operate completely in a distributed fashion. It is considered the reference control plane for IP/dense wavelength division multiplexing (DWDM) multi-layer optical networks. However, due to its distributed nature, the number of protocols, and the interactions among different layers, the GMPLS-based UCP is overly complex [2,3]. Moreover, the implementation of this technology is difficult as there is much noncapable GMPLS equipment.

Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) [4,5] is a software that solves this problem using Simple Network Management Protocol (SNMP) to extend non-GMPLS equipment and to make them capable of working in a GMPLS network. In this paper, we use this software and adapt it to operate with our optical switch (Cisco ROADM 15454[1]).

Furthermore, we propose using SDN [6] as a promising solution for UCP. Generally, the SDN technology separates the control and data planes so that we can introduce new functionalities by writing software programs that run 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 networks and match the carrier's preferences. One of the widely used SDN implementations is OpenFlow (OF) [7]. OF protocol is mature for L2/L3 packet switching networks, but still at a starting stage for wavelength-switched optical networks. So, it needs extensions to support the optical domain.

Some efforts have been done to present OF-based UCP to control packet and circuit switches. Most notably, experiments with PAC.C [8] have used alternative approaches. Other works [9,10] have presented a proposition similar to PAC.C by providing an experimental study or a proof-of-concept to support the use of OF as a UCP. However, [11] presents a comparative study between OF and GMPLS solutions based only on simulations. In this paper, we propose two approaches based on OF protocol to control both optical and electrical networks. Then we experimentally compare these two solutions with a real implementation of the GMPLS approach. To the best of our knowledge, this is the first study to consider both OF and GMPLS UCP solutions and to compare them using testbed experimentation. We conduct a real case study of implementing an end-to-end lightpath and lightpath restoration by establishing a dynamically configured backup lightpath.

Manuscript received June 9, 2014; revised November 27, 2014; accepted January 12, 2015; published March 26, 2015 (Doc. ID 213591).

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

http://dx.doi.org/10.1364/JOCN.7.000301

[1]ROADM: reconfigurable optical add–drop multiplexer.

1943-0620/15/040301-13$15.00/0

The first solution is named *OpenFlow Messages Mapping*. This solution maps the standard messages from OF into optical channel requests that are compatible, while not modifying the OF protocol. The second one is named *OpenFlow Extension*, in which new messages have been added to the OF protocol in order to support the circuit switching. The proposed solutions are implemented in a testbed to demonstrate their effectiveness, as well as a GMPLS-based approach. For both solutions, we implement an *OpenFlow Optical Agent* to translate the OF messages to be executed on the optical switches. Moreover, an *OpenFlow Path Computation Element (OF-PCE)* module is added to the OF controller as a network application in order to control the optical domain.

The remainder of this paper is organized as follows. Section II describes how OF can be used to define a UCP for both IP and optical networks and presents the implementation details 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 elaborate the adaptation of DRAGON software for our ROADM (Cisco ROADM 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 performance of the proposed solutions. Concluding remarks are given in Section VI.

II. OpenFlow-Based Unified Control Plane

A. Overview

In this section, we outline the main characteristics of OF. For more detail, readers are referred to [12] and [13]. OF is an open protocol developed at Stanford University for running new experimental protocols and technologies on real networks, without disrupting the traffic of the network or the availability of the network [12]. In a traditional network, the data path and the control path occur on the same device (switch, router). OF separates these two functions. OF switches perform the data plane functions and OF controllers implement the control plane intelligence and communicate with the OF switch via the OF protocol.

An OF 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 on matching packets.

OF 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

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

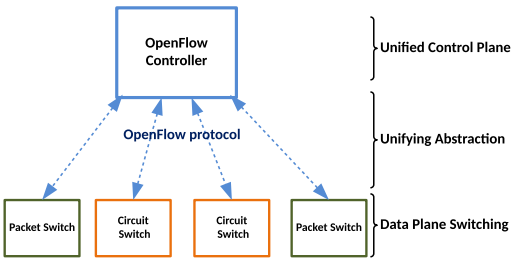


Fig. 1. Unified architecture of a converged packet–circuit network.

L1/L0 circuit flows, provides a simple flow abstraction that fits well with both types of networks. Hence, OF presents a common platform for the control of the underlying switching hardware that switches flows of different granularities, while allowing all the routing control and management to be defined by software outside the data path, in the OF controller as shown in Fig. 1.

B. OpenFlow Message Mapping and OpenFlow Extension

This paper proposes two solutions using OF protocol as a UCP for both optical and electrical domains (OpenFlow Messages Mapping and OpenFlow Extension). For both solutions, we implement an OpenFlow Optical Agent to translate the OF messages to its proper Transaction Language 1 (TL1) commands [14] to be executed on the optical switch using a telnet channel. A path computation element (PCE) module is added to the OF controller as a network application (Fig. 2). Upon request arrival, PCE calculates the corresponding lightpath and sends the cross-connection messages to the involved ROADMs. In the next sections, we describe two solutions separately.

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

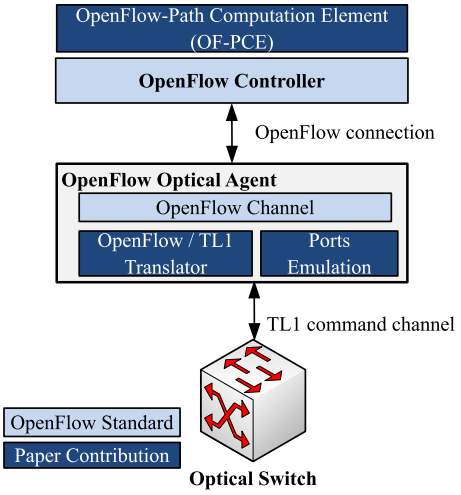


Fig. 2. OpenFlow Optical Agent interactions.

