ISCTE IUL
Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias da Informação

GMPLS-controlled OBS Network Simulator:
Implementation of the signaling protocol

João Pedro Nunes Caldeira Baião

Dissertação submetida como requisito parcial para obtenção do grau de Mestre em Engenharia de Telecomunicações e Informática

Orientador:

PhD Luís Gonçalo Lecoq Vences e Costa Cancela, Professor Auxiliar,

ISCTE-IUL

Co-Orientadores:

PhD Davide Careglio, Professor Auxiliar

Universitat Politècnica de Catalunya (UPC), Barcelona

Pedro Pedroso, PhD Candidate

Universitat Politècnica de Catalunya (UPC), Barcelona

Outubro, 2010

The present document was accomplished by the Master's student João Pedro Nunes Caldeira Baião from the Instituto Superior de Ciências do Trabalho e da Empresa (ISCTE) in collaboration with the Telecommunications Department of Universitat Politècnica de Catalunya (UPC).

The academic services of ISCTE and UPC

ACKNOWLEDGEMENTS

First of all I would like to say that this was a wonderful academic year in many aspects. Writing this thesis means that I'm finishing an important chapter of my life as a university student and at the same time opening a completely new one, which is my professional life.

From colleagues to teachers, everyone was important to me to achieve the moment that I'm living right now.

I would like to show my gratitude to Dr. Luis Cancela (ISCTE-IUL, Lisbon), Pedro Pedroso and Pedro's advisers Dr. Davide Careglio and Dr. Josep Solé-Pareta (UPC, Barcelona) for their patience and support that was given to me during my stay at Barcelona.

I also want to leave an appreciate word to everybody I have met at the Àgora BCN University Residence Barcelona, they might not have had influence on this dissertation but definitely have helped me to keep up my state of mind and didn't let me down in those 5 months that I had to live alone and far away from my family.

Last but not the least, very special thanks for the scholarship awarded to me by the Institute of Telecommunications (IT, Lisbon) and of course to my family who always have supported me and had to make a great financial effort to let me stay in Barcelona. Without that, this dissertation wouldn't have been possible.

To those that I have mentioned, thank you very much for all of your support!

Acronyms

ASON - Automatic Switched Optical Network

ACK – Acknowledgment

ATM – Asynchronous Transfer Mode

BCP – Burst Control Packet

CR-LDP - Constraint-based Label Distribution Protocol

CU – Control Unit

DB - Database

DWDM – Dense Wavelength-Division Multiplexing

FT – Forwarding Table

FDL – Fiber Delay Line

GMPLS – Generalized Multiprotocol Label Switching

GC – GMPLS Controller

IP – Internet Protocol

IS-IS-TE - Intermediate System to Intermediate System - Traffic Engineering

JAVOBS – Java OBS Networks

JA-GOBS – Java – GMPLS/OBS Networks

JET – Just Enough Time

JIT – Just In Time

LMP – Link Management Protocol

LSP – Label Switched Path

MILP - Mixed-integer Linear Programming

MPLS – Multiprotocol Label Switching

OBS – Optical Burst Switching

OCS	– Optical Circuit Switching
OPS	– Optical Packet Switching
O-PNNI	– Optical Private Network to Network Interface
O/E/O	– Optical / Electronic / Optical convertion
OSPF-TE	– Open Shortest Path First - Traffic Engineering
OXC	– Optical Cross-Connect
PATH	– Signaling Path Message
RSVP-TE	- Reservation Protocol-Traffic Engineering
RESV	– Signaling Reservation Path Message
SC	- SwitchController
SDH/SONET	– Synchronous Digital Hierarchy /Synchronous Optical Network
TAW	- Tell and Wait
TE	– Traffic Engineering
UNI	– User-To-Network Interface
VC	– VirtualAggregatorController
WDM	– Wavelength-Division Multiplexing

Abstract

The Optical Burst Switching (OBS) paradigm is regaining greater attention by the professionals and researchers of the optical networking field, as it offers a number of advantages when compared with other optical switching paradigms. This type of technology was developed with the objective to carry information all-optically without using any kind of buffering device. However, due to its one-way signaling process, the presence of a control plane is extremely useful to manage complementary signaling and routing features, providing flexibility, reliability and taking more benefits of the OBS networks.

The goal of this project is to extend Generalized Multiprotocol Label Switching (GMPLS) control plane architecture to properly handle OBS networks. In spite of GMPLS is not prepared to lead with these type of networks, this flexible architecture has been seen as a potential candidate to be used as the control plane of other kinds of optical networks (e.g., IP, Ethernet, Optical Circuit networks) and therefore to manage control OBS networks.

In this project, the existent event-driven JAVA simulator for OBS networks – JAVOBS – is extended to simulate a possible interoperability model between GMPLS and OBS technologies. The first objective is to implement a new control layer (GMPLS) separated from the data layer of the OBS network. The second and main objective fits on the basic signaling procedures implementation of the GMPLS Reservation Protocol-Traffic Engineering (RSVP-TE) protocol, in order to analyze the performance of the OBS network's behavior when it is controlled by such interoperable control plane (GMPLS/OBS).

Keywords: OBS, GMPLS, Control Plane, LSP, Simulator

Resumo

O paradigma de Redes de comutação de Rajadas (*Optical Burst Switching* - OBS) tem vindo a ganhar maior atenção por parte dos profissionais e investigadores do campo das redes ópticas, uma vez que oferece um número de vantagens quando comparado com outros tipos de paradigmas de comutação óptica. Este tipo de tecnologia foi desenvolvido com o objectivo de transportar a informação apenas no domínio óptico, sem o uso de qualquer tipo de dispositivo de memória. No entanto, devido ao seu processo de sinalização *one-way*, a presença de um plano de controlo é extremamente útil para gerir sinalização complementar e características de encaminhamento, proporcionando flexibilidade, fiabilidade e tirando maior proveito das redes OBS.

O objectivo deste projecto consiste na extensão da arquitectura do plano de controlo *Generalized Multiprotocol Label Switching* (GMPLS) para suportar correctamente redes OBS. Apesar de o GMPLS não estar preparado para lidar com este tipo de redes, esta arquitectura flexível tem sido vista como uma potencial candidata a ser usada como plano de controlo para outros tipos de redes ópticas (e.g. IP, *Ethernet*, Redes de Comutação de Circuitos) e por conseguinte, para a gestão das redes OBS.

Neste projecto, o simulador *JAVA* baseado em eventos para redes OBS- JAVOBS – é estendido para simular um possível modelo de interoperabilidade entre as tecnologias GMPLS e OBS. O primeiro objectivo consiste na implementação da nova camada de controlo (GMPLS) separada da camada de dados da rede OBS. Quanto ao segundo e principal objectivo, este enquadra-se na implementação dos procedimentos básicos de sinalização do protocolo de reserva RSVP-TE do GMPLS, tendo em vista a análise da *performance* do comportamento da rede OBS quando esta é controlada por um plano de controlo interoperável (GMPLS/OBS).

Palavras-Chave: OBS, GMPLS, *Control Plane*, LSP, *Simulator*

List of Figures

Contents

Chapter I

Introduction

1.1 Background and Motivation

Nowadays, one of the major concerns in the Internet-based information is the demand for even more bandwidth. Optical communication technology has the potential for meeting the upcoming needs of getting information at a much faster yet more consistent rates (i.e., less errors and losses in the information transmission), because of its potential capabilities(e.g. vast bandwidth provided). The objective is to turn the promise of optical networking into reality and meet the demands of the Internet communication for the coming years.

Current optical networks, though offering high capacity, are limited due to inherent inflexibility of manually provisioned large scale networks. Long provisioning times, inefficient resource utilization or complex network managements are some of the major drawbacks that can be enumerated, as it says in [1].

The *Automatic Switching Optical Network* (ASON) framework appears to be a potential solution to cover this situation. Opposite to the drawbacks referred above, this architecture provides fast provisioning, scalability, and easier network management, amongst others. However, the ASON control plane architecture is specified in a neutral way protocol, which enables a generic approach for defining and validating potential solutions. The ITU-T idea is to integrate the *Generalized Multiprotocol Label Switching* (GMPLS) protocols or other valid applicants in the ASON framework [2, 3].

Two different control plane model approaches were discussed: the *Optical Private Network to Network Interface* (O-PNNI) [4] and the GMPLS [5]. What essentially differs the GMPLS architecture from O-PNNI is the fact that the first one is able to support the separation of the control plane from the data plane, which is one of the most important control plane requirements demands by the ASON framework. Once the control plane is separated, it can be used not only for one but for several types of networks.

On one side, the creation of a control plane separated from the data plane will considerably impact the optical networks operations and the management systems. Such systems will be relieved from functions, such as the connections setting up/teardown, and route selections. On the other side, it will provide more flexibility in handling higher capacity networks and increasing scalability which contributes to support switching processes on a global scale [1, 2]. Structurally, the control plane functions include signaling, routing, path selection and link management, while all physical resources management and the transmission of optical data traffic through the optical network are done over the data plane.

The GMPLS is the architecture adopted for the control plane implementation that consists of a *Multiprotocol Label Switching* (MPLS) [6] generalization that extends the data routing based on labels for non-based packets switching devices. Inclusively, it also adds other types of switching capabilities [6]: *cell* switching (e.g. Asynchronous Transfer Mode, ATM), *timeslot* switching (i.e., Time Division Multiplexing – TDM), wavelength switching (Wavelength Division Multiplexing – WDM) and spatial switching (i.e. between fibers).

Due to the GMPLS-based control plane dynamical capacity to deal with these several types of traffic switching, makes us to think, why not to use the GMPLS control plane for Optical Burst Switching (OBS) networks? The idea of this project is to have a control plane that can handles with all kind of network types, such as Optical Circuit Switching (OCS) networks, Optical Packet Switching (OPS) networks and OBS networks.

1.2. Challenges and Objectives

In this work we face a challenge that aims for the possible integration of the GMPLS in the OBS networks control plane [7].

The objectives are:

1) The implementation of a GMPLS control layer in OBS-based JAVA simulator [8]. The current version of the simulator is based on a single OBS network layer. Thus, the idea is

to separate the basic features of the GMPLS control plane from the OBS network (data and control planes) and establishing a correct interaction between them.

2) Implementation of basic signaling procedures of the GMPLS signaling protocol, namely *Resource Reservation Protocol – Traffic Engineering* (RSVP-TE). Demonstration of the end-to-end exchange process of the signaling messages and the successful LSP set-up to forward traffic over OBS network.

The master thesis is organized as follows.

Chapter II – Takes a look on the state of the art. First, it is given an overview over the optical switching technologies with a special focus on the OBS technology. Second, the GMPLS architecture is briefly described, with emphasis on the signaling protocol. This chapter ends with the GMPLS/OBS architecture that is considered along this project, pointing out the ideas and techniques defined in order to solve the problems faced in the OBS networks.

Chapter III – Explains in detail the functionality of a Java simulator implemented for burst transmissions in OBS networks. In a first phase, it is explained how the simulator works without the presence of the GMPLS-based control plane. Then, basic signaling features of the GMPLS control plane architecture are implemented (RSVP-TE protocol), where we will be able to observe the exchange of the signaling messages, through a simulation demo.

Chapter IV – The fourth chapter exposes a simulation demo of the RSVP-TE protocol, showing all the signaling messages that are exchanged for a successfully burst-LSP establishment.

Chapter V – Refers to the conclusions taken from the entire project, which demonstrate the correct implementation and interaction between the two separated networks and the GMPLS signaling protocol RSVP-TE performance on the GMPLS control network. This chapter ends with a future work suggestion to be accomplished as part of this vast field of Optical Communications.

Chapter II

State of the Art

2.1 Introduction

Many studies have been carried out aiming to reduce the number of O/E/O signal conversions. This results in all-optical switching networks where the data transmission is done only in the optical domain.

This chapter explains the characteristics of the OBS technology [9, 10], but it is also given a brief explanation of the OCS [9, 11] and the OPS [9, 12] technologies, later on, the GMPLS control plane is summarized, focusing on its signaling protocol (RSVP-TE). This chapter ends with the approach on a possible interoperability between the GMPLS and the OBS technologies.

2.1.1 Optical Circuit Switching

Today's current all-optical networks are based on circuit switched. Optical packet switched networks are still being studied, or in other words they are on "laboratory level" yet. It seems that in the optical world, circuit switching is considerably more appropriate alternative than packet switching due to its characteristics in comparison to this last switching technology, as it will be referred in this subsection. This type of switching technology, as it is shown on *fig. 1*, has three distinct phases:

- Circuit set-up
- Data transmission
- Circuit tear-down

There are two main features of optical circuit-switching that differentiates it from the OBS and OPS technologies. The first one is its *two-way* reservation process in the first phase (also known as the *Tell-and-Wait, TAW* technique), where a source sends a request for setting up a circuit and needs to wait for an acknowledgment (ack) from the corresponding destination. The second one is the fact that circuits tend to be fairly static and provide a fixed amount of bandwidth.

One of the major advantages of optical circuit switching is that is a fundamentally transparent service with no storage requirement. Once the connection is established, constant data rate is provided and sent to the connected stations. An important consequence of transparency is that, no buffer is required to accommodate the optical information that travels through the Label Switched Path (LSP), since all the resources until the destination point are reserved. An LSP in the optical networks scope refers to a connection establishment from a source to a destination node, composed by a group of links which form the lightpath [6].

On what concerns the disadvantages, the OCS has an undesired delay provoked by the time that data needs to wait for the connection establishment. Means that, most of the times the transmission time tends to be smaller compared to the roundtrip propagation delay required by the signaling protocol, leading to inefficient use of the LSP and overhead. For example, consider a case in which a connection is established with all the necessary available resources dedicated to the entire duration of a connection, where no data is actually being transmitted. In case we are in presence of voice communications, since the idle times are minimum, high utilization is expected, but for data communications, since the capacity may be idle during most of the time of the connection (e.g., when a user is only reading a downloaded web site), this type of circuit switching can result in inefficient utilization.

Furthermore, since the circuits-switching uses the reservation TAW technique [9], in case the connection goes down, the entire information will be lost without any possibility to recover.

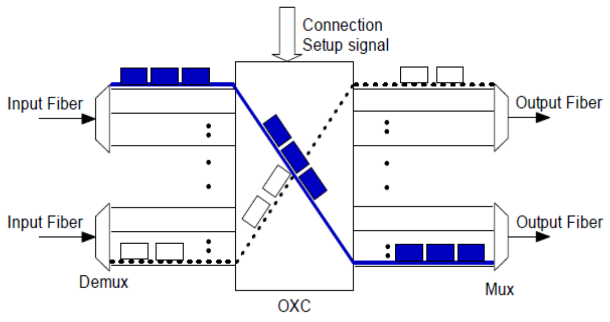


Figure 1. OCS architecture

2.1.2 Optical Packet Switching

The OPS technology will probably be used in the future as the preferential switching technology for optical networks. This kind of network rather than the OCS, attempts to use resources only when data is being transferred, thereby providing a high degree of the bandwidth utilization.

However, there are some factors that put the OPS technology "behind" the OCS: the difficult implementation, the higher costs due to the presence of several O/E/O converters and the optical buffers implementation.

As screened on *fig. 2,* this architecture follows two phases on its transmission mode:

- A header is added to the payload forming what we call a packet, before any transmission. Uses *one-way* reservation process (also known as the *Tell-and-Go, TAG* technique [9]).

- Each packet is transmitted without the need of any ack. Every time a packet reaches a node, the header must be extracted by using a converter O/E/O to make the resources reservation, while the payload is stored in a *fiber-optic delay line* (FDL) until the header's extraction process is completed.

The two major advantages in comparison to OCS technology is the reduced delay on the pre-transmission of each packet and the OPS robustness. The OPS has the possibility to route the packets to another path in case the resource reservation process fails in a switching point, avoiding the lost of the entire information.

However, every time a packet reaches a switching node and the header is extracted, it increases the packet's delay. Besides that, the conversion O/E/O is not simple to implement and the use of a single converter for each single packet takes costs. Furthermore, another drawback concerns on the difficult allocation of optical memory for the packets switching. Even the FDL's are not enough due to their limited storage capacity, where in a situation which there is a continuously coming of data packets, there is a chance to lose some of them. On one hand, the OCS increases the data delay but it gives near 100% of feasibility that after receiving the *ack* the data will not be lost. On the other hand, the OPS does not have much delay but it is susceptible to packets lost.

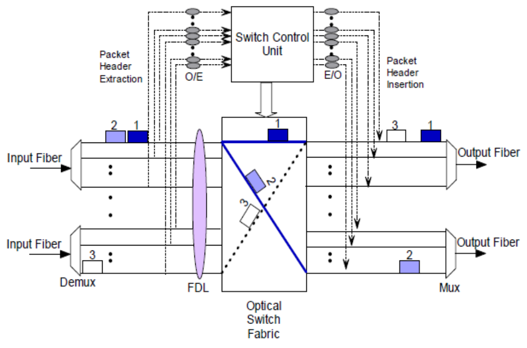


Figure 2. OPS architecture

2.1.3 Optical Burst Switching

A new switching paradigm called OBS has been proposed as the viable technology for the next generation optical internet, as shown on *fig. 3a*.

The OBS is a near-term alternative to OPS in which packets are assembled into bursts at the edge of the OBS network and kept in the optical domain, while its control packet or header can be converted to electronics for processing.

The OBS architecture follows two phases:

- Uses *one-way* (but can also use a *two-way* process) reservation process (*off-set*-based approach), where the packets are assembled into a burst.

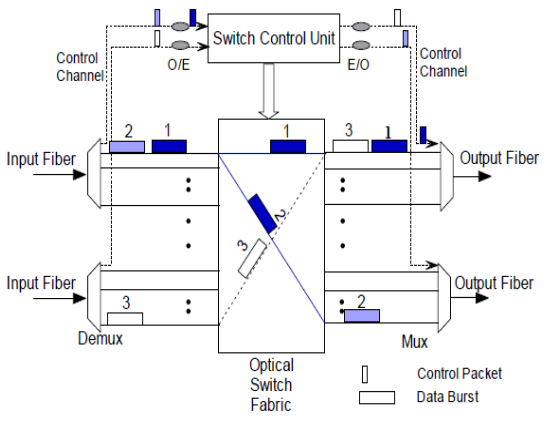
- An out-of-band *burst control packet* (BCP) leaves the source first to reserve resources and configure switchers along the burst's route in the network, while the data burst follows the BCP after an *off-set* time, as it is demonstrated on *fig. 3b*.

Since this technology uses an *off-set*-based approach, there is enough time for the intermediate nodes to process the BCP, avoiding the burst's delay during its journey to the destination node. It is used only one O/E/O converter for the control packet, which turns this technology simpler to implement in comparison to OCS that uses an O/E/O for every single packet. Furthermore, what mostly differentiates the OBS from the OPS is the lack of buffering in the network. Since BCP leaves the source first, to make the reservation along the path, there is no need for the burst to stop every time it reaches an optical switching node.

In spite of this technique does not need to wait for a connection's confirmation from the destination, there is a smaller delay provoked by the *off-set* time.

Another major drawback is the fact that, this technology uses most of the time *one-way* reservation process, which does not guarantee the successful reservation of resources, increasing the possibilities of bursts lost. The way to solve this problem is through the use of some contention techniques explained below.

a)



b)

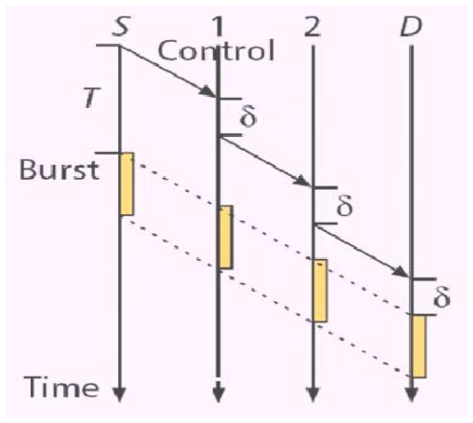


Figure 3. a) OBS architecture b) OBS transmission mechanism

In the OBS signaling/reservation and contention scope there are some protocols and techniques [13, 14] capable to complete their functions in different modes. Some of those protocols are used by different switching technologies.

In this project is used the *Just-in-Time* (JIT) reservation protocol [15]. The JIT is a reservation protocol for OBS networks that works in an out-of-band signalling process, eliminating the buffering of data bursts in intermediate nodes of the LSP. This protocol comparing to other reservation protocols, like *Just-enough-Time* (JET) [16], makes use of immediate reservation and explicit release. In this kind of approach, the optical channel is not used during the time between the BCP arrival and the burst arrival. However this kind of technique is not as efficient as the JET in terms of resource utilization but in another side, is simpler to implement and does not have complicated scheduling algorithms [15, 16].

A typical problem that all OBS networks face is the possible loss of data bursts in case the reservation is not accomplished at a switching point. Therefore, it is extremely necessary to find a solution when it is needed to hold bursts always when the out-link’s wavelengths of some OBS node are reserved or occupied for other burst transmission.

In this project the contention techniques are not yet implemented, but they are definitely one of the imminent issues that are going to be worked in a near future. As referred in [10], there are some techniques with the capacity of reducing the number of possible dropped bursts on the OBS network, one of them is the already mentioned, the FDL.

Another technique is the *Burst segmentation* (also known as preemption technique) that separates the burst into segments, dropping only those segments that may cover other burst when contention occurs. The most probably contention technique to be adopted in this project is the *deflection* [17, 18], which deflects the burst, to another available output when contention occurs. This type of technique is simpler to implement and guaranties the burst loss reduce.

2.2 GMPLS Overview

The GMPLS is the architecture adopted for the control plane implementation in this project. It is a well consistent framework, and can be easily extended by IETF when new requirements come up [6]. Certainly, the question is why such an extension is needed and what does it mean for? Remind that MPLS has been designed to *switch* packets using a labeling mechanism. Though, there is the need of a MPLS control-type functionality that is beyond just switching packets, such as the different switching capabilities referred in the previous chapter.

The big difference between GMPLS and MPLS is the different kind of traffic that can be switched and transmitted in the optical network. In response to the need of higher flexibility in optical networks, the GMPLS has proposed traffic engineering extensions to some of the MPLS signaling and routing protocols such as, the RSVP and the Open Shortest Path First (OSPF), allowing these ones to support dissimilar information that can be handled by the GMPLS Traffic Engineering [TE] switching capabilities [19].

The GMPLS extends two signaling protocols defined by the MPLS: the RSVP-TE [20**]** and the Constraint-based Routed Label Distribution Protocol (CR-LDP) [21]**.** There are also two traditional routing protocols: the OSPF-TE [22**]** and the Intermediate System to Intermediate System (IS-IS-TE) [23]. Besides that, the GMPLS proposes a new link management protocol, the Link Management Protocol (LMP) [24], whose functions are in [*Appendix D – GMPLS Framework and Control plane functions*].

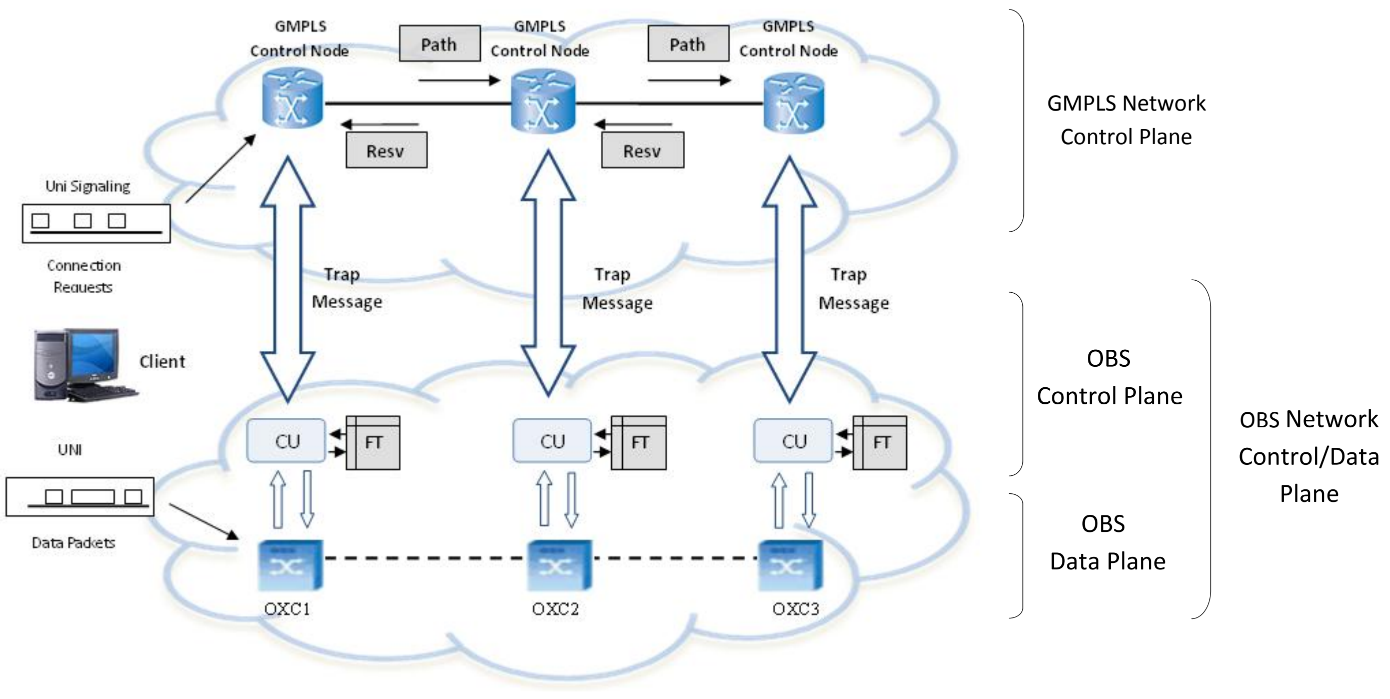
The GMPLS focuses on the control plane services that perform connection management for the data plane, including both packet-switched interfaces and non-packet-switched interfaces, as well as routing control and network topology discovery.

The fundamental service offered by the GMPLS control plane is dynamic end-to-end connection provisioning, where the operators need only to specify the connection parameters and send them to the ingress node. The GMPLS network control plane then determines the optical paths across the network according to the parameters that the user provides, and signals the corresponding nodes to establish the connection. The following point gives an explanation about the implemented RSVP-TE signaling protocol in this project.

2.2.1 RSVP-TE Signaling Protocol

The GMPLS-based control plane has two types of signaling protocols available to implement the LSP setup: the RSVP-TE [20, 25] and the CR-RDP.

Figure 4 below shows how the client’s data is transmitted through the OBS network, from a source (edge node) to a destination node by using the RSVP-TE signaling protocol architecture.



Path - Path Message;
Resv - Resv Message;
CU - Control Unit;
FT - Forwarding Table;
UNI - User-network Interface;
UNI Signaling - User-network signaling Interface;
OXC - Optical Cross-connect.

Figure 4. RSVP-TE signaling model

In the beginning of this signaling process, a client tries to send data packets (e.g. IP, Ethernet) from an edge to a destination node. Since there is no connection between the client and the physical network layer, the transmission of the data is done through a *User-network Interface* (UNI) from the adopted network overlay model. For example, consider the case that the client wants to transmit data from the edge node (*Optical Cross-Connect*, OXC1) to the destination node (OXC3) on *fig.* 4: the source OXC1 receives the packets from the client through the UNI, and assembles them into bursts. The edge node before launches the bursts to the OBS network, checks its forwarding table to see if a LSP to the OXC3 is already established. If so, the bursts are sent one by one following their BCPs until the destination node. If not, a connection request for a LSP is sent through a *user-network signaling interface* (UNI Signaling) to the corresponding GMPLS edge node in the control layer. That is where the GMPLS signaling protocol (RSVP-TE) comes into action. The connection request is filled up with some important information such as the source, the destination, the quality of service (QoS) and the load to be sent.

The GMPLS edge node receives the connection request and computes the two shortest paths (Shortest Path algorithms) according to the destination field received in the connection request message. Two RSVP-TE *Path* messages are created, each one with one of the shortest paths, and sent to the destination ingress node to setup the LSP between OXC1 and OXC3. The creation of two *Path* messages allows the destination node to decide which path is better to be used for the LSP establishment, basing on its arrival time. The first to arrive is defined as the LSP and the second is discarded. At the same time, two entries (corresponding to the next nodes ID of each path) are inserted in a temporary table of the GMPLS control node. Each *Path* message follows the corresponding shortest path, where the resources reservation process (i.e. bandwidth reserved on a desired output channel within the control network) is done every time the *Path* message is switched from a node to the next node. If all the reservation processes are correctly accomplished, the two *Path* messages reach successfully the destination node. Immediately a new RSVP-TE signaling message (*Resv)* is generated by the destination node, and sent back as acknowledge, confirming that the respective LSP is successfully established.

Meanwhile, and independently of what is happening on the control layer, the data nodes, through the respective OBS control Unit (CU), send updating messages (Trap messages) to the corresponding GMPLS control nodes, informing them about the links state on the physical data network. As an answer to those trap messages, each control node sends an acknowledge trap message as soon as it receives the *Resv* message, informing about the established tunnel LSP label, the next hop and the number of wavelengths to reserve. A tunnel in the GMPLS control plane signaling context is composed by a single LSP however, due to operations such as load distribution, LSP protection backup amongst others, a tunnel might be supported by several parallel LSPs [6].

On the other side, each physical node (OBS node) receives that information and updates its *Forwarding Table* (FT). If no error occurs during the RSVP-TE messages transmission, as soon as the OXC1 edge node receives the confirmation of the successful connection establishment, the BCP is sent to its destination followed by the corresponding burst in the OBS network. [9, 10]

2.3 GMPLS-OBS Architecture

The GMPLS-OBS network architecture is based on a transparent all-optical data plane and a hybrid control plane. Such hybrid control plane refers to a GMPLS control layer and an OBS control layer, where each one operates in separated networks, *fig. 5a* [7]. However, the integration of the GMPLS architecture in the OBS Control Plane Network has some variances:

1) OBS control plane uses in-fiber control, while GMPLS uses out-of-fiber (but also in-fiber) control.

2) OBS operates in one-way reservation process while GMPLS operates in two-way reservation process.

Due to the transmission of burst control packets (BCP) and data payloads, both optical/electronic control and all-optical data planes can be seen as two parallel networks, the *physical* (i.e. data) and the *control* network. On one hand, in the OBS control plane, control packets and the corresponding data bursts use the same optical fiber (in-fiber), or in other words, the bursts and the control packets use the same path (i.e. also known as, LSP) until the destination. On the other hand, the GMPLS control plane uses out-of-band, out-of-fiber (but it can be also in-fiber) control architecture, this means that control packets and the data bursts can travel into two separated fibers [26, 27]. The other major divergence between GMPLS and OBS is the signaling issue where GMPLS operates in a two-way reservation process opposite to the OBS that uses a one-way procedure. The two-way process means that, the GMPLS uses a connection oriented signaling technique, while the OBS uses packet oriented process. This issue brings up the question: how does the GMPLS control plane can be integrated with the OBS control plane?

The solution for these variations can pass by having two separated control planes (*fig. 5b*). Where the signaling, routing and link management functions are in charge by the GMPLS protocols [*Appendix D* – GMPLS Framework and Control Plane Protocols], while the process of physical resources reservation through the BCPs information is from the OBS control plane’s responsibility [7, 28]. Therefore, a high level of independency for both planes is achieved, providing a fine network management, and it also contributes to improve the optical resources availability leading to the reduction of the developing costs.

On what concerns the signaling decision, the one-way technique is the chosen technique, because in the presence of a large OBS networks the two-way reservation technique is totally excluded, due to the large latency (i.e. delay) in connections establishment [7].

This idea of separate both OBS- and GMPLS control planes, leads also to a separation of the OBS tasks [7]:

OBS Control Plane (Tasks): Resource Reservation; Network Resource Availability.

GMPLS Control Plane (Tasks): Virtual Topology Management; Network Topology.

OBS

Resource Reservation – Process of bandwidth reservation for data bursts transmission over the data plane, where the BCP contains all the necessary information to route the burst to the corresponding destination point.

Network Resource Availability – The OBS control plane is also responsible to gather all the information about the network’s resources availability, in order to achieve a better traffic balance and to reduce the lost-burst probability.

GMPLS

Virtual Topology Management – The GMPLS control plane is responsible to control the signaling processes as well as routing processes. On what concerns the signaling, the GMPLS control plane is in charge to setting up and tearing down the connection establishments in the OBS network in a two-way procedure. On the routing side, it has the function to compute the shortest paths for the ligthpaths establishment between the two OBS nodes, and update their forwarding tables stored in the control units within the data plane [7, 29].

Network Topology Information – This feature concerns about the link states management functions that can be done by the OSPF-TE routing protocol, under GMPLS framework. This protocol is also in charge of warning the OBS nodes about the links connectivity, fault management and other important information like the traffic distribution in the OBS network, aiming to the traffic balance to avoid possible burst losses.

a) b)

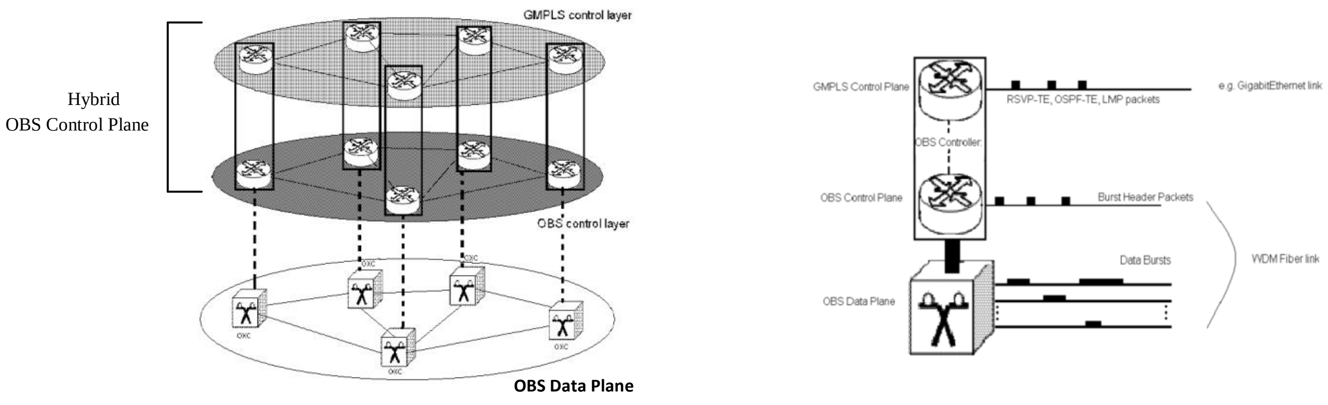


Figure 5. a) GMPLS/OBS Control Plane interoperability, b) Two physical separate networks [7].

Chapter III

A novel event-driven JAVA Simulator for GMPLS-OBS network

3.1 Introduction

The study of a possible integration scenario of the GMPLS control plane in the OBS networks will be done on top of an existent Java OBS network's simulator [JAVOBS]. In such way, it will be here further extended to cope with the basic features of GMPLS technology. A separated control layer will be deployed, comprising basic GMPLS features and messages, and will interoperate with OBS control layer, in order to provide reliable and efficient optical data forwarding service.

This chapter is organized as follows. Firstly, it is given a succinct explanation of the simulator's background. Secondly, it is explained how the simulator's extended version [JA-GOBS] was implemented in order to achieve the proposed objectives of this project.

3.1.1 Basis for JAVOBS

The JAVOBS is a flexible simulation tool running in a Java environment, exclusively developed for the study of OBS networks on top of the JAVANCO framework [8]. The JAVANCO is no more than an implemented software that works as a supporter to the JAVOBS simulator, where is possible to represent network topologies, graphs, as well as it also provides support for simulation models construction.

The name defined for this simulator's first version, JAVOBS, comes from: JAV – Because this simulator was entirely developed in Java programming language and - OBS because it is dedicated to the study of the optical burst switching paradigm. Using the functionality offered by JAVANCO, multiple OBS network topologies can be constructed over particular objects, called *NetworkHandlers*, as it is shown in *fig*. 6. The *NetworksHandlers* are the keystone of the JAVOBS architecture in charge of organizing the references towards each object, composing the graph (i.e. links, layers, nodes) and providing access to several engines and managers (i.e. interface manager, script manager). Furthermore, JAVOBS offers a basic OBS model which can be adapted and modified in many ways [8].