Hence, the *OFPT_FLOW_MOD* message of type *OFPFC_ADD* is mapped into the *ENT-OCHNC* TL1 command to create a lightpath channel. The *OFPT_FLOW_MOD* message of type *OFPFC_DELETE* is mapped into the *DLT-OCHNC* TL1 command to delete the lightpath channel. When the agent receives the *OFPT_FEATURES_REQUEST* message, it encapsulates the emulated port information into the *OFPT_FEATURES_REPLY* message. Finally, the agent periodically reads the ROADM events (using the *RTRV-ALM-ALL* TL1 command), and if it finds any critical alerts, it creates an *OFPT_PORT_STATUS* message and forwards this message to the controller for processing.

2) OpenFlow Extension: In this solution, OF messages are extended and new messages are added. The specification of the new messages [15] allows the controller to differentiate between the circuit-switching and the packet-switching networks. For example, the *OFPT_FEATURES_REPLY* message is extended by adding extra information about the circuit-switching ports. In order to send the information about a new cross-connect, a new match structure called *OFPT_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 OF messages into TL1 commands to be executed on optical nodes (Fig. 2).

This agent is associated with each optical node and acts as a virtual switch. It consists of i) an *OpenFlow Channel* to communicate with the OF controller, ii) an *OpenFlow / TL1 Translator* to convert OF messages into TL1 commands, and iii) a *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 the ports database and to calculate the lightpath.[3]

D. OpenFlow Path Computation Element

The OF-PCE implements an algorithm to establish lightpaths between source–destination pairs to create a fully connected logical topology [16]. A traffic engineering database (TED) is created to save the network topology information. As the OF controller has centralized management, the TED will be updated in case of lightpath creation/release and port status change. Two modules are proposed to implement the PCE: i) Executor and ii) Optical Switch Adapter (Fig. 3).

[3]Port discovery is out of scope in this paper.

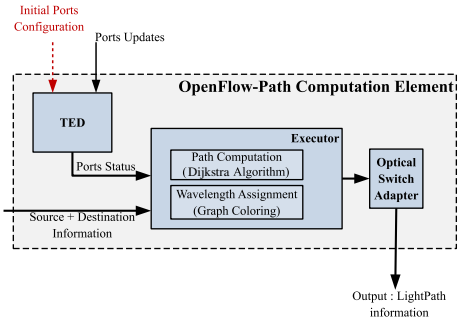


Fig. 3. OF-PCE workflow.

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 lightpaths through one fiber. These connections between source and destination nodes in DWDM networks are performed in two steps as follows:

- **Routing:** The shortest path between each pair of nodes is identified using the Dijkstra algorithm. In this case, we are interested in a network topology composed of OF 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 a 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 [17]. ROADM edges are connected to OF switches via wavelength selective switch (WSS) and channel demultiplexer (DMX) cards, whereas ROADM core interfaces are interconnected via line cards. Two fibers are used for the bidirectional connection between two ROADMs. These specifications led us to add this module.

III. GMPLS-Based Unified Control Plane

A. Overview

GMPLS evolved from **multi-protocol label switching** (MPLS), which was first introduced in the 1990s. Its best characteristics are that it can set up multiple tunnels and apply traffic engineering properties on them. MPLS found a way to make two opposing technologies coexist and establish end-to-end paths in both packet-based and cell-based networks. The GMPLS is an extension of MPLS that solves many problems and adds new features. GMPLS has a set of five interfaces, such as 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, due to the diverse networking technologies that the GMPLS

supports, it eliminates the need for an operator. In this way, the entire network can be automated and tunneling can be achieved without human interference. Using a distributed protocol on large networks makes the path computation process very complex and resource consuming. To address this problem, the Internet Engineering Task Force (IETF) has introduced a centralized PCE entity in the GMPLS control plane.

B. GMPLS With PCE Signaling

Due to 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 assigns wavelength on each link for each request. The PCE is used in a GMPLS-controlled wavelength switched optical network (WSON) [18,19]. The PCE uses a messaging protocol called PCEP to exchange information between the GMPLS controller of each node and the PCE. The PCE maintains the information of the nodes, link status, and wavelength availability in the TED.

The link update is carried out by the open shortest path first (OSPF) messaging [link state advertisements (LSAs)]. These updates are sent when a new wavelength status change occurs (reserve/release). A full link status update occurs when a new node joins or leaves the network.

The detailed message sequence on GMPLS with a PCE mechanism to create a lightpath is as follows:

- 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 PCReplay message. If the PCE fails in computing a path or in assigning a wavelength on it, it replies with a PCReplay message with NO-PATH, and the lightpath request is refused (forward blocking).
- Upon reception of the PCReplay 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 a 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 the latter request is accomplished before the former request, the node will refuse the former request and reply with RSVP refuse message (backward blocking).
- When the wavelength is assigned, the destination node sends back a Resv message to update the status of the links and wavelengths through the path to “Reserved.”
- 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 way similar to the setup process (in a distributed manner through RSVP-TE signaling [20]). As in 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), while the link availability TED is outdated when the path request reaches the PCE.

C. DRAGON

In reality, there is still much non-GMPLS-capable equipment. DRAGON software solves this problem in the Ethernet networks using SNMP to adapt this equipment to a GMPLS control plane. In this paper, we use this software and adapt it to operate with our optical switch (Cisco ROADM 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 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 equipment 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 works like a control plane within a GMPLS network. The control plane architecture consists of two basic elements[4]: the client system agent (CSA) and the virtual label switch router (VLSR).

a) *CSA:* The CSA is a software that runs on (or on behalf of) any system that 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:* GMPLS has not yet been implemented on a large 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, such as Ethernet, TDM, or optical switches. VLSR translates GMPLS commands into switch-specific commands,

[4]The information found in this section is based on the Sara Project documentation in IETF RFC 3945 [1].

such as those coming from SNMP. By the use of these commands, a VLSR can control the switch and, for example, set a switch port in a specific virtual local area network (VLAN). To communicate with other VLSRs and CSAs, a VLSR uses the routing protocol open shortest path first–traffic engineering (OSPF-TE) and path signaling protocol RSVP-TE. 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 to send information about the TE links. Information that could be sent over the control plane is information about upcoming and outgoing label switched paths (LSPs). 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 TE information 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 Universität Darmstadt's KOM-RSVP [22]. 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 ROADM 15454: The DRAGON software suite is being developed under the GNU general public license [23]. In order to install the DRAGON software, the VLSR implementation guide has been followed [24].

By default, the VLSR PC uses SNMP RFC 2674 to communicate with the switch. To manage and control the Cisco ROADM 15454, we use TL1 commands. Thus, we implement an SNMP/TL1 gateway that acts as a proxy to adapt the VLSR software with Cisco ROADM 15454 specifications (Fig. 4).

As shown in Fig. 4, the SNMP/TL1 Gateway is composed of two modules as follows:

- **SNMP Agent:** We have developed an SNMP agent using the *snmp4j* [25] open source Java library. It provides functions to receive and send SNMP protocol data units (PDUs).
- **TL1 Agent:** Using the *iReasoning* [26] TL1 API, we have developed a TL1-based management application that communicates with the Cisco ROADM 15454. Its main function is to map the SNMP messages into TL1 commands to set up configurations in the Cisco ROADM 15454.

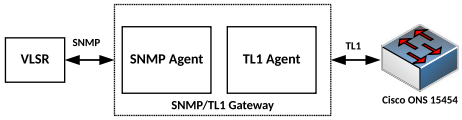


Fig. 4. SNMP/TL1 gateway.

IV. Experimental Setup

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

A. OpenFlow Experiments

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

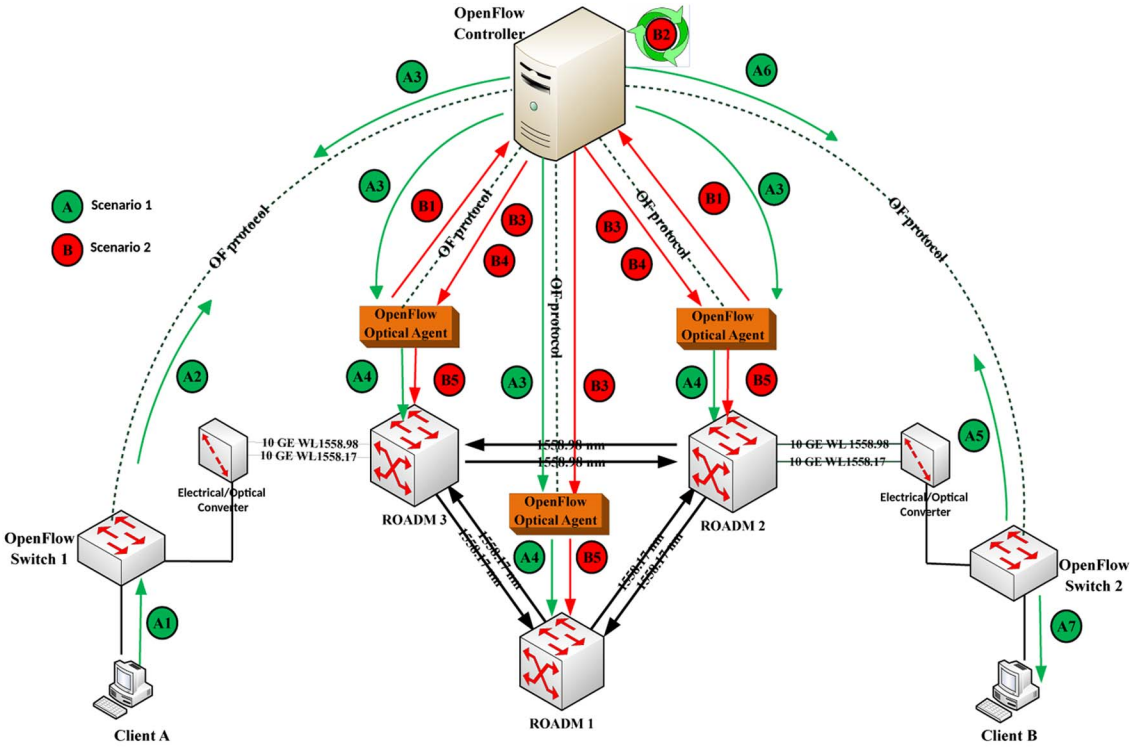
1) Testbed Setup: The architecture of our testbed is depicted by Fig. 5. It consists of two clients, A and B, which are connected directly to OF switches 1 and 2, respectively. Each switch is connected to an electrical/optical converter. These converters are connected to a DWDM optical network composed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

2) Scenario A—End-to-End Lightpath Setup: As shown in Fig. 5, a data flow sent from Client A to Client B arrives at OF switch 1. When OF switch 1 does not find any flow entry that matches with this flow, it encapsulates the first flow packet in an *OFPT_PACKET_IN* message and forwards it to the controller. The controller then uses the OF-PCE to calculate the lightpath and creates the lightpath by sending the *OFPT_FLOW_MOD* message (OpenFlow Messages Mapping solution) or the *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 (Fig. 5). The Wireshark screenshot presents the exchanged messages during this scenario (Fig. 6).