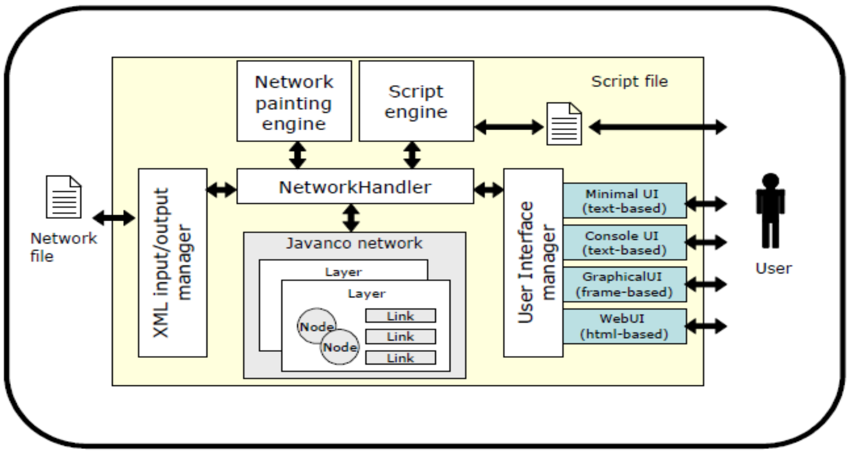


Figure 6. General perspective of the Javanco architecture [8]

This simulator was proposed to analyze the behavior of the OBS networks and its nodes when are submitted to burst transmission situations, in order to get statistical results about the number of success and unsuccessful transmissions in the network. In it, it is defined an OBS network composed by a group of nodes and the corresponding connecting links. There is also implemented a group of Java classes with their functions, which represent the major features and employed mechanisms for the bursts transmission simulation, like the OBS nodes, the bursts, the control packets (BCPs) and the resources reservation techniques (e.g. JIT and JET). All of this simulator's functions and developed methods will be explained with more detail in the JAVOBS extended version subsection.

3.1.2 Basis for JA-GOBS

The **JA**-**G**OBS is a new simulation package of the JAVANCO framework. The idea for **JA**-**G**OBS is due to: – **JA**, because it was deployed within a Java environment and – **G**OBS because it refers to the GMPLS control plane implementation over OBS networks.

The proposal for this new simulator's version is based on the GMPLS signaling protocol, RSVP-TE implementation. Following the defined scheme by having to separated planes [7, 30] (data and control), new nodes network were initially implemented, representing the GMPLS control plane. On one side, functions like the signaling and routing operations will be processed by the GMPLS control nodes (i.e. GMPLS control plane), on the other side, the extraction of burst control packets in each OBS core node will be associated to OBS control plane. On what concerns the data plane, this one will be in charge of bursts transmission in the all-optical domain.

3.2 JA-GOBS – Simulator Description

In an initial phase, the simulator (JAVOBS) was constituted by a single OBS network, where all functions from signaling to routing were rolled by the OBS control plane without any presence of any other control plane architecture.

Later, the simulator was extended to a new version (JA-GOBS) with the implementation of a new network referencing the GMPLS control plane, where it would be also implemented the respective GMPLS RSVP-TE signaling protocol. *Figure* 7 and 8 demonstrate the migration from a network composed by the OBS control/data plane to a two separated networks represented by the OBS data/control plane and the GMPLS control plane.

In *fig. 7* is represented the OBS network's data and control planes, which are composed by a group of OBS nodes (also referred as OXC nodes) connected by links. Each OBS node can either be an edge node if the data transmission starts from it or a core node if the data is received by or switched to another node. Remind that an edge node cannot receive any kind of traffic from another node.

Besides that, both OBS edge nodes and OBS core nodes have their own control units to handle with all kind of operations that are part of the bursts switching, which are defined by the classes *VirtualAggregatorController* (VC) and *SwitchController* (SC) respectively.

On *Fig. 8* are represented the two separated networks formed by the GMPLS control network, and on the bottom the OBS data/control network.

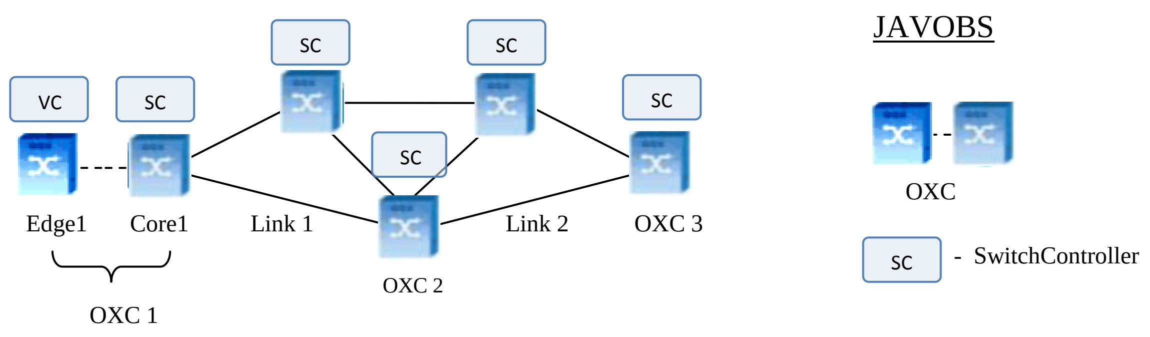


Figure. 7 OBS Network Data and Control Plane

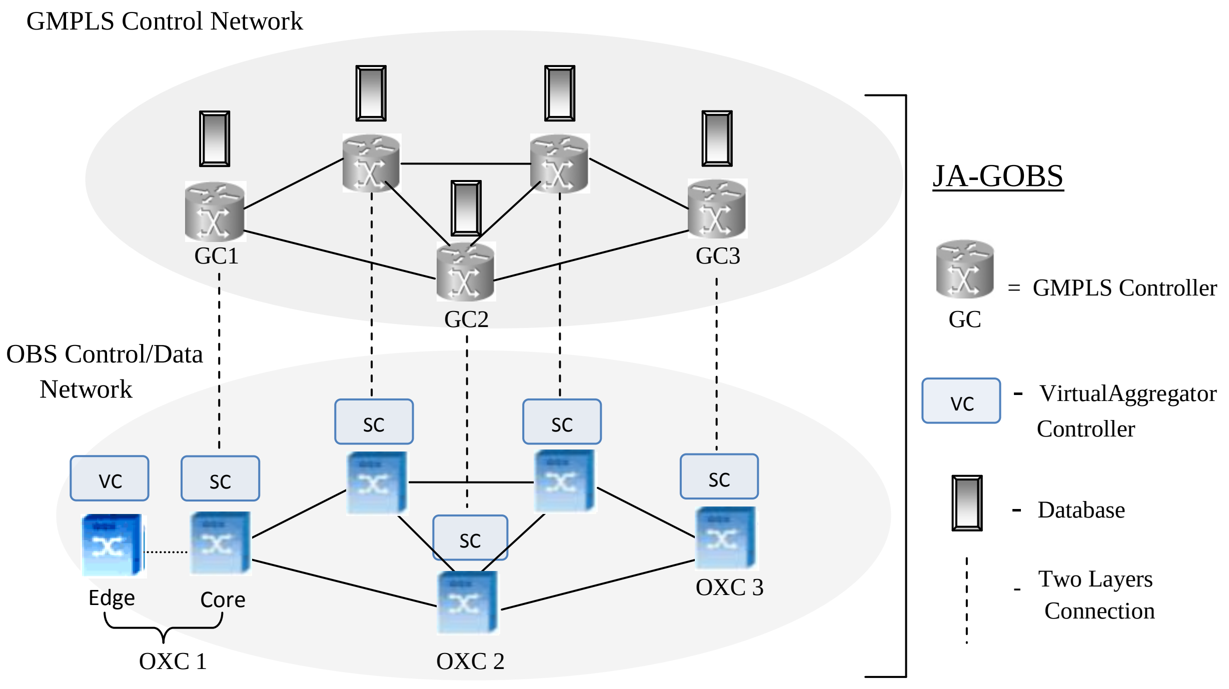


Figure. 8 GMPLS Control network and OBS Control/Data Network

<u>The Interaction between the different networks</u>

As a first approach, the communication between the GMPLS nodes and the OBS nodes is done by a relation 1:1, this means that each OBS node is only connected to its own GMPLS node. In a future, we might consider a 1:N (i.e. each OBS node will be able to interact with any GMPLS node, and vice-versa) relation establishment between the nodes of the two networks. To make this 1:1 interaction possible, it was necessary to implement four extra links for each OBS node. Figure 9 shows the two groups of two links between an OBS node and the corresponding GMPLS node.

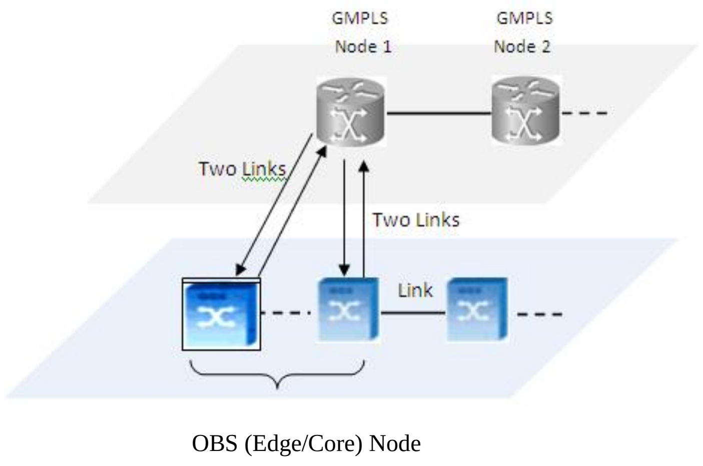


Figure 9. Links establishment between two nodes in different layers.

Two links with opposite directions are established for the OBS edge node (for the *connection request* transmission and its confirmation), and the other two links are established for the Trap Messages transmission and its acknowledgment.

The difference between these links and the links connected between nodes of the physical network is in its capacity. Links connecting two OBS nodes have 10 Gbit/s of capacity while the rest of them have 1 Gbit/s of capacity. This disparity is due to the fact that between the formers only client's traffic travels, while for the others only signaling messages travel.

The reservation process for LSP establishment

The migration from the *static* mode to the *dynamic* mode has changed the way of how the LSP establishment process is implemented. In the situation that a LSP is not established, the OBS source node has to request to the GMPLS node for a connection establishment. This action of computing and establishing a LSP is in charged by GMPLS RSVP-TE signaling protocol.

Many important processes occur before the LSP establishment is completed and one of those processes concerns on the way that the reservation procedure is made. As referred on the previous chapter, the establishing of a connection between a source node and a destination node demands the resources reservation in each intermediate node of the path.

There are two different situations for the resources reservation on a LSP establishment, during the *Path* message exchange between the GMPLS nodes: the normal reservation, where the link in process is only used to carry out traffic of one LSP at the time and the second situation, where the link in process needs to carry out the traffic from two different LSPs at the same time. For both situations, the reservation procedure is made through the use of the well known Erlang B table [*Appendix* H – *Erlang B Traffic Table*]. On a normal reservation situation, each GMPLS node receives trap messages from its corresponding OBS node and extracts the expected load ρ and the QoS factor. On the presence of those two features {load, Qos}, the Erlang B table is used to get the exactly number of necessary wavelengths to achieve the QoS defined for each burst transmission.

However, there are some different situations that might occur, changing the way of how the reservation process is made. On *fig. 10* is exemplified a reservation situation when a link is shared at the same time by two different LSPs.

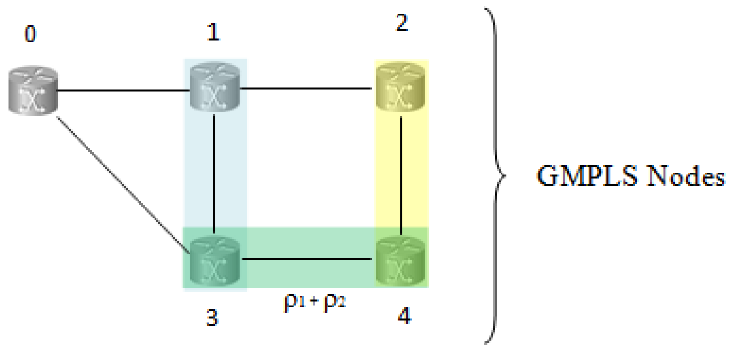


Figure 10. Control Network - Resources reservation for a shared link.

The GMPLS node 1 and node 3 are the sources of two different LSPs with paths [1; 3; 4] and [3; 4; 2] respectively and both have in common the link [3; 4] in order to reach their destination node. The solution is to summon up both loads ρ_1 and ρ_2 of the link in common and pass it, also with the QoS factor, to the respective simulator's method. This method uses the Erlang B table to get the total number of λ wavelengths, which will carry out the traffic that is shared by both LSPs, taking into account the used scheduling algorithm, last available unscheduled channel (LAUC) explained in [9].

3.3 Implementation Description

This section explains how the simulator is implemented and how it works, first with one layer (OBS Data/Control Plane) and then with the new implemented GMPLS control layer.

In the first phase, before the GMPLS control layer development, the simulator was defined by a group of important classes: the *Obs_Simulator* [*Appendix A – JA-GOBS Configuration Classes*] is the main class or, in other words, the engine for each simulation. Then, there are the classes that define the OBS nodes such as the *Obs_Edge_Node* and the *Obs_Core_Node,* the corresponding OBS nodes control units *VirtualAggregatorController* (VC), *SwitchController* (SC) and the classes that represent the reservation protocols (e.g. JIT, and JET) [*Appendix B –OBS Data/Control Plane main classes*].

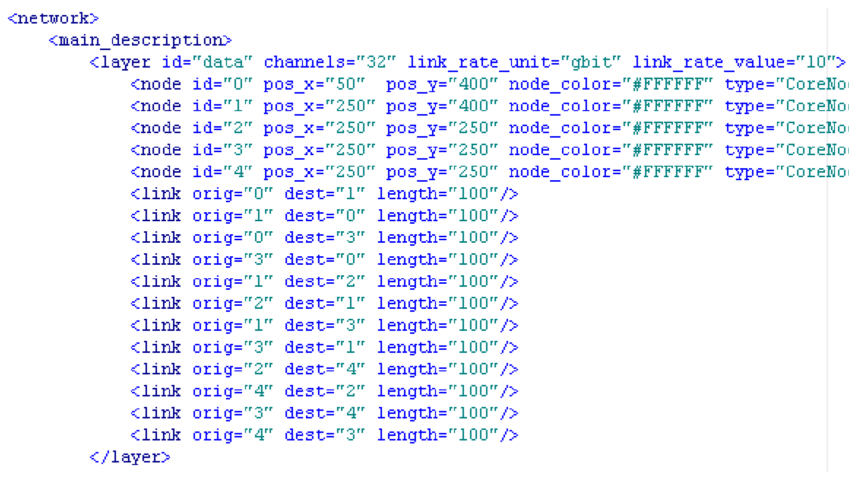
Running the simulator in a *static* mode [*Appendix G – JAVOBS configuration classes diagram*] was the first scheme to take place before we assume the change for a *dynamic* mode. The idea with the *static* mode was to get a first approach of all functions and procedures implementation, with a list of LSPs already established for the OBS network.

Later on, with two implemented networks, the *static* mode is changed to a *dynamic* mode, where instead of having a list of established LSPs, each OBS node needs to request for a LSP establishment to the corresponding GMPLS node, every time the connection between a source and a destination does not exist. The idea for this change is to follow as much as possible the real mechanism of bursts transmission in the OBS network with the GMPLS control plane integrated.

3.3.1 JAVOBS Simulator – One Network Layer

Each simulation is events-based that occur between a number of steps defined by the user. Within a 30ms length step, some actions take place from the traffic generated to the bursts transmission.

In general when the simulator is running the first thing to do is to load the OBS data network from the XML file (*5nodes.XML)* with all the nodes and links connected to each other, like it is shown on *fig. 11*.



```
<network>
    <main_description>
        <layer id="data" channels="32" link_rate_unit="gbit" link_rate_value="10">
            <node id="0" pos_x="50"  pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
            <node id="1" pos_x="250" pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
            <node id="2" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
            <node id="3" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
            <node id="4" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
            <link orig="0" dest="1" length="100"/>
            <link orig="1" dest="0" length="100"/>
            <link orig="0" dest="3" length="100"/>
            <link orig="3" dest="0" length="100"/>
            <link orig="1" dest="2" length="100"/>
            <link orig="2" dest="1" length="100"/>
            <link orig="1" dest="3" length="100"/>
            <link orig="3" dest="1" length="100"/>
            <link orig="2" dest="4" length="100"/>
            <link orig="4" dest="2" length="100"/>
            <link orig="3" dest="4" length="100"/>
            <link orig="4" dest="3" length="100"/>
        </layer>
```

Figure 11. XML file with the OBS Physical Network

Our OBS data network is composed by five nodes, each one with its own index from 0 to 4, followed by the link connections between them. On the following *fig*. 12, is represented the OBS network topology, which will be the same for the GMPLS control network.