3) Scenario B—Shared Optical Restoration: This scenario demonstrates how an OF controller acts when a link failure occurs. The path deletion is performed by the controller using an *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 messages exchanged during this scenario (Fig. 7).

B. GMPLS Experiments

In order to experiment with GMPLS, we construct a transparent optical network testbed with two ROADMs (Fig. 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. Generic routing encapsulation (GRE) 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



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

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.

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.

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

A5 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.

A6 The controller sends a OFPT_PACKET_OUT message to OpenFlow switch 2 to forward this packet to client B.

A7 OpenFlow switch 2 forwards this packet to client B.

B1 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 .

B2 OpenFlow controller calculates alternative lightpaths to the existing failed lightpaths.

B3 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).

B4 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.

B5 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 OF experiments.

gateway has a connection with the switch hub to allow SNMP management by the VLSRs. It translates SNMP messages to TL1 commands 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.

Figure 9 depicts GMPLS signaling to create an LSP from CSA2 to CSA1 using Wireshark capture in VLSR2 [Fig. 9(a)] and VLSR1 [Fig. 9(b)].

CSA2 sends the *RSVP_PATH* message to VLSR2 with the destination set to 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 the *RSVP_RESV* message and sends it to VLSR1. VLSR1 forwards this message to VLSR2 because 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

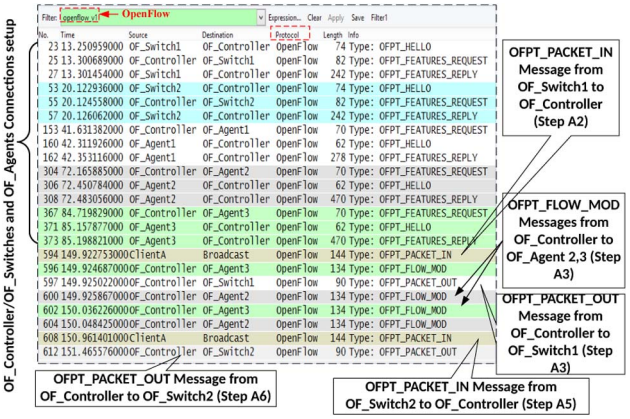


Fig. 6. OF scenario A: Wireshark screenshot.

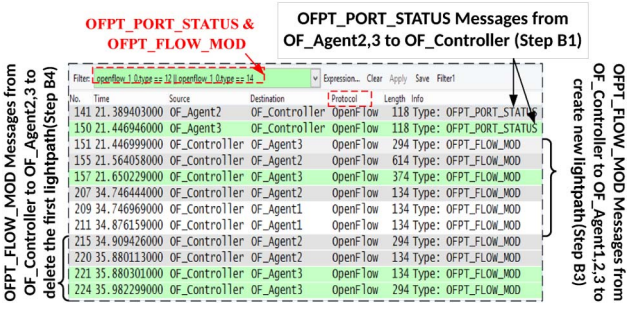


Fig. 7. OF scenario B: Wireshark screenshot.

translates the SNMP messages sent by the two VLSRs to TL1 commands in order to configure the two ROADMs.

C. Experimentation Results

Table I shows the lightpath establishment time (in milliseconds) consumed on each solution (*OpenFlow Messages Mapping* and *OpenFlow Extension*) and the GMPLS approach. In this table, Path 1 and Path 2 refer to the primary and backup lightpaths, respectively. Path 1 nodes are OF_Switch 1 → ROADM 2 → ROADM 3 → OF_Switch 2, while Path 2 nodes are OF_Switch 1 → ROADM 2 → ROADM 1 → ROADM 3 → OF_Switch 2. LSP on the table refers to the label switch path for GMPLS. LSP nodes are CSA1 → ROADM 2 → ROADM3 → CSA2. The experiment results show that the *OpenFlow Extension* solution (216 ms) outperforms the *OpenFlow Messages Mapping* (227 ms) solution. This result is expected because the *OpenFlow Extension* solution uses one message to encapsulate bidirectional lightpath information and *OpenFlow Messages Mapping* needs two messages. For the backup lightpath (Path 2), which spans on three nodes, the *OpenFlow*

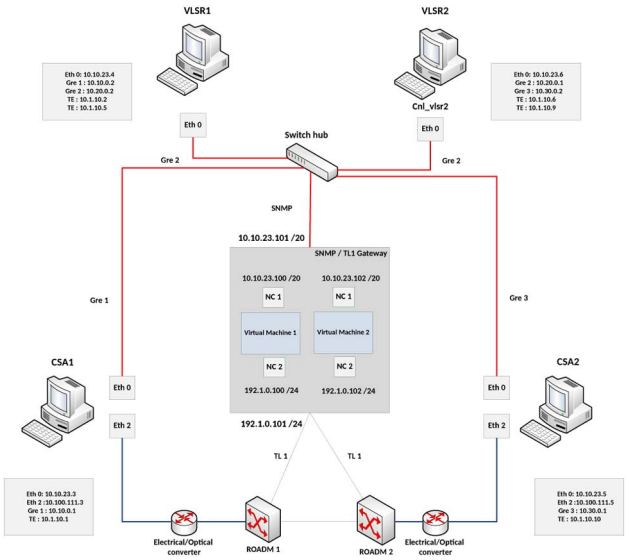


Fig. 8. DRAGON test with two ROADMs.

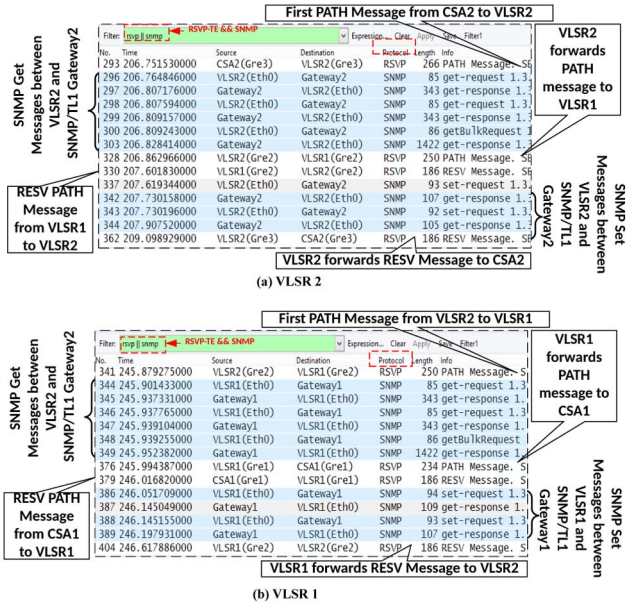


Fig. 9. GMPLS scenario: Wireshark screenshot.

TABLE I
EXPERIMENT TIMING

	OpenFlow Messages Mapping Solution				
	Switch Establishment				
	Controller	ROADM 2	ROADM 1	ROADM 3	Total (ms)
Path1	16	121	—	90	227
Path2	18	110	30	111	269
	OpenFlow Extension Solution				
	Switch Establishment				
	Controller	ROADM 2	ROADM 1	ROADM 3	Total (ms)
Path1	16	100	—	100	216
Path2	18	90	30	101	239
	GMPLS Solution				
	Switch Establishment				
	RSVP-TE	ROADM 2	ROADM 1	ROADM 3	Total (ms)
LSP	130	110	—	100	340

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 the lightpath than OF solutions. This is because the GMPLS-based control plane is complicated. 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 signalization and reservation messages must be updated and exchanged among all the intermediate VLSRs. However, the OF-based UCP provides the maximum flexibility and manageability for carriers since all the functionalities are integrated into a single OF controller. More importantly, the OF-based control plane is a natural choice for a UCP in IP/DWDM multilayer networks due to its centralized behavior (as shown in Fig. 5). Thus, the technical evolution from GMPLS to OF is a process in which the control plane evolves from a fully distributed architecture to a fully centralized one.

V. Simulation Study

In this section we present a simulation comparative study of the OF solutions (OpenFlow Messages Mapping and 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 Subsection 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. The topologies adopted are the optical network topologies of the United States National Science Foundation (NSF) and the European Union Ultrahigh Capacity Optical Transmission Network (European Research Project COST 239). The next subsection presents the simulation environment, parameters, and algorithms. Then, the results for each topology are presented in Subsections V.B and V.C.

A. 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 Subsection 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 bidirectional. Each link supports 32 wavelengths. The controller and the PCE use the first-fit approach for assigning wavelengths. Wavelengths cannot 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 s, and the average interarrival time is varied from 0.3 to 18 s. This varies the Erlang from 600 to 10.

TABLE II
Summary of Simulated Solutions

	Messaging Protocol		
	OFP	OSPF-TE	RSVP-TE
GMPLS with PCE	NO	YES	YES
OpenFlow Message Mapping	YES	NO	NO
OpenFlow Extension	YES	NO	NO

The first algorithm explains how the written application simulates the OF solutions. The application uses the network topology nodes (G:Graph), the connections among them (V:Vertex), and the simulation end time as inputs. Then, it starts by generating one event of the type create channel. After that, it reads events one at a time and handles them. Depending on the event type, each event type is treated differently, as explained in the algorithm. For the create-channel event, it generates a new create-channel event based on the Poisson interarrival time, updates the controller's time, calculates the lightpath, and 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 a blocked request. For the events of the 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 the type create cross-connect, it generates an event of the type delete channel. For both events of the type create/delete cross-connect, it updates the node time (emulating the cross-connect creation time 60 ms). Then, it updates vertex information.

Algorithm 1 OpenFlow Event-Driven Simulator Algorithms

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 interarrival 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 interarrival 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
    endsw
    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);
    endsw
    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);
    endsw
    case Delete Cross-Connect
      Update nodes' time (emulating the cross-connect
      creation time 60 ms);
      Update vertex information;
    endsw
  endsw
end
```

The GMPLS simulation is shown in Algorithm 2. The algorithm explains how the written application simulates the GMPLS with the PCE approach. In this algorithm, the inputs and the initialization are the same as in 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 interarrival time, updates the controller time, calculates the lightpath, finds a free channel (wavelength), and 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 events of the type 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 the type create/delete cross-connect, it updates the node time (emulating the cross-connect creation time of 60 ms that is calculated from the testbed experiment). Then, it updates vertex information. For the event of the 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 a delete-channel request. If it is available and this is not the last switch in the lightpath, it generates an event of the type create cross-connect for the next switch in the lightpath; otherwise it generates an event of the type delete channel. For both events of the type LSA update (create/delete), it updates the TED (controller vertex information).

Algorithm 2 GMPLS/PCE Event-Driven Simulator Algorithms