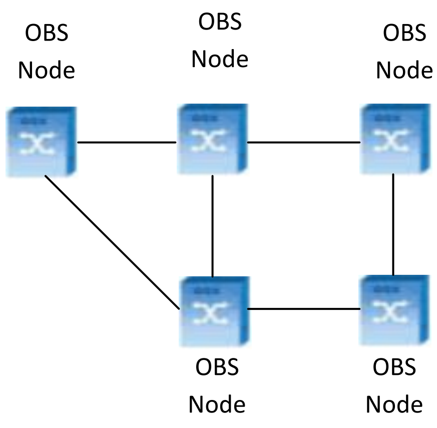


Figure 12. OBS Data Network's topology

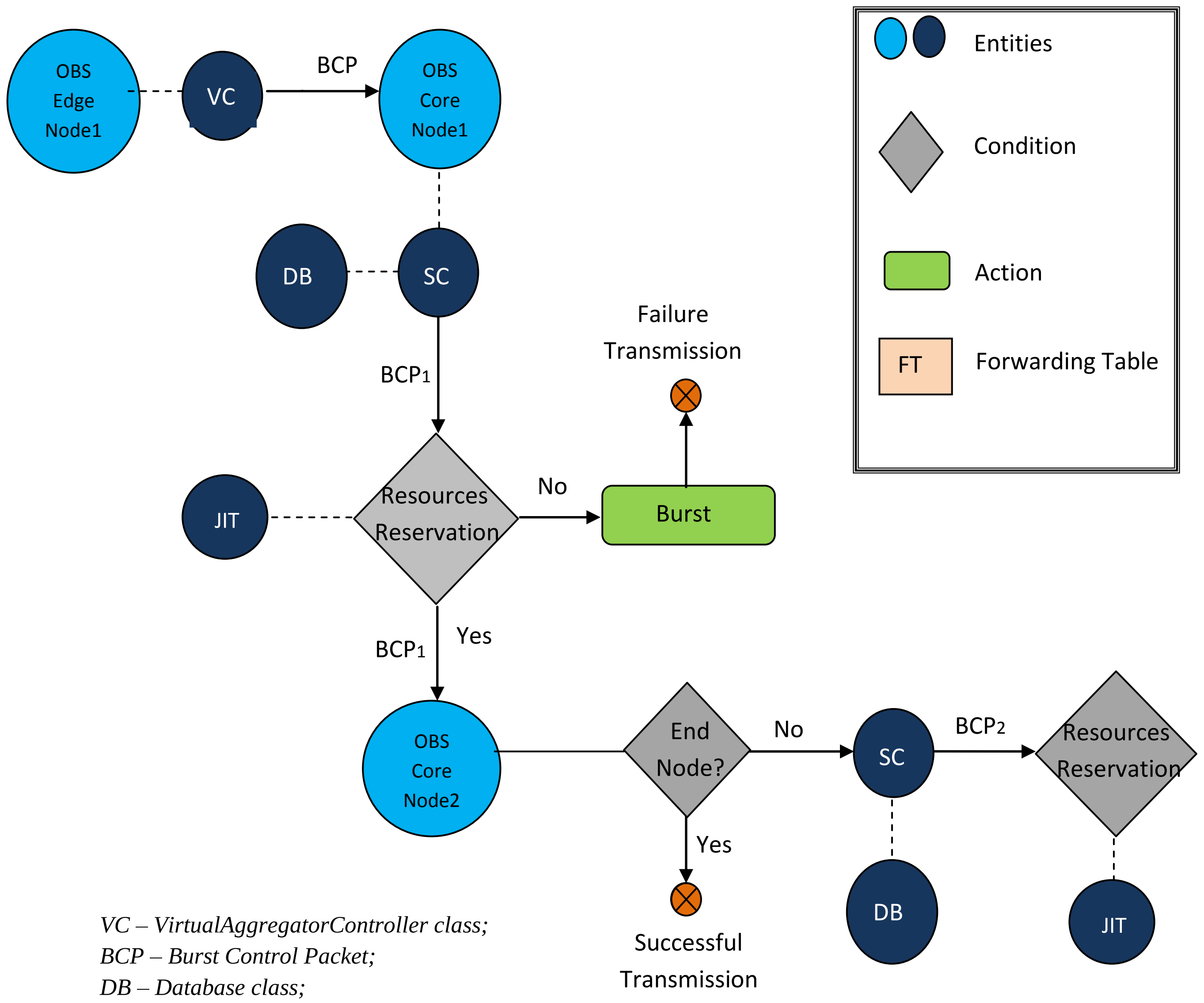
For the node 0, there are two link connections with two other nodes, node 1 and node 3. It is important to take into account that each connection between two nodes has two established links, one for one direction and the other for the opposite, for instance (*link orig=0 dest=1* and *link orig=1 dest=0*) defined in the XML file.

Besides the network initialization, all the other essential features are also loaded within the class *Obs_Simulator* [*Appendix A – JA-GOBS Configuration Classes*], such as the calculation of the network diameter, the number of steps, the assignment of the number of wavelengths per link (given by *Mixed Integer Linear Programming,* MILP formulation problem), burst length, the quantity of load to transmit, the tunnels setup for the transmission of *High Priority* (HP), *Best Effort* (BE) data bursts and the *Database* initialization.

On what concerns the nodes (Core/Edge) classes, the *Obs_Edge_Node* class takes care of the generated traffic from the *Obs_Sim_Generator* [*Appendix A – JA-GOBS Configuration Classes*] class to the *Obs_Edge_Node* control unit class *VirtualAggregatorController (VC)* and gives the initial start of the steps transmission sequence. On the other hand, the *Obs_Core_N*ode makes the resources reservation of the network and all the procedures that must be done before sending any information to its neighbor nodes. The bond between the *Obs_Core_Node* class and the reservation protocol class, the *Just_In_Time* class, is made through its control unit, the *SwitchController (SC)* class. This protocol has the function of checking if there are internal available resources to transmit the data burst to the next destination node.

The following *fig. 13* explains step by step the interaction of the JAVOBS java classes for the process of bursts transmission in the OBS network, running on *static* mode. The initial phase of this transmission process starts with the OBS data network initialization, as well as the initialization of all the important features, such as the defined list of the established LSPs and the correct wavelengths assignment, forming what was called the *Virtual Network Topology*. Due to the presence of a default list of established LSPs, the edge node after receive the generated data traffic, already knows what LSP to use to start the transmission. Therefore, its only function is to send the BCP and the corresponding burst towards its destination node.

Simulation Scheme of the OBS Network Control/Data Plane



VC – VirtualAggregatorController class;
BCP – Burst Control Packet;
DB – Database class;
SC – SwitchController class;
JIT – Just_In_Time class.

Figure 13. Simulation Scheme of the OBS Network Control/Data Plane

The initial phase starts with the class *Obs_Sim_Generator* generating bursts to be sent by *Obs_Edge_Node*. Since this simulator only works with traffic transmission between core nodes, the BCP information is initially sent from the *Obs_Edge_Node* control unit VC to the *Obs_Core_Node* class. The *Obs_Core_Node* control unit, *SC*, checks the priority of the BCP packet, if it is *High Priority* (HP) or *Best Effort* (BE), after that, a *Database (DB)* call is made to get the next hop of the LSP. All the necessary burst control information is collected and a new BCP_1 is mounted.

On the following phase, the SC checks if there are enough available resources to reserve, by calling the reservation protocol JIT (*Just_In_Time*) class. If it receives an *ack* from the JIT class, all the resources are reserved for the burst, otherwise, that burst will be discarded. The same process continues until the destination node.

Meanwhile, all successful and unsuccessful burst transmissions from a source to a destination are counted and collected by the class *ObsSimulationDataManager* [*Appendix E – OBS Network Classes Diagram*], which at the end of the simulation, a final statistical result is displayed.

3.3.2 JA-GOBS Simulator – Two Network Layers

Two implemented network layers will show us how the OBS networks work with two separated networks, where one refers to the OBS data plane and control plane while the other refers to the GMPLS control plane.

After everything has been checked and run correctly with just one network layer, it is time to optimize the architecture. The idea is to use the GMPLS-based control plane architecture to manage all the burst transmissions in the OBS network.

It was introduced a new network of nodes in the same XML file, corresponding to the GMPLS control nodes and its link as shown on *fig. 14.*

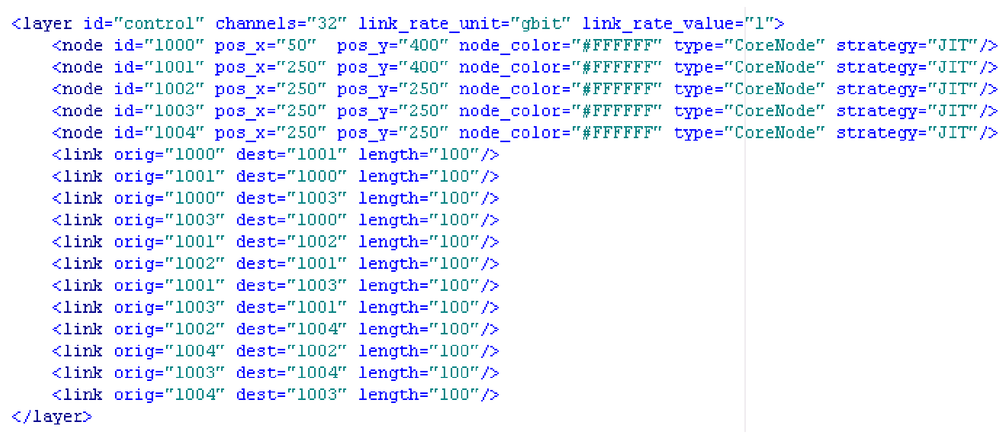

```
<layer id="control" channels="32" link_rate_unit="gbit" link_rate_value="1">
    <node id="1000" pos_x="50"  pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
    <node id="1001" pos_x="250" pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
    <node id="1002" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
    <node id="1003" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
    <node id="1004" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JIT"/>
    <link orig="1000" dest="1001" length="100"/>
    <link orig="1001" dest="1000" length="100"/>
    <link orig="1000" dest="1003" length="100"/>
    <link orig="1003" dest="1000" length="100"/>
    <link orig="1001" dest="1002" length="100"/>
    <link orig="1002" dest="1001" length="100"/>
    <link orig="1001" dest="1003" length="100"/>
    <link orig="1003" dest="1001" length="100"/>
    <link orig="1002" dest="1004" length="100"/>
    <link orig="1004" dest="1002" length="100"/>
    <link orig="1003" dest="1004" length="100"/>
    <link orig="1004" dest="1003" length="100"/>
</layer>
```

Figure 14. XML file with the GMPLS Control Network

With a relation 1:1 between each node of the two layers, it was decided to set the GMPLS nodes with an ID starting with the OBS node ID plus 1000. For that reason the OBS node with ID 0 has a relation to the GMPLS node with ID 1000 plus 0, and so on for the rest of the nodes.

After run the *Obs_Simulator* class, the same features explained before are initialized together with the new ones: the new GMPLS control nodes, its links and the links which connect the two layers in order to allow each OBS node to interact with the corresponding control node.

However, there is a difference facing the old architecture. Whereas the optical tunnels with one layer are all established, some of the LSPs with two layers will not be defined after this optimization. In this case, when an OBS edge node wants to send a burst to a destination node and the LSP is not yet defined, it will have to request for a LSP establishment (i.e. a connection) to the corresponding GMPLS control node in the layer above [*Appendix F – GMPLS Network classes Diagram*].

Figure.15 shows the RSVP-TE protocol signaling process implemented in JA-GOBS, and how it acts when a connection request is required. Since this protocol was already explained, it will be given a briefly resume of the signaling messages exchange, focusing essentially on the new JA-GOBS classes interaction.

RSVP-TE Signaling Process diagram

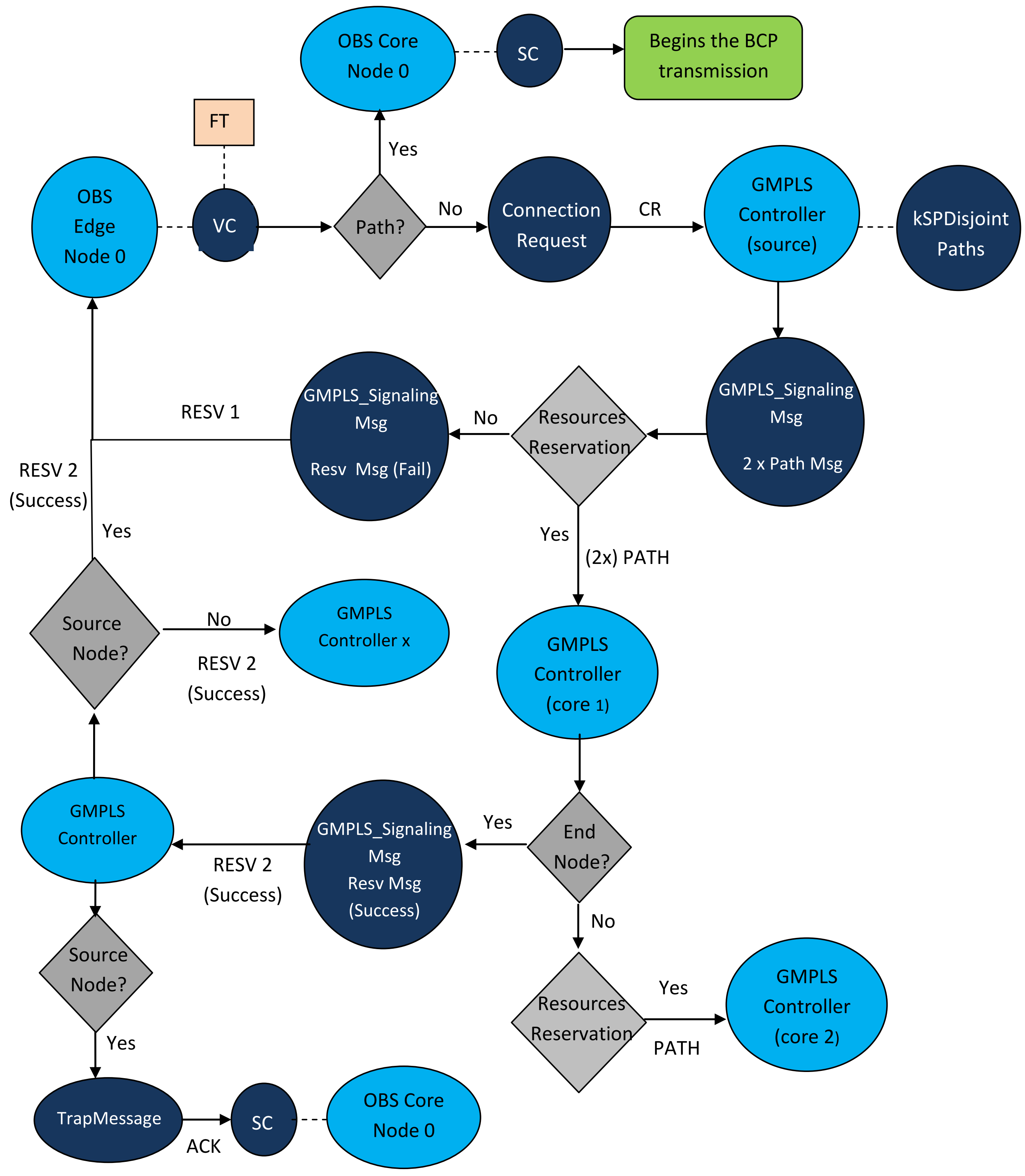


Figure 15. RSVP-TE Signaling Process Diagram

The RSVP-TE signaling protocol is constituted by a main class, *GMPLSController* [*Appendix C – GMPLS Control Plane main class*] which represents each GMPLS control node of the network. Following the steps explained about the RSVP-TE signaling protocol subsection of the previous chapter, each *Obs_Edge_Node* checks its forwarding table (FT) to see if the pretended connection is already defined. If so, the normal burst transmission starts either in the OBS data plane, or a *Connection Request* is sent by the *VC* to the GMPLS edge node of the layer above.

When that control node receives the request, the two distinct shortest paths are computed by calling the class *kSPDisjointPaths*, where the "*k*" is the number of paths pretended. When the two new *Path* signaling messages are built, each one is represented by the *GMPLS_SignalingMsg* class. The difference between the *Path* and the *Resv* messages is the field "*Type*" that differentiates one from the other. The next step is to send those two signaling messages to the destination node following distinct paths. This process is done by calling a class defined as *GMPLSTransmitter.*

The control core node on the control plane receives one of the *Path* messages and checks if the destination node ID corresponds to its ID. If so, a new signaling message is created (Type: *Resv*) and sent back to the source control node. At the same time on every Resv message switching, a new message (*ACK TrapMessage)* represented by the *TrapMessage* class is sent to the equivalent OBS core node, allowing this one to update its FT. However, we must take into account the path IDs conversion: when the ACK Trap Message is sent to each OBS node of the path but the destination node, the ID of the GMPLS node is subtracted by 1000 in order to get the correctly physical node ID.

When the *Resv* message reaches the destination node, this one warns the OBS edge node, represented by the *Obs_Edge_Node* class, about the successfully tunnel establishment, completing the signaling process.

Chapter IV

Signaling Protocol - Simulation Demo

4.1 Introduction

In this chapter, a demonstration of the GMPLS signaling protocol RSVP-TE for the LSP establishment between two OBS nodes is done, where different phases of the signaling process are shown and explained in detail. The main idea of this demo is to show all the steps that are made during the signaling process, with the respective networks initialization, followed by the connections request generation, and ended with the respective exchange of the RSVP-TE signaling messages in order to successfully establish a LSP for the burst transmission.

4.2 Simulation Demo conclusions

This demonstration allows the analysis of the simulator functionality after have been implemented all the proposed tasks. This demonstration can be divided into a group of three important sections. The initialization of the new control network (GMPLS) and the correct interaction made by the two different layer nodes, let us to think on a possible optimization where each OBS node will be allowed to interact with any of the GMPLS control nodes, avoiding the existed 1:1 relationship. With that change we can get a GMPLS control network topology completely different from the defined in the OBS network, which in fact, takes us closer of how the real interaction between the two different networks should be. Finally, the signaling section, where the RSVP-TE signaling protocol basic procedures such as, the correct exchange of both signaling messages (*Path, Resv*) for the LSP establishment, has let us to conclude that the main two objectives for this project were successfully accomplished.

However some functions of this signaling protocol are not implemented yet, such as:

- The resources reservation failure for the LSP establishment request in the control layer, leads to a new signaling message, with type *Resv*, confirming to the OBS edge node that the LSP was not possible to establish.