```
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 the 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
    endsw
    case Delete Channel
      Update the controller's time;
      Generate delete Cross-Connect event for the
      first node in the calculated path (with the
      information of event time, path, and wavelength);
    endsw
    case Create Cross-Connect
      Update node 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;
    endsw
    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;
      endsw
      case LSA update (Create/Delete)
        Update the TED (controller Vertex information);
      endsw
    endsw
end
```

B. National Science Foundation Topology

The first topology we ran our simulation on was the NSF topology [27].

The NSF topology consists of 14 nodes and 21 links; each link has 32 channels (wavelength) (Fig. 10). The distance between each pair is shown in the same figure. The Dijkstra algorithm uses these distances to calculate the shortest path.

The simulation was run for a period of 3000 s (50 min) to ensure the stability of the network. Lightpath establishment time, control traffic 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 versus network load (Erlang) (Fig. 11), ii) number of control messages (controller load) versus network load (Erlang) (Fig. 12), and iii) lightpath blocking probability versus network load (Erlang) (Fig. 13).

Figure 11 depicts the establishment time for a bidirectional lightpath. It shows that the OF Extension solution experiences the lowest setup time, as shown by the blue line. Because OpenFlow Messages Mapping uses two *FLOWMOD* messages to establish the lightpath, it is expected that this solution consumes more time than the OpenFlow Extension solution, as shown by the blue line in Fig. 11. OF solutions execute the lightpath on parallel; hence, the establishment time of a lightpath is around a fixed value. On the other hand, the GMPLS approach executes the lightpath sequentially. As a result, it has the highest setup time, as shown by the red line in Fig. 11, in the range of 600–900 ms for a bidirectional lightpath. 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 Fig. 14 (it decreases from 3.6 to 2.6 nodes per request). Even though the number of hops decreases also in OF-based solutions, this does not affect the lightpath setup time since the requests are executed in parallel. Figure 12 depicts the control traffic for each solution. It shows that both OF solutions experience low control traffic compared to the GMPLS solution, as shown by the blue and green lines. This difference is due to the PCEP messaging that 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 the link state changes.

Figure 13 depicts the blocking probability. This figure shows that both OF-based solutions have the same blocking probability values, which is expected since both techniques use the same Dijkstra algorithm and the same resource database. On the other hand, the GMPLS-based approach experiences backward blocking, which makes this technique have a higher blocking ratio with low

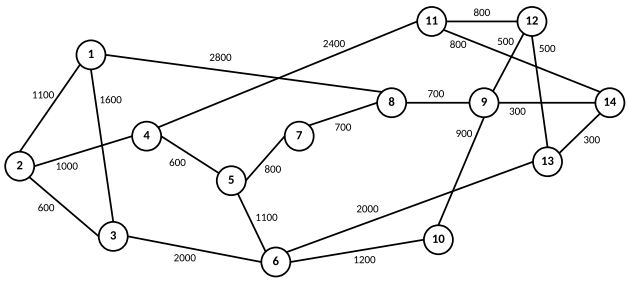


Fig. 10. NSF topology (14 nodes and 21 links).

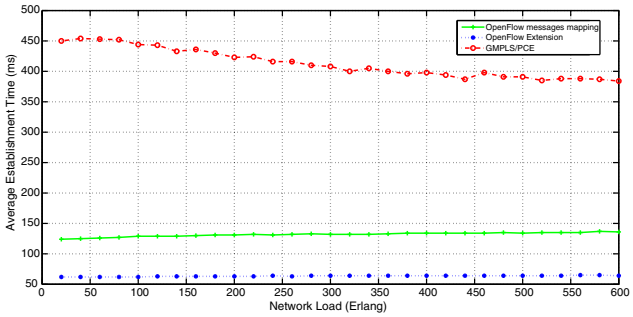


Fig. 11. Lightpath establishment time (milliseconds) versus network load (NSF topology).

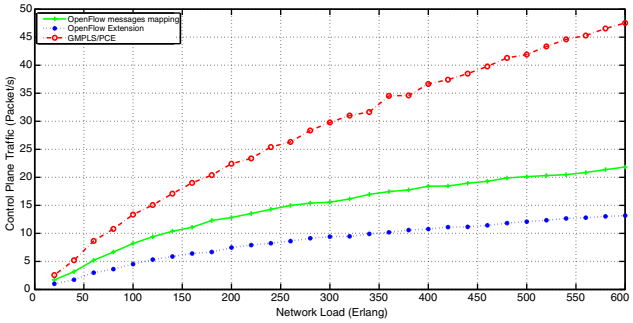


Fig. 12. Number of control messages versus network load (NSF topology).

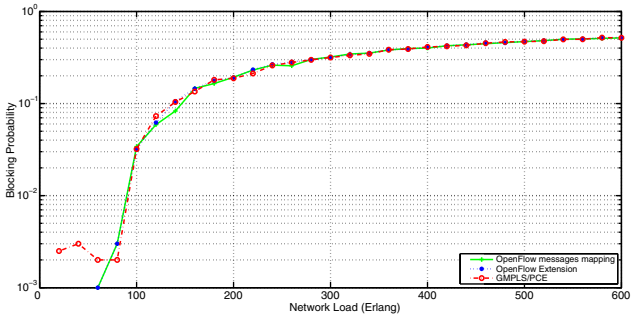


Fig. 13. Lightpath blocking probability versus network load (NSF topology).

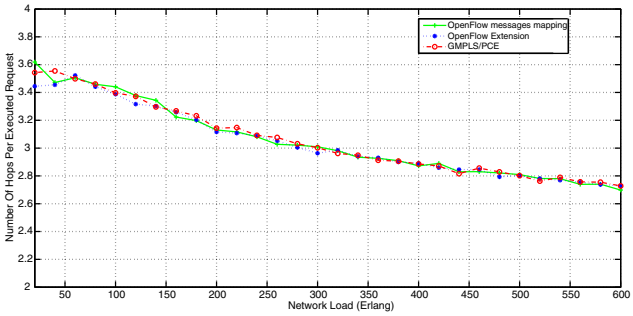


Fig. 14. Number of hops per request versus network load (NSF topology).

network load, as shown by the red line in Fig. 13. As we mentioned before, 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 the PCE, causing this contention.

C. European Optical Network Topology

The Ultrahigh Capacity Optical Transmission Network (European Research Project COST 239) [28] is the second topology on which we ran our simulation. This topology is depicted in Fig. 15.

The COST 239 topology consists of 11 nodes and 26 links, and each link has 32 channels (wavelength). The distance between each pair is also shown in Fig. 15. The Dijkstra algorithm uses these distances to calculate the shortest path.

The same simulation steps were followed as for the NSF topology. The simulation was run for a period of 3000 s

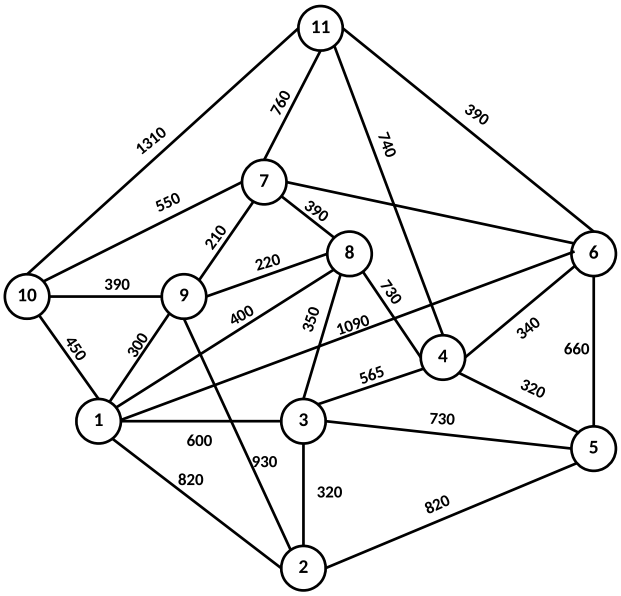


Fig. 15. COST 239 topology (11 nodes and 26 links).

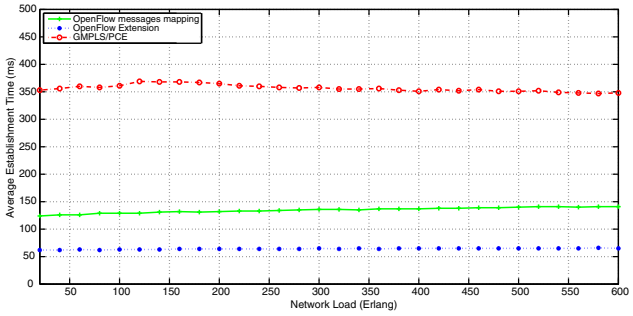


Fig. 16. Lightpath establishment time (milliseconds) versus network load (COST 239 topology).

(50 min) to ensure the stability of the network. The lightpath establishment time, control traffic into and out of the controller and PCE, and blocking probability are calculated from the simulation. The results are shown in the graphs: i) lightpath establishment time expressed in milliseconds versus network load (Erlang) (Fig. 16), ii) number of control messages (controller load) versus network load (Erlang) (Fig. 17), and iii) lightpath blocking probability versus network load (Erlang) (Fig. 18).

The results shown in Fig. 16 support the same result as the NSF topology. It depicts that the OpenFlow Extension solution experiences the lowest setup time, as shown by the blue line. It also depicts that GMPLS has the highest setup time, as shown by the red line in Fig. 16. As in the previous topology, the figure shows that GMPLS lightpath

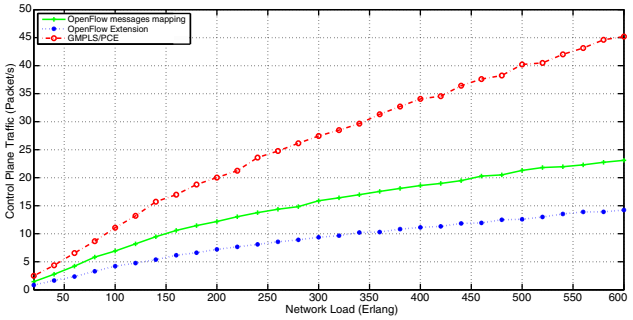


Fig. 17. Number of control messages versus network load (COST 239 topology).

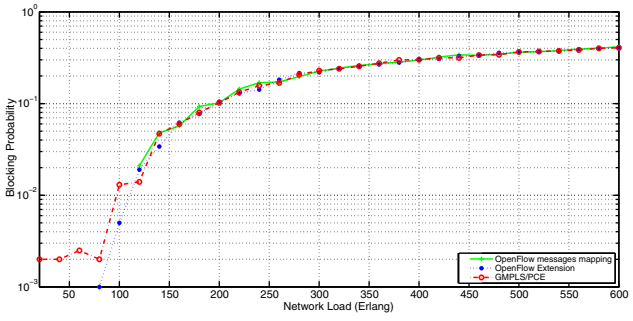


Fig. 18. Lightpath blocking probability versus network load (COST 239 topology).

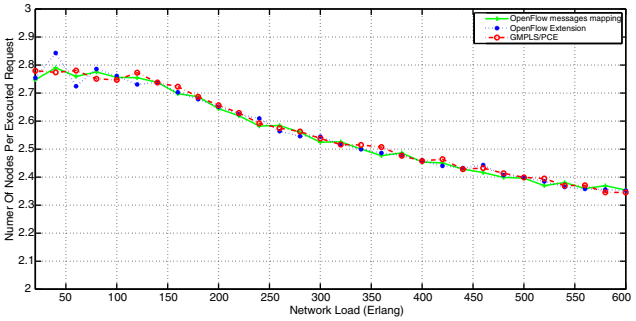


Fig. 19. Number of hops per request versus network load (COST 239 topology).

establishment time decreases as the network load increases because at high network load the average path length is shorter, as shown in Fig. 19 (it decreases from 2.77 to 2.34 hops per request). Figure 17 depicts the control messages for each solution. It confirms the result we got on the NSF topology. It shows that OF solutions experience the lowest control traffic. It also depicts that GMPLS has the highest control traffic, as shown by the red line in Fig. 17.

Figure 18 depicts the blocking probability and it also confirms the result we got on the NSF topology. This figure shows that the two OF-based solutions have almost the same blocking probability values. On the other hand, the GMPLS protocol experiences backward blocking, which makes this technique have a higher blocking ratio with low network load, as shown by the red line in Fig. 18.

VI. Conclusion

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

References

[1] E. Mannie, "Generalized multi-protocol label switching (GMPLs) architecture," IETF RFC 3945, Oct. 2004 [Online]. Available: http://www.ietf.org/rfc/rfc3945.txt.