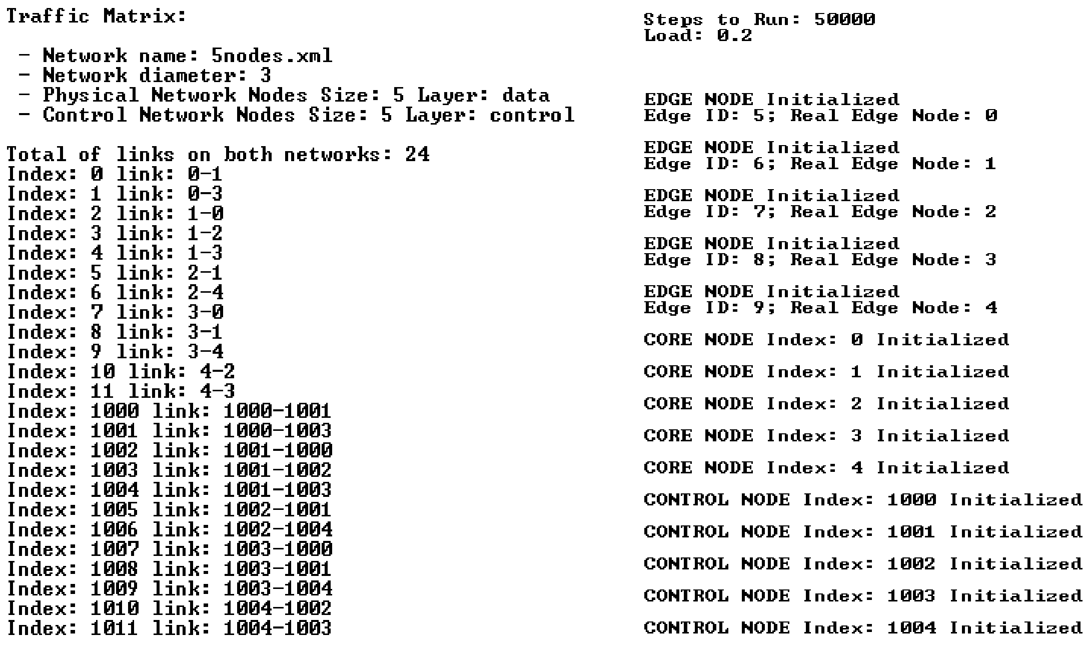
- From the time that a LSP is no longer being used for any bursts transmission, it must be disconnected, through the use of the signaling message *Teardown*.

Networks – Nodes & Links Initialization

On the following *Fig*. 16 is shown the beginning of each simulation with all the initially configured features and the respective networks initialization, composed by its nodes and links. All of the nodes are identified with an index: the OBS core nodes with an index from 0 to 4, the OBS edge nodes also with their pair's index from 0 to 4 plus their own index from 5 to 9 and, finally, the GMPLS control nodes with the OBS core's index + 1000. The option for that number is due to the fact that the OBS network can be enlarged at any time, so the best way to differentiate both nodes from each layer is to give a large index number to the GMPLS control nodes. After the nodes initialization, the links are automatic initialized with the respectively index numbers.

The number of steps defined for this simulation demo is 50000, however, since we are just going to pay attention to the RSVP-TE signaling function for one LSP establishment, the demonstration will end before it reaches that number.

The next phase leads us to a first study of the connections generation, where in each 5ms of a step, one connection request of a total of four, is forwarded to the GMPLS node from each node of the OBS network.


```
Traffic Matrix:

 - Network name: 5nodes.xml
 - Network diameter: 3
 - Physical Network Nodes Size: 5 Layer: data
 - Control Network Nodes Size: 5 Layer: control

Total of links on both networks: 24
Index: 0 link: 0-1
Index: 1 link: 0-3
Index: 2 link: 1-0
Index: 3 link: 1-2
Index: 4 link: 1-3
Index: 5 link: 2-1
Index: 6 link: 2-4
Index: 7 link: 3-0
Index: 8 link: 3-1
Index: 9 link: 3-4
Index: 10 link: 4-2
Index: 11 link: 4-3
Index: 1000 link: 1000-1001
Index: 1001 link: 1000-1003
Index: 1002 link: 1001-1000
Index: 1003 link: 1001-1002
Index: 1004 link: 1001-1003
Index: 1005 link: 1002-1001
Index: 1006 link: 1002-1004
Index: 1007 link: 1003-1000
Index: 1008 link: 1003-1001
Index: 1009 link: 1003-1004
Index: 1010 link: 1004-1002
Index: 1011 link: 1004-1003
```

```
Steps to Run: 50000
Load: 0.2


EDGE NODE Initialized
Edge ID: 5; Real Edge Node: 0

EDGE NODE Initialized
Edge ID: 6; Real Edge Node: 1

EDGE NODE Initialized
Edge ID: 7; Real Edge Node: 2

EDGE NODE Initialized
Edge ID: 8; Real Edge Node: 3

EDGE NODE Initialized
Edge ID: 9; Real Edge Node: 4

CORE NODE Index: 0 Initialized

CORE NODE Index: 1 Initialized

CORE NODE Index: 2 Initialized

CORE NODE Index: 3 Initialized

CORE NODE Index: 4 Initialized

CONTROL NODE Index: 1000 Initialized

CONTROL NODE Index: 1001 Initialized

CONTROL NODE Index: 1002 Initialized

CONTROL NODE Index: 1003 Initialized

CONTROL NODE Index: 1004 Initialized
```

Figure 16. Simulation Demo – Nodes and Links Initialization

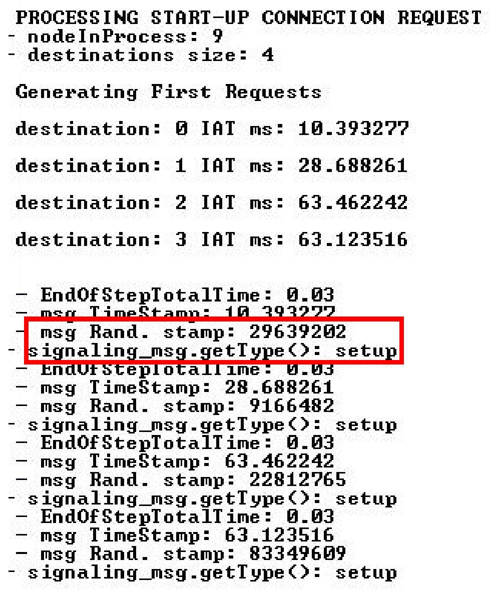
Connection Generation

The idea in the connections generation phase is to follow a dynamic scenario, where the interaction between the two nodes of each network for a LSP establishment should be done dynamically. As mentioned before, each OBS node has four LSP connections to request, each one for a different destination of the network. Since we are just analysing and proving that the RSVP-TE signaling protocol is correctly implemented, only one LSP establishment is demonstrated for just one node of the OBS network.

As mentioned before, the RSVP-TE signaling protocol is only called if a LSP is not established from a source to a destination point in the data network, therefore, a connection request is generated and sent from the OBS edge node to the corresponding GMPLS node.

On *fig. 17* the node in process has an index 9 that means we are in the presence of the OBS edge node with core's index 4 (remember that each OBS edge node has its own index). If there are 5 OBS core nodes from indexes 0 to 4, then, the core node 0 has an edge pair node with index 5, therefore the forth core node has an edge pair's index of 9. From now on we are always referring to the core's indexes.

Continuing with our example, let's pay attention to the connection request with index *29639202* (red box) forwarded from the OBS node 4 to the GMPLS node, in order to establish a path towards the destination 0. An *initial arrival time* (IAT) of 10,39 ms is computed, this means that the corresponding GMPLS node receives the first connection request at 10,39 ms of a step with 30ms of duration.



```
PROCESSING START-UP CONNECTION REQUEST
- nodeInProcess: 9
- destinations size: 4

Generating First Requests

destination: 0 IAT ms: 10.393277

destination: 1 IAT ms: 28.688261

destination: 2 IAT ms: 63.462242

destination: 3 IAT ms: 63.123516

 - EndOfStepTotalTime: 0.03
 - msg TimeStamp: 10.393277
 - msg Rand. stamp: 29639202
- signaling_msg.getType(): setup
 - EndOfStepTotalTime: 0.03
 - msg TimeStamp: 28.688261
 - msg Rand. stamp: 9166482
- signaling_msg.getType(): setup
 - EndOfStepTotalTime: 0.03
 - msg TimeStamp: 63.462242
 - msg Rand. stamp: 22812765
- signaling_msg.getType(): setup
 - EndOfStepTotalTime: 0.03
 - msg TimeStamp: 63.123516
 - msg Rand. stamp: 83349609
- signaling_msg.getType(): setup
```

Figure 17. Simulation Demo – Connection Request generation

Signaling Process Initialization

This phase refers to the LSP end-to-end signaling process initialization. From now on, we are just focus on the GMPLS network, where the first two signaling *path* messages are implemented (by the node 1004) in order to be sent to the respective destination (node 1000). Each *path* message is differentiated from the other, through the colourful sections that surrounds it (red section for the *Path* message is the shortest path, and the green section for the *path* message is longest path).

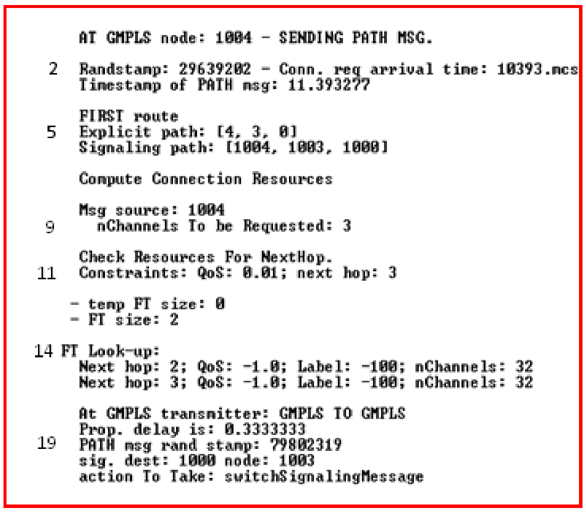
Figure 18 shows the period of the signaling process where the GMPLS node with index 1004, receives the first LSP *connection request* message with index *29639202*, from its corresponding OBS node with index 4 (line 2- Red section). Following with our demonstration, the GMPLS node with index 1004 computes the first of two shortest paths and gets the first route with an explicit path [4, 3, 0] (line 5 – Red section), with the equivalent signaling path [1004, 1003, 1000]. In the resources reservation computation, some features are detailed such as the source of the message and the total of wavelengths necessary to reserve (line 9 – Red section) and on line 11 is displayed the QoS priority, which in this case is *High Priority* (i.e. 0.01), otherwise it would be *Best Effort* (i.e. -1.0).

Following this example, line 14 shows the number of entries and the next hop characteristics of the node 1004 Forwarding Table (*FT*). As we can see there are already two entries in the table belonging to *Best Effort* transmissions.

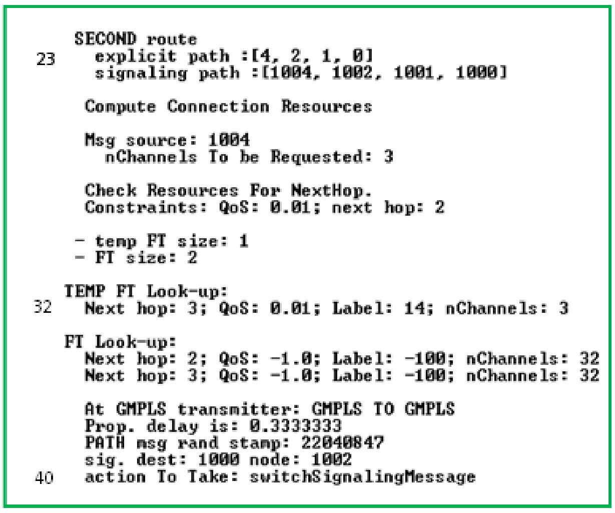
The first *Path* message is built and ready to be sent. However, since there are two types of *Path* messages to send, the second route (Green section) is also computed with a different shortest path (line 23 – Green section). In this case the second shortest *Path* message has to pass by two intermediate nodes 1002 and 1001 before it reaches its destination node 1000, rather than the first *path* message that only has one intermediate node. This longer way increases the possibility of unsuccessful resources reservation.

If we look carefully at line 32, it shows a temporary FT of node 1004, meaning that a path was already established from node 1004 to node 1003 carrying *High Priority* traffic, identified by label 14. This entry refers exactly to the first route computed before.

To complete the analysis of this message, it is set an identification number of *22040847* (line 39). Both messages are now defined and ready to carry on their own journey until the destination node 1000.



```
AT GMPLS node: 1004 - SENDING PATH MSG.

Randstamp: 29639202 - Conn. req arrival time: 10393.mcs
Timestamp of PATH msg: 11.393277

FIRST route
Explicit path: [4, 3, 0]
Signaling path: [1004, 1003, 1000]

Compute Connection Resources

Msg source: 1004
  nChannels To be Requested: 3

Check Resources For NextHop.
Constraints: QoS: 0.01; next hop: 3

- temp FT size: 0
- FT size: 2

FT Look-up:
Next hop: 2; QoS: -1.0; Label: -100; nChannels: 32
Next hop: 3; QoS: -1.0; Label: -100; nChannels: 32

At GMPLS transmitter: GMPLS TO GMPLS
Prop. delay is: 0.3333333
PATH msg rand stamp: 79802319
sig. dest: 1000 node: 1003
action To Take: switchSignalingMessage
```



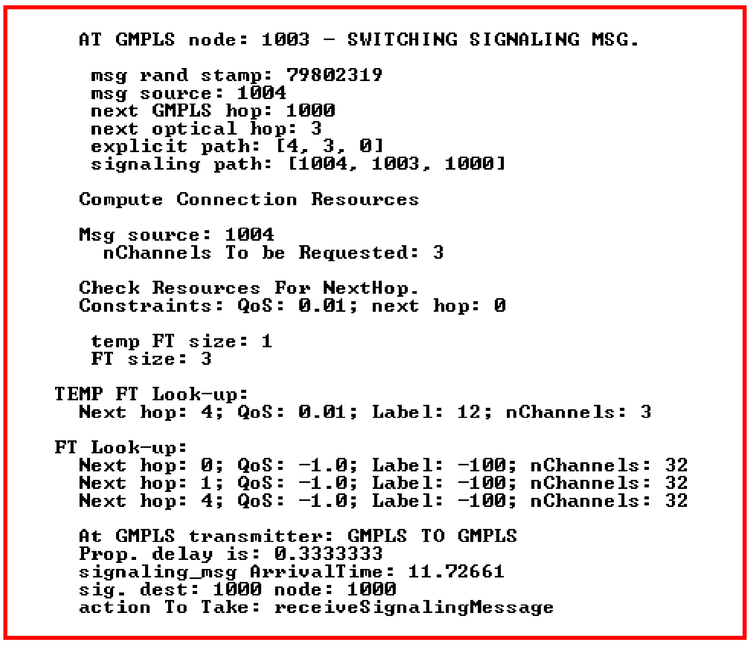
```
SECOND route
  explicit path :[4, 2, 1, 0]
  signaling path :[1004, 1002, 1001, 1000]

 Compute Connection Resources

 Msg source: 1004
   nChannels To be Requested: 3

 Check Resources For NextHop.
 Constraints: QoS: 0.01; next hop: 2

- temp FT size: 1
- FT size: 2

TEMP FT Look-up:
 Next hop: 3; QoS: 0.01; Label: 14; nChannels: 3

FT Look-up:
 Next hop: 2; QoS: -1.0; Label: -100; nChannels: 32
 Next hop: 3; QoS: -1.0; Label: -100; nChannels: 32

 At GMPLS transmitter: GMPLS TO GMPLS
 Prop. delay is: 0.3333333
 PATH msg rand stamp: 22040847
 sig. dest: 1000 node: 1002
 action To Take: switchSignalingMessage
```

Figure 18. Simulation Demo – Path messages Initialization

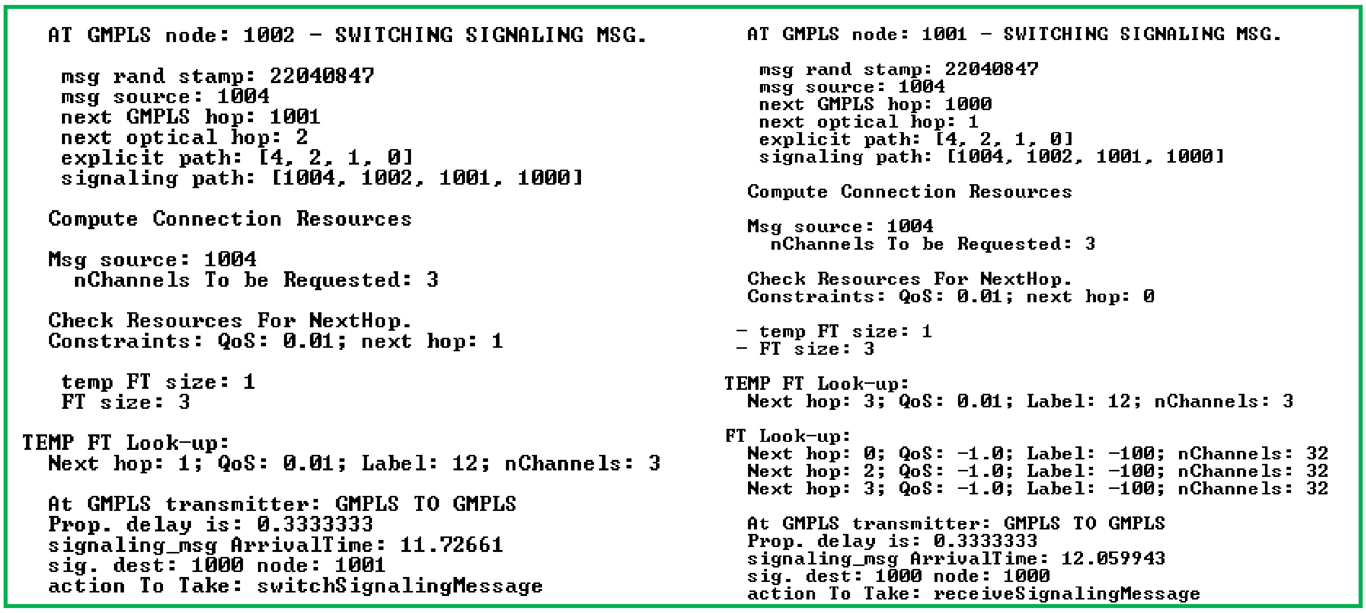
Switching Signaling Path Messages

This phase of the LSP establishment shows the switching of the *Path* messages.