[2] L. Liu, T. Tsuritani, and I. Morita, "Experimental demonstration of OpenFlow/GMPLs interworking control plane for IP/DWDM multi-layer optical networks," in *IEEE 14th Int. Conf. on Transparent Optical Networks (ICTON)*, 2012, pp. 1–4.

[3] 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 Conf. and Expo. and the Nat. Fiber Optic Engineers Conf. (OFC/NFOEC)*, 2013, pp. 1–3.

[4] "DRAGON: Dynamic Resource Allocation via GMPLS Optical Networks" [Online]. Available: http://Dragon.maxgigapop.net.

[5] T. Lehman, J. Sobieski, and B. Jabbari, "DRAGON: A framework for service provisioning in heterogeneous grid networks," *IEEE Commun. Mag.*, vol. 44, no. 3, pp. 84–90, Mar. 2006.

[6] ONF: Open Networking Foundation [Online]. Available: https://www.opennetworking.org/.

[7] OpenFlow [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow.

[8] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with OpenFlow," in *Optical Fiber Communication Conf. and the Nat. Fiber Optic Engineers Conf. (OFC/NFOEC)*, Mar. 2010.

[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 *IEEE 37th European Conf. and Exhibition on Optical Communication (ECOC)*, 2011, pp. 1–3.

[10] 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 Conf. and Expo. and the Nat. Fiber Optic Engineers Conf. (OFC/NFOEC)*, Mar. 2012.

[11] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow and PCE architectures in wavelength switched optical networks," in *16th IEEE Int. Conf. on Optical Network Design and Modeling (ONDM)*, 2012.

[12] 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 Comput. Commun. Rev.* vol. 38, no. 2, pp. 69–74, 2008.

[13] OpenFlow Switch Consortium, "OpenFlow switch specification version 1.0.0," 2009.

[14] Cisco Systems, Inc., "TL1 command reference for the Cisco ONS 15808 DWDM system," 2003.

[15] S. Das, "Extensions to the OpenFlow protocol in support of circuit switching: Addendum to OpenFlow Protocol Specification (v1.0) – Circuit Switch Addendum v0.3," 2010.

[16] V. Tintor and J. Radunović, "Multihop routing and wavelength assignment algorithm for optical WDM networks," *Int. J. Netw. Commun.*, vol. 2, no. 1, pp. 1–10, 2012.

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

[18] D. Li, G. Bernstein, G. Martinelli, and Y. Lee, "A framework for the control of wavelength switched optical networks (WSONs) with impairments," IETF RFC 6566, Mar. 2012.

[19] V. López, B. Huiszoon, J. Fernández-Palacios, O. G. de Dios, and J. Aracil, "Path computation element in telecom networks: Recent developments and standardization activities," in *14th IEEE Conf. on Optical Network Design and Modeling (ONDM)*, 2010, pp. 1–6.

[20] A. Giorgetti, N. Sambo, I. Cerutti, N. Andriolli, and P. Castoldi, "Label preference schemes for lightpath provisioning and restoration in distributed GMPLs networks," *J. Lightwave Technol.*, vol. 27, no. 6, pp. 688–697, 2009.

[21] GNU Zebra [Online]. Available: http://www.gnu.org/software/zebra/.

[22] KOM RSVP Engine [Online]. Available: http://www.kom.tu-darmstadt.de/.

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

[24] https://spaces.internet2.edu/download/attachments/57278477/dragon-vlsr-implement-v2.1b.pdf?version=1&modificationDate=1212654353149.

[25] SNMP4J API [Online]. Available: http://www.snmp4j.org/.

[26] iReasoning TL1 API [Online]. Available: http://ireasoning.com/tl1api.shtml.

[27] National Science Foundation [Online]. Available: http://www.nsf.gov.

[28] M. O'Mahony, "Results from the COST 239 project. Ultra-high capacity optical transmission networks," in *22nd European Conf. on Optical Communication (ECOC)*, vol. 2, Sept. 1996, pp. 11–18.