On *Fig*. 19 is shown the two *Path* messages switching process, distinguished by the colourful sections. The red section, represents the *Path* message with the shortest path (i.e., the first route), below the red section we have the second route's *Path* message (green section) with the longer path [1004, 1002, 1001, 1000] corresponding to the explicit path [4, 2, 1, 0].



```
AT GMPLS node: 1003 - SWITCHING SIGNALING MSG.

 msg rand stamp: 79802319
 msg source: 1004
 next GMPLS hop: 1000
 next optical hop: 3
 explicit path: [4, 3, 0]
 signaling path: [1004, 1003, 1000]

Compute Connection Resources

Msg source: 1004
  nChannels To be Requested: 3

Check Resources For NextHop.
Constraints: QoS: 0.01; next hop: 0

 temp FT size: 1
 FT size: 3

TEMP FT Look-up:
 Next hop: 4; QoS: 0.01; Label: 12; nChannels: 3

FT Look-up:
 Next hop: 0; QoS: -1.0; Label: -100; nChannels: 32
 Next hop: 1; QoS: -1.0; Label: -100; nChannels: 32
 Next hop: 4; QoS: -1.0; Label: -100; nChannels: 32

At GMPLS transmitter: GMPLS TO GMPLS
Prop. delay is: 0.3333333
signaling_msg ArrivalTime: 11.72661
sig. dest: 1000 node: 1000
action To Take: receiveSignalingMessage
```



```
  AT GMPLS node: 1002 - SWITCHING SIGNALING MSG.

   msg rand stamp: 22040847
   msg source: 1004
   next GMPLS hop: 1001
   next optical hop: 2
   explicit path: [4, 2, 1, 0]
   signaling path: [1004, 1002, 1001, 1000]

  Compute Connection Resources

  Msg source: 1004
    nChannels To be Requested: 3

  Check Resources For NextHop.
  Constraints: QoS: 0.01; next hop: 1

   temp FT size: 1
   FT size: 3

TEMP FT Look-up:
  Next hop: 1; QoS: 0.01; Label: 12; nChannels: 3

  At GMPLS transmitter: GMPLS TO GMPLS
  Prop. delay is: 0.3333333
  signaling_msg ArrivalTime: 11.72661
  sig. dest: 1000 node: 1001
  action To Take: switchSignalingMessage
```

```
  AT GMPLS node: 1001 - SWITCHING SIGNALING MSG.

   msg rand stamp: 22040847
   msg source: 1004
   next GMPLS hop: 1000
   next optical hop: 1
   explicit path: [4, 2, 1, 0]
   signaling path: [1004, 1002, 1001, 1000]

  Compute Connection Resources

  Msg source: 1004
    nChannels To be Requested: 3

  Check Resources For NextHop.
  Constraints: QoS: 0.01; next hop: 0

 - temp FT size: 1
 - FT size: 3

TEMP FT Look-up:
  Next hop: 3; QoS: 0.01; Label: 12; nChannels: 3

FT Look-up:
  Next hop: 0; QoS: -1.0; Label: -100; nChannels: 32
  Next hop: 2; QoS: -1.0; Label: -100; nChannels: 32
  Next hop: 3; QoS: -1.0; Label: -100; nChannels: 32

  At GMPLS transmitter: GMPLS TO GMPLS
  Prop. delay is: 0.3333333
  signaling_msg ArrivalTime: 12.059943
  sig. dest: 1000 node: 1000
  action To Take: receiveSignalingMessage
```

Figure 19. Simulation Demo – Path messages switching

Receiving Signaling Path Messages

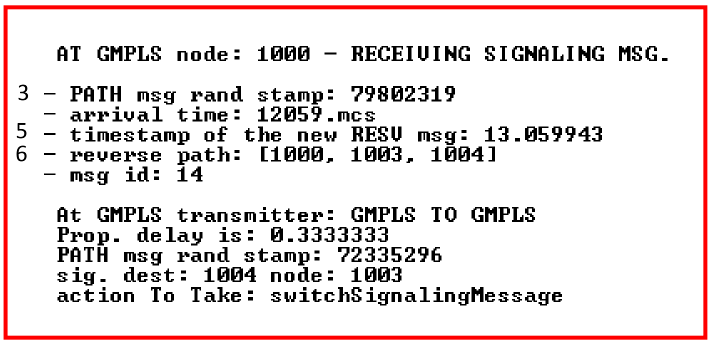
On *Fig.* 20 it is shown the receiving process of the first signaling *Path* message by the destination node 1000. Following the RSVP-TE signaling process protocol, the destination builds a new signaling message *Resv* and sends it back as an acknowledgement to the source node with ID 1004, confirming the correctly LSP establishment.

Continuing with our example, at this moment, one of the *Path* messages is about to accomplish its journey, whereas the other one still needs to be switched one more time until it gets its destination node (index 1000).

On this second phase of the signaling process, the GMPLS destination node 1000 has just received the first *Path* message corresponding to the signaling path [1004, 1003, 1000], with an arrival time of *12059 µseg* (line 3).

The new signaling message (*Resv* message) is created and is sent to the following node (1003) of the reverse path (line 5) with a LSP identifier label 14 (line 6).

]



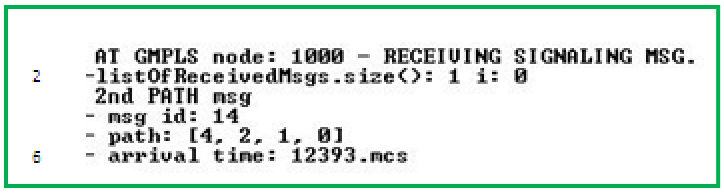
```
  AT GMPLS node: 1000 - RECEIVING SIGNALING MSG.
3 - PATH msg rand stamp: 79802319
  - arrival time: 12059.mcs
5 - timestamp of the new RESV msg: 13.059943
6 - reverse path: [1000, 1003, 1004]
  - msg id: 14

  At GMPLS transmitter: GMPLS TO GMPLS
  Prop. delay is: 0.3333333
  PATH msg rand stamp: 72335296
  sig. dest: 1004 node: 1003
  action To Take: switchSignalingMessage
```

Figure 20. Simulation Demo – Path message receiving

Second path Message Drooped

This sector shows the dropping action taken by the destination node, when it receives the last *Path* message. Since the destination node has already received the first *path* message, the second one will be immediately dropped.

Figure *21 shows* the time that the second *Path* message reaches the destination with an arrival time (line 6) greater than the one that belongs to the first *Path* message (line 3 of *Fig. 20*). Like it is represented on line 2, the list of received messages has already one entry that refers to the first *Path* message, leading this second one to its drop. Therefore the LSP is established through the signaling path [1004, 1003, 1000].



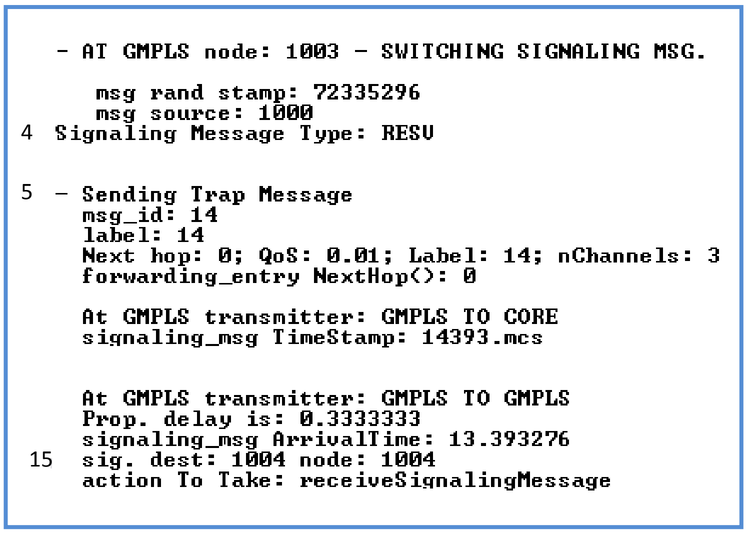
```
      AT GMPLS node: 1000 - RECEIVING SIGNALING MSG.
2    -listOfReceivedMsgs.size(): 1 i: 0
      2nd PATH msg
     - msg id: 14
     - path: [4, 2, 1, 0]
6    - arrival time: 12393.mcs
```

Figure 21. Simulation Demo – Path message dropping

Resv Message Switching

This sector refers to the *Resv* switching process until it reaches the original source node 1004, before it confirms the successfully LSP connection.

The node 1003 receives the *Resv* message (line 4), shown on *fig. 22* and right away it is switched again to the last node (line 15). On the same time, a new message (line 5) is created (Trap Message) and sent to the core node of the data layer with index 3, where it gets the LSP label, the next hop of the explicit path and other information to be retained on its forwarding table.



```
   - AT GMPLS node: 1003 - SWITCHING SIGNALING MSG.

      msg rand stamp: 72335296
      msg source: 1000
4  Signaling Message Type: RESV

5  - Sending Trap Message
     msg_id: 14
     label: 14
     Next hop: 0; QoS: 0.01; Label: 14; nChannels: 3
     forwarding_entry NextHop(): 0

     At GMPLS transmitter: GMPLS TO CORE
     signaling_msg TimeStamp: 14393.mcs


     At GMPLS transmitter: GMPLS TO GMPLS
     Prop. delay is: 0.3333333
     signaling_msg ArrivalTime: 13.393276
 15  sig. dest: 1004 node: 1004
     action To Take: receiveSignalingMessage
```

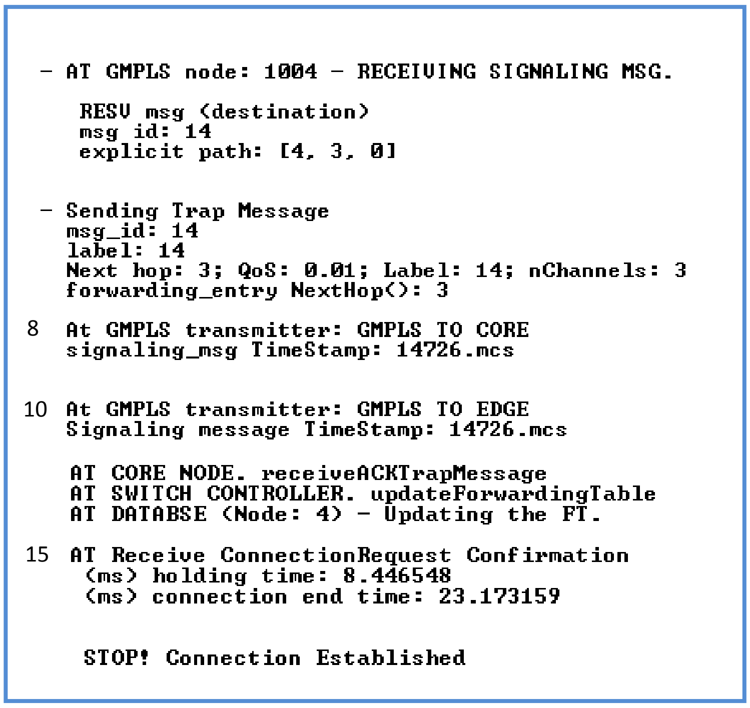
Figure 22. Simulation Demo – Resv message switching

Receiving Resv Message – Connection Establishment

Figure 23 shows the last step of the RSVP-TE protocol signaling process between the nodes 1004 and 1000, corresponding to the data nodes 4 and 0, respectively.

The source node 1004 receives the *Resv* message from the node 1003 and the same process of *Trap messages* exchange that was done on the node before is repeated. As an acknowledgment to the received *Trap message,* an *ACK Trap Message* is created and sent to the corresponding physical node (line 8) on the data layer with all the information about the

LSP. Meanwhile, the GMPLS control node 1004 confirms to its corresponding OBS edge node (line 10) that the connection was finally successfully established (line 15), finishing the signaling process for that LSP.



```
 - AT GMPLS node: 1004 - RECEIVING SIGNALING MSG.

    RESV msg (destination)
    msg id: 14
    explicit path: [4, 3, 0]


 - Sending Trap Message
   msg_id: 14
   label: 14
   Next hop: 3; QoS: 0.01; Label: 14; nChannels: 3
   forwarding_entry NextHop(): 3

8  At GMPLS transmitter: GMPLS TO CORE
   signaling_msg TimeStamp: 14726.mcs


10 At GMPLS transmitter: GMPLS TO EDGE
   Signaling message TimeStamp: 14726.mcs

    AT CORE NODE. receiveACKTrapMessage
    AT SWITCH CONTROLLER. updateForwardingTable
    AT DATABSE (Node: 4) - Updating the FT.

15  AT Receive ConnectionRequest Confirmation
     (ms) holding time: 8.446548
     (ms) connection end time: 23.173159


     STOP! Connection Established
```

Figure 23. Simulation Demo – Established Path Connection

Chapter V

Conclusions and Future Work

In this project was discussed the signaling and control aspects related to the GMPLS-based control plane interaction with the optical burst switching (OBS) control plane network.

The GMPLS control plane technology is currently accepted as a unifying multiprotocol for deploying IP over DWDM networks. Its big advantage is that it is already based on existing and widely organized protocols which simplify network management and engineering tasks that can be performed in an integrated way in both the data and the optical domains.

Although GMPLS control plane is a very extensive and detailed technology to cover and be implemented, generally all the proposed objectives were accomplished. The separation and interaction between both GMPLS and OBS control layers was done, as well as the basic features implementation of the GMPLS signaling protocol, RSVP-TE.

Future Work

As a future work, our objective is focusing on the rest of the GMPLS control plane protocols, including some RSVP-TE functions that were not yet implemented (e.g. LSP teardown), and trying to implement them on our unique simulator that can test the behaviour of the OBS networks managed by the GMPLS-based control plane. It will also be included the implementation of the traffic engineering mechanisms that will permit the management of alternative paths for the bursts, avoiding their lost and to improve the quality of service in the optical networks traffic transmissions.

Appendixes

Appendix A – JA-GOBS Configuration classes

OBS_Simulator

The *Obs_Simulator* is the main class of the JA-GOBS simulator. In it all the nodes and the links of the two networks are initialized and stored in the main *Database*.

Configuration Features

Load – A load with value 0.5 means that half of the resources are occupied.

Number of Channels (Wavelengths) – The total number of wavelengths available at the beginning before any distribution is set to 32.

Bursts – The burst's length is variable, in this demonstration the used value was set to 0.278 x 1E6.

Wavelength Capacity – The capacity of each wavelength in the OBS network is 10Gbit/s.

Steps – The simulation depends on the number of chosen steps at the beginning. Each step has duration of 30µs.

Matrix-type – In this simulation is used a uniform traffic matrix, means that the amount of load sent is equal for all nodes.

ProcessingTime - Time that takes to process each packet.

tOXC – Time that takes to switch a packet.

Main Methods

initNetwork - Load the nodes / links of the two networks (Data, Control) from the XML file.

initNodes – Initialization of the nodes network , the edge nodes, the core nodes and the GMPLS control nodes.

initLinks – Initialization of all links from each network and also the link connections between both networks.

ObsSimulationDataManager

The class *ObsSimulationDataManager* has the role to collect every information about the received- and dropped bursts during each step. At the end of the simulation the *ObsSimulationDataManager* displays the statistic of the entire simulation.

Main Methods

processData – This method processes the data of every burst transmission between the nodes in the OBS network.

getBLP – This method recovers the QoS pretended for the clients traffic transmission in the OBS network.

getTotalBurstDropped – This method displays at the end the number of lost bursts during the simulation.

Obs_Sim_Generator

This class, like the name says, is the generator of client's traffic to be sent. It has an interaction with the edge node's control unit, the class *VirtualAggregatorController* where it is sent the generated High Priority or Best Effort traffic (bursts).

Main Methods

setLabelMachine - Method that sets the labels for the sent traffic through the tunnels.

generateHPBETraffic – Method that generates High Priority (HP) or Best Effort (BE) traffic to be sent to the OBS nodes. The HP traffic has priority over the BE traffic on the resources reservation.

forcedTrafficPeak – A method that forces the sending of traffic peaks. At the moment this option is disabled since we are not simulating traffic peaks.

Appendix B – OBS Data/Control plane main classes

Obs_Core_Node

The class *Obs_Core_Node* represents the nodes in the OBS network. As explained before, in the simulation an OBS node is represented by two classes the *Obs_Edge_Node* and the *Obs_Core_Node*. This class in the Data Plane has the objective to receive or switch the packet / burst to the next node of the path though its control unit, the *SwitchController* class.

When used to interact to the Control Plane, the *Obs_Core_Node* objective is to send to the control node the links state connected to it.

Main Methods

implReset - This method is called through the *SwitchController* class, setups the node with the outgoing links and also the link connection to the corresponding GMPLS control node.

receiveBurst – This method receives the BCP/burst from the stack of events.

switchBurst – This method has the objective to switch the burst for the next node of the current path after the resources reservation has been successful accomplished.

receiveACKTrapMessage – This method receives the acknowledge from the GMPLS control node where is sent the label of the established tunnel, time indicators, message ID among other information fields.

SwitchController

The *SwitchController* class is the control unit for every single core node. This class has the function to call the reservation protocol (JIT) to reserve the resources for the respective burst.

Besides that it also communicates to the GMPLS node, where it sends Trap messages indicating the state of each link connected to the core node.

Main Methods

checkInternalResourcesAvailabilityAndTransmit – Method that calls the reservation protocol JIT represented by the class *Just_In_Time* to compute the reservation of the available resources for the transmission. If it receives a NACK as an answer, then the packet is dropped, otherwise, is reserved successfully and the burst will not be lost.

contentionWindow / occupancyWindow - These methods tell us the right time that the load should be distributed to another link, avoiding the overload in the tunnel.

sendTrapMessage - This method interacts with the corresponding GMPLS control node where it sends the links state connected to the core node in the OBS network. Then waits for an ACK trap message where it receives the label of the established tunnel and the number of wavelengths necessary to the burst reservation.

Obs_Edge_Node

The class *Obs_Edge_Node* is one of the two classes, which represents the OBS node in the real view. This class refers to the source node that wants to transmit traffic over the OBS network. However it also receives the confirmation (positive or negative) of the established path from the corresponding GMPLS control node.

It has as its control unit, the class *VirtualAggregatorController* that receives the HP or BE traffic from the *Obs_Sim_Generator* to be transmitted through the optical path.

Main Method

receiveConnectionRequestConfirmation – This method receives the confirmation from the GMPLS control node, if the tunnel was successful or unsuccessful established.

VirtualAggregatorController

The class *VirtualAggregatorController* is the control unit of each edge node that sends traffic (bursts) to the destination nodes within the OBS network. Besides that, it also interacts with the GMPLS control plane by sending a connection request for the establishment of a tunnel.

Main Methods

transmitEvent - Method used to transmit the BCP to the following core node through the stack of events.

getOffsetTime - Method that gets the offset time for the waiting burst.

sendConnectionRequest - Method used to send a tunnel connection request to the control layer when there is not any established tunnel in the data layer. As acknowledge it might receives the confirmation that the tunnel is established or an error message that at some point of the path it was not possible to make the reservation.

ExplicitRoutingLogic

The class *ExplicitRoutingLogic* implements and follows the rules of the OSPF protocol by routing the data through the tunnels and other functions like the setup of all tunnels and the assignment of the number of LSPs per tunnel. In other words, every time a node needs to send the data burst to a neighbor node, as soon as the resources reservation is done, the properly methods from the *ExplicitRoutingLogic* class are called to give the information needed from the tunnel and the output link selected, so the data burst can be routed.

Main Methods

getNextHop / getNextHopSP - Methods used to get the next hop of a given optical path to flew the bursts until the destination node. The *getNextHop* is used for burst with HP and the other is for BE.

wavelengthAssignment - This method as the role to assign the number of wavelengths (resources) for each Label Switching Path (LSP).

tunnelSetup - Important method that establishes the tunnel from a source node to the given destination node.

Appendix C - GMPLS Control Plane main class

GMPLSController

The class *GMPLSController* represents the GMPLS control nodes of the control plane network. Therefore, every time is requested the establishment of a new path, this class takes into action by executing all the processes needed to a correctly functionality of the GMPLS signaling protocol, RSVP-TE.

Main Methods

sendPATHMessage/ sendRESVMessage – These are the two methods that perform the exchange of the two signaling messages Path and Resvof the RSVP-TE protocol.

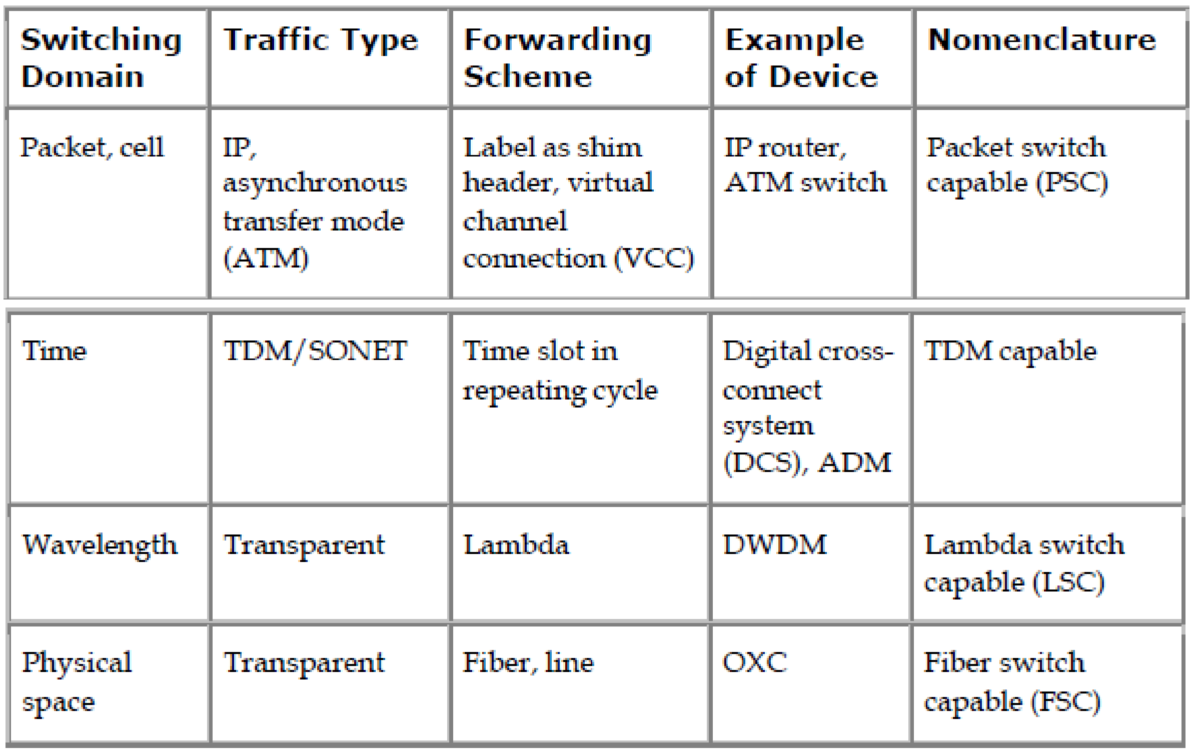
computeConnection – This method decides if is possible to make the reservation of the available resources for the establishment of the optical path between two control nodes.

LSPLabelGenerator - Generates a new label to indentify each established tunnel.

checkTrafficIncrement / Decrement – Those two methods allow to check the traffic that was incremented or decremented inside a link shared by two different tunnels.

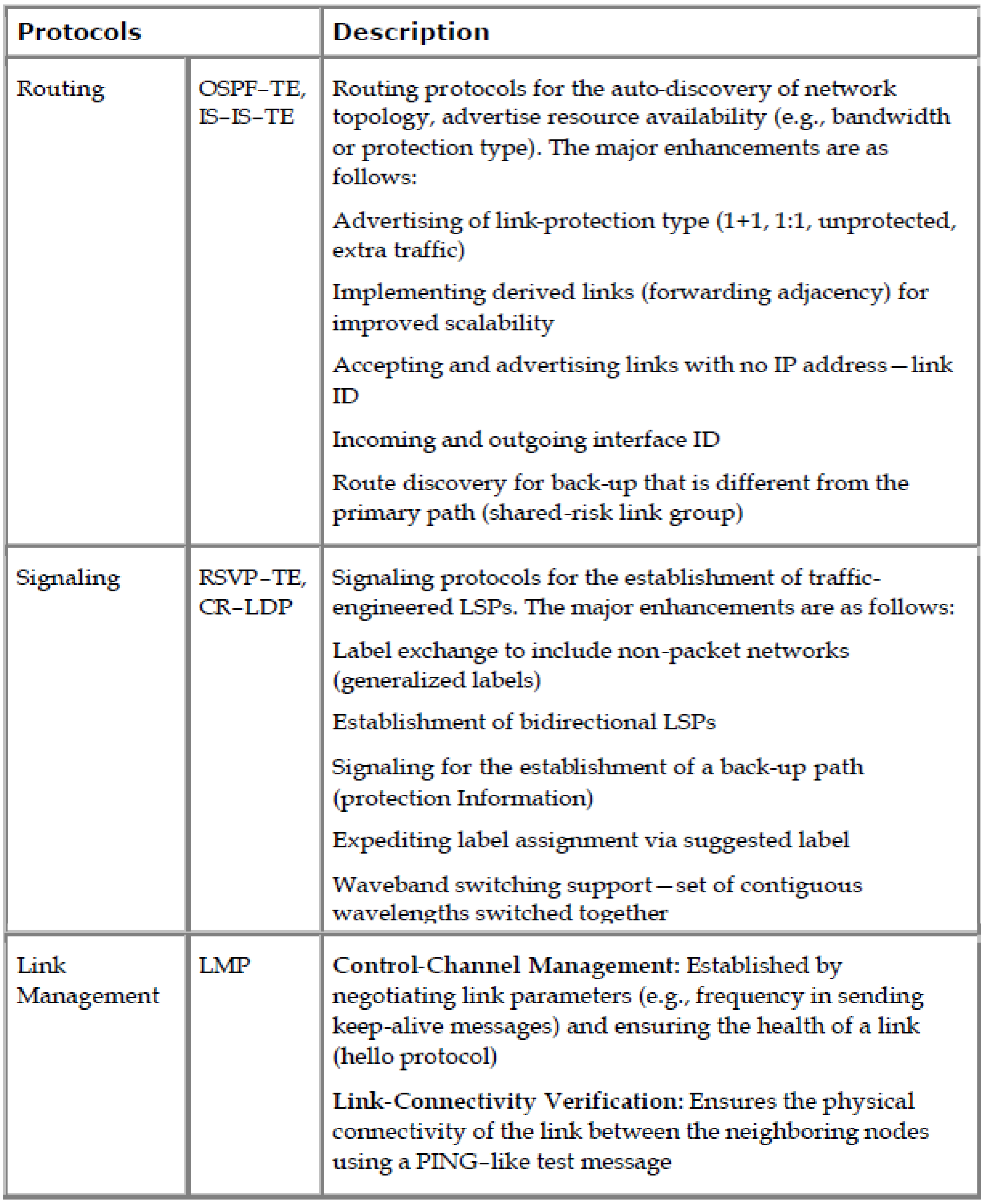
Appendix D – GMPLS Framework and Control Plane Protocols [31]

GMPLS Framework



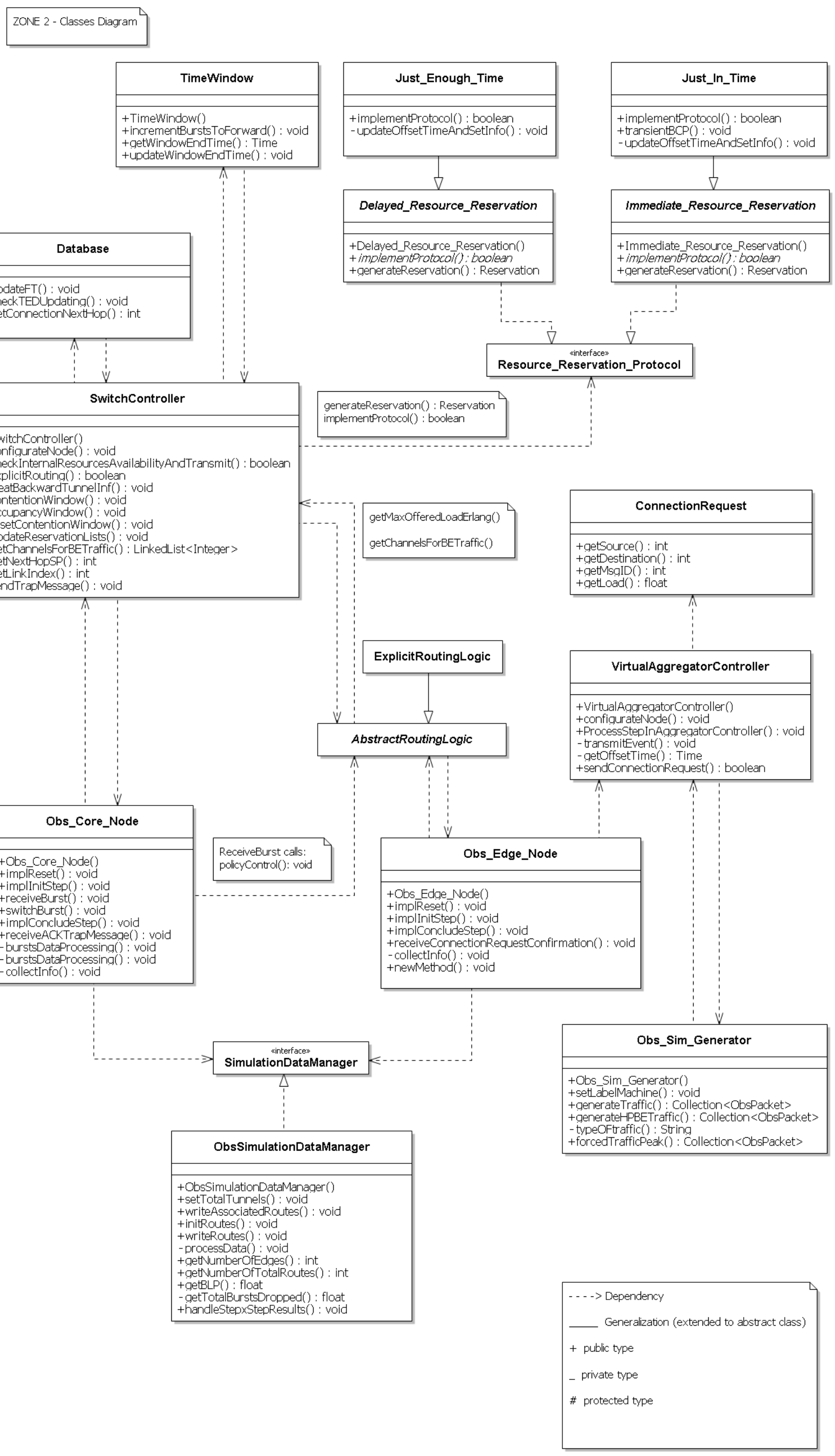
Switching Domain	Traffic Type	Forwarding Scheme	Example of Device	Nomenclature
Packet, cell	IP, asynchronous transfer mode (ATM)	Label as shim header, virtual channel connection (VCC)	IP router, ATM switch	Packet switch capable (PSC)
Time	TDM/SONET	Time slot in repeating cycle	Digital cross-connect system (DCS), ADM	TDM capable
Wavelength	Transparent	Lambda	DWDM	Lambda switch capable (LSC)
Physical space	Transparent	Fiber, line	OXC	Fiber switch capable (FSC)

GMPLS Control Plane Protocols



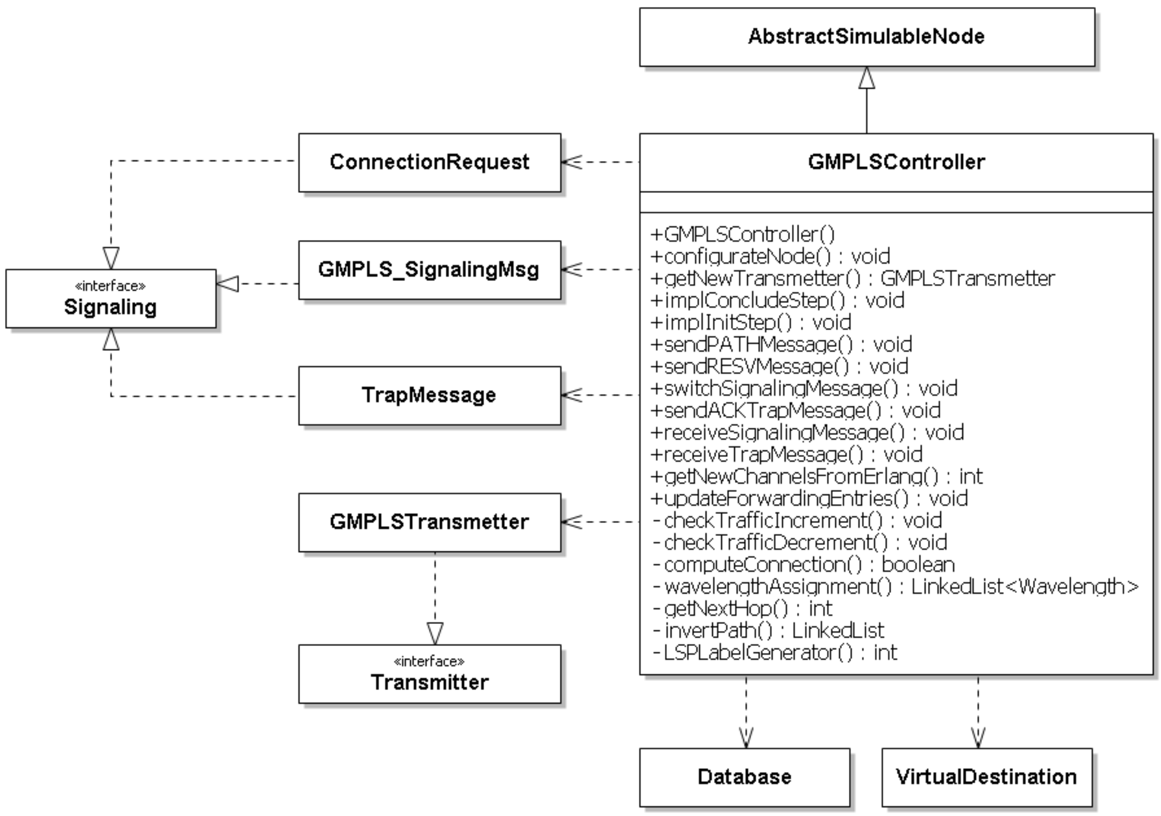
Protocols		Description
Routing	OSPF–TE, IS–IS–TE	Routing protocols for the auto-discovery of network topology, advertise resource availability (e.g., bandwidth or protection type). The major enhancements are as follows: Advertising of link-protection type (1+1, 1:1, unprotected, extra traffic) Implementing derived links (forwarding adjacency) for improved scalability Accepting and advertising links with no IP address—link ID Incoming and outgoing interface ID Route discovery for back-up that is different from the primary path (shared-risk link group)
Signaling	RSVP–TE, CR–LDP	Signaling protocols for the establishment of traffic-engineered LSPs. The major enhancements are as follows: Label exchange to include non-packet networks (generalized labels) Establishment of bidirectional LSPs Signaling for the establishment of a back-up path (protection Information) Expediting label assignment via suggested label Waveband switching support—set of contiguous wavelengths switched together
Link Management	LMP	**Control-Channel Management:** Established by negotiating link parameters (e.g., frequency in sending keep-alive messages) and ensuring the health of a link (hello protocol) **Link-Connectivity Verification:** Ensures the physical connectivity of the link between the neighboring nodes using a PING–like test message

Appendix E – OBS Network Classes Diagram

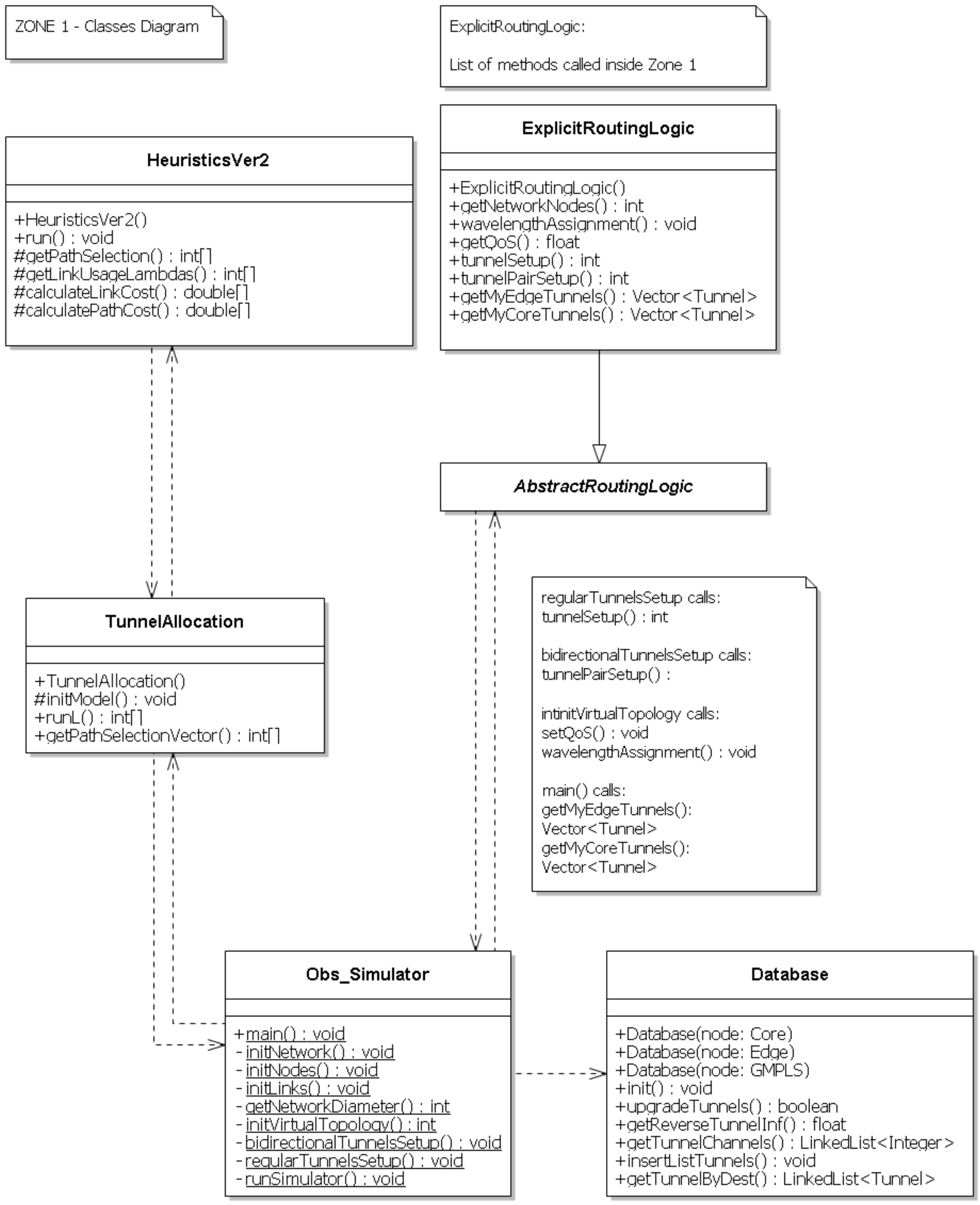


ZONE 2 - Classes Diagram
TimeWindow
+TimeWindow()
+incrementBurstsToForward() : void
+getWindowEndTime() : Time
+updateWindowEndTime() : void
Just_Enough_Time
+implementProtocol() : boolean
-updateOffsetTimeAndSetInfo() : void
Just_In_Time
+implementProtocol() : boolean
+transientBCP() : void
-updateOffsetTimeAndSetInfo() : void
Delayed_Resource_Reservation
+Delayed_Resource_Reservation()
+implementProtocol() : boolean
+generateReservation() : Reservation
Immediate_Resource_Reservation
+Immediate_Resource_Reservation()
+implementProtocol() : boolean
+generateReservation() : Reservation
Database
+updateFT() : void
+checkTEDUpdating() : void
+getConnectionNextHop() : int
«interface»
Resource_Reservation_Protocol
generateReservation() : Reservation
implementProtocol() : boolean
SwitchController
+SwitchController()
+configurateNode() : void
+checkInternalResourcesAvailabilityAndTransmit() : boolean
-explicitRouting() : boolean
+treatBackwardTunnelInf() : void
+contentionWindow() : void
+occupancyWindow() : void
+resetContentionWindow() : void
+updateReservationLists() : void
+getChannelsForBETraffic() : LinkedList<Integer>
+getNextHopSP() : int
+getLinkIndex() : int
+sendTrapMessage() : void
getMaxOfferedLoadErlang()
getChannelsForBETraffic()
ConnectionRequest
+getSource() : int
+getDestination() : int
+getMsgID() : int
+getLoad() : float
ExplicitRoutingLogic
AbstractRoutingLogic
VirtualAggregatorController
+VirtualAggregatorController()
+configurateNode() : void
+ProcessStepInAggregatorController() : void
-transmitEvent() : void
-getOffsetTime() : Time
+sendConnectionRequest() : boolean
Obs_Core_Node
+Obs_Core_Node()
+implReset() : void
+implInitStep() : void
+receiveBurst() : void
+switchBurst() : void
+implConcludeStep() : void
+receiveACKTrapMessage() : void
-burstsDataProcessing() : void
-burstsDataProcessing() : void
-collectInfo() : void
ReceiveBurst calls:
policyControl(): void
Obs_Edge_Node
+Obs_Edge_Node()
+implReset() : void
+implInitStep() : void
+implConcludeStep() : void
+receiveConnectionRequestConfirmation() : void
-collectInfo() : void
+newMethod() : void
Obs_Sim_Generator
+Obs_Sim_Generator()
+setLabelMachine() : void
+generateTraffic() : Collection<ObsPacket>
+generateHPBETraffic() : Collection<ObsPacket>
-typeOFtraffic() : String
+forcedTrafficPeak() : Collection<ObsPacket>
«interface»
SimulationDataManager
ObsSimulationDataManager
+ObsSimulationDataManager()
+setTotalTunnels() : void
+writeAssociatedRoutes() : void
+initRoutes() : void
+writeRoutes() : void
-processData() : void
+getNumberOfEdges() : int
+getNumberOfTotalRoutes() : int
+getBLP() : float
-getTotalBurstsDropped() : float
+handleStepxStepResults() : void
- - - -> Dependency
_____ Generalization (extended to abstract class)
+ public type
_ private type
# protected type

Appendix F - GMPLS Network Classes Diagram


AbstractSimulableNode
ConnectionRequest
GMPLSController
«interface»
Signaling
GMPLS_SignalingMsg
TrapMessage
GMPLSTransmetter
«interface»
Transmitter
Database
VirtualDestination
+GMPLSController()
+configurateNode() : void
+getNewTransmetter() : GMPLSTransmetter
+implConcludeStep() : void
+implInitStep() : void
+sendPATHMessage() : void
+sendRESVMessage() : void
+switchSignalingMessage() : void
+sendACKTrapMessage() : void
+receiveSignalingMessage() : void
+receiveTrapMessage() : void
+getNewChannelsFromErlang() : int
+updateForwardingEntries() : void
-checkTrafficIncrement() : void
-checkTrafficDecrement() : void
-computeConnection() : boolean
-wavelengthAssignment() : LinkedList<Wavelength>
-getNextHop() : int
-invertPath() : LinkedList
-LSPLabelGenerator() : int

Appendix G - JAVOBS Configuration Classes Diagram



ZONE 1 - Classes Diagram
ExplicitRoutingLogic:
List of methods called inside Zone 1
ExplicitRoutingLogic
+ExplicitRoutingLogic()
+getNetworkNodes() : int
+wavelengthAssignment() : void
+getQoS() : float
+tunnelSetup() : int
+tunnelPairSetup() : int
+getMyEdgeTunnels() : Vector<Tunnel>
+getMyCoreTunnels() : Vector<Tunnel>
HeuristicsVer2
+HeuristicsVer2()
+run() : void
#getPathSelection() : int[]
#getLinkUsageLambdas() : int[]
#calculateLinkCost() : double[]
#calculatePathCost() : double[]
AbstractRoutingLogic
TunnelAllocation
+TunnelAllocation()
#initModel() : void
+runL() : int[]
+getPathSelectionVector() : int[]
regularTunnelsSetup calls:
tunnelSetup() : int
bidirectionalTunnelsSetup calls:
tunnelPairSetup() :
intinitVirtualTopology calls:
setQoS() : void
wavelengthAssignment() : void
main() calls:
getMyEdgeTunnels():
Vector<Tunnel>
getMyCoreTunnels():
Vector<Tunnel>
Obs_Simulator
+main() : void
- initNetwork() : void
- initNodes() : void
- initLinks() : void
- getNetworkDiameter() : int
- initVirtualTopology() : int
- bidirectionalTunnelsSetup() : void
- regularTunnelsSetup() : void
- runSimulator() : void
Database
+Database(node: Core)
+Database(node: Edge)
+Database(node: GMPLS)
+init() : void
+upgradeTunnels() : boolean
+getReverseTunnelInf() : float
+getTunnelChannels() : LinkedList<Integer>
+insertListTunnels() : void
+getTunnelByDest() : LinkedList<Tunnel>

Appendix H - Erlang B Traffic Table

$$E_{1,N}(A) = B_N = \frac{\frac{A^N}{N!}}{\sum_{i=0}^{N} \frac{A^i}{i!}},$$

where B is loss probability,
A offered traffic,
N number of circuits.

Erlang B Traffic Table

Maximum Offered Load Versus B and N
B is in %

N/B	0.01	0.05	0.1	0.5	1.0	2	5	10	15	20	30	40
1	.0001	.0005	.0010	.0050	.0101	.0204	.0526	.1111	.1765	.2500	.4286	.6667
2	.0142	.0321	.0458	.1054	.1526	.2235	.3813	.5954	.7962	1.000	1.449	2.000
3	.0868	.1517	.1938	.3490	.4555	.6022	.8994	1.271	1.603	1.930	2.633	3.480
4	.2347	.3624	.4393	.7012	.8694	1.092	1.525	2.045	2.501	2.945	3.891	5.021
5	.4520	.6486	.7621	1.132	1.361	1.657	2.219	2.881	3.454	4.010	5.189	6.596
6	.7282	.9957	1.146	1.622	1.909	2.276	2.960	3.758	4.445	5.109	6.514	8.191
7	1.054	1.392	1.579	2.158	2.501	2.935	3.738	4.666	5.461	6.230	7.856	9.800
8	1.422	1.830	2.051	2.730	3.128	3.627	4.543	5.597	6.498	7.369	9.213	11.42
9	1.826	2.302	2.558	3.333	3.783	4.345	5.370	6.546	7.551	8.522	10.58	13.05
10	2.260	2.803	3.092	3.961	4.461	5.084	6.216	7.511	8.616	9.685	11.95	14.68
11	2.722	3.329	3.651	4.610	5.160	5.842	7.076	8.487	9.691	10.86	13.33	16.31
12	3.207	3.878	4.231	5.279	5.876	6.615	7.950	9.474	10.78	12.04	14.72	17.95
13	3.713	4.447	4.831	5.964	6.607	7.402	8.835	10.47	11.87	13.22	16.11	19.60
14	4.239	5.032	5.446	6.663	7.352	8.200	9.730	11.47	12.97	14.41	17.50	21.24
15	4.781	5.634	6.077	7.376	8.108	9.010	10.63	12.48	14.07	15.61	18.90	22.89
16	5.339	6.250	6.722	8.100	8.875	9.828	11.54	13.50	15.18	16.81	20.30	24.54
17	5.911	6.878	7.378	8.834	9.652	10.66	12.46	14.52	16.29	18.01	21.70	26.19
18	6.496	7.519	8.046	9.578	10.44	11.49	13.39	15.55	17.41	19.22	23.10	27.84
19	7.093	8.170	8.724	10.33	11.23	12.33	14.32	16.58	18.53	20.42	24.51	29.50
20	7.701	8.831	9.412	11.09	12.03	13.18	15.25	17.61	19.65	21.64	25.92	31.15
21	8.319	9.501	10.11	11.86	12.84	14.04	16.19	18.65	20.77	22.85	27.33	32.81
22	8.946	10.18	10.81	12.64	13.65	14.90	17.13	19.69	21.90	24.06	28.74	34.46
23	9.583	10.87	11.52	13.42	14.47	15.76	18.08	20.74	23.03	25.28	30.15	36.12
24	10.23	11.56	12.24	14.20	15.30	16.63	19.03	21.78	24.16	26.50	31.56	37.78
25	10.88	12.26	12.97	15.00	16.13	17.51	19.99	22.83	25.30	27.72	32.97	39.44
26	11.54	12.97	13.70	15.80	16.96	18.38	20.94	23.89	26.43	28.94	34.39	41.10
27	12.21	13.69	14.44	16.60	17.80	19.27	21.90	24.94	27.57	30.16	35.80	42.76
28	12.88	14.41	15.18	17.41	18.64	20.15	22.87	26.00	28.71	31.39	37.21	44.41
29	13.56	15.13	15.93	18.22	19.49	21.04	23.83	27.05	29.85	32.61	38.63	46.07
30	14.25	15.86	16.68	19.03	20.34	21.93	24.80	28.11	31.00	33.84	40.05	47.74
31	14.94	16.60	17.44	19.85	21.19	22.83	25.77	29.17	32.14	35.07	41.46	49.40
32	15.63	17.34	18.21	20.68	22.05	23.73	26.75	30.24	33.28	36.30	42.88	51.06
33	16.34	18.09	18.97	21.51	22.91	24.63	27.72	31.30	34.43	37.52	44.30	52.72
34	17.04	18.84	19.74	22.34	23.77	25.53	28.70	32.37	35.58	38.75	45.72	54.38
35	17.75	19.59	20.52	23.17	24.64	26.44	29.68	33.43	36.72	39.99	47.14	56.04
36	18.47	20.35	21.30	24.01	25.51	27.34	30.66	34.50	37.87	41.22	48.56	57.70
37	19.19	21.11	22.08	24.85	26.38	28.25	31.64	35.57	39.02	42.45	49.98	59.37
38	19.91	21.87	22.86	25.69	27.25	29.17	32.62	36.64	40.17	43.68	51.40	61.03
39	20.64	22.64	23.65	26.53	28.13	30.08	33.61	37.72	41.32	44.91	52.82	62.69
40	21.37	23.41	24.44	27.38	29.01	31.00	34.60	38.79	42.48	46.15	54.24	64.35
41	22.11	24.19	25.24	28.23	29.89	31.92	35.58	39.86	43.63	47.38	55.66	66.02
42	22.85	24.97	26.04	29.09	30.77	32.84	36.57	40.94	44.78	48.62	57.08	67.68
43	23.59	25.75	26.84	29.94	31.66	33.76	37.57	42.01	45.94	49.85	58.50	69.34

44	24.33	26.53	27.64	30.80	32.54	34.68	38.56	43.09	47.09	51.09	59.92	71.01
45	25.08	27.32	28.45	31.66	33.43	35.61	39.55	44.17	48.25	52.32	61.35	72.67
46	25.83	28.11	29.26	32.52	34.32	36.53	40.55	45.24	49.40	53.56	62.77	74.33
47	26.59	28.90	30.07	33.38	35.22	37.46	41.54	46.32	50.56	54.80	64.19	76.00
48	27.34	29.70	30.88	34.25	36.11	38.39	42.54	47.40	51.71	56.03	65.61	77.66
49	28.10	30.49	31.69	35.11	37.00	39.32	43.53	48.48	52.87	57.27	67.04	79.32
50	28.87	31.29	32.51	35.98	37.90	40.26	44.53	49.56	54.03	58.51	68.46	80.99
51	29.63	32.09	33.33	36.85	38.80	41.19	45.53	50.64	55.19	59.75	69.88	82.65
52	30.40	32.90	34.15	37.72	39.70	42.12	46.53	51.73	56.35	60.99	71.31	84.32
53	31.17	33.70	34.98	38.60	40.60	43.06	47.53	52.81	57.50	62.22	72.73	85.98
54	31.94	34.51	35.80	39.47	41.51	44.00	48.54	53.89	58.66	63.46	74.15	87.65
55	32.72	35.32	36.63	40.35	42.41	44.94	49.54	54.98	59.82	64.70	75.58	89.31
56	33.49	36.13	37.46	41.23	43.32	45.88	50.54	56.06	60.98	65.94	77.00	90.97
57	34.27	36.95	38.29	42.11	44.22	46.82	51.55	57.14	62.14	67.18	78.43	92.64
58	35.05	37.76	39.12	42.99	45.13	47.76	52.55	58.23	63.31	68.42	79.85	94.30
59	35.84	38.58	39.96	43.87	46.04	48.70	53.56	59.32	64.47	69.66	81.27	95.97
60	36.62	39.40	40.80	44.76	46.95	49.64	54.57	60.40	65.63	70.90	82.70	97.63
61	37.41	40.22	41.63	45.64	47.86	50.59	55.57	61.49	66.79	72.14	84.12	99.30
62	38.20	41.05	42.47	46.53	48.77	51.53	56.58	62.58	67.95	73.38	85.55	101.0
63	38.99	41.87	43.31	47.42	49.69	52.48	57.59	63.66	69.11	74.63	86.97	102.6
64	39.78	42.70	44.16	48.31	50.60	53.43	58.60	64.75	70.28	75.87	88.40	104.3
65	40.58	43.52	45.00	49.20	51.52	54.38	59.61	65.84	71.44	77.11	89.82	106.0
66	41.38	44.35	45.85	50.09	52.44	55.33	60.62	66.93	72.60	78.35	91.25	107.6
67	42.17	45.18	46.69	50.98	53.35	56.28	61.63	68.02	73.77	79.59	92.67	109.3
68	42.97	46.02	47.54	51.87	54.27	57.23	62.64	69.11	74.93	80.83	94.10	111.0
69	43.77	46.85	48.39	52.77	55.19	58.18	63.65	70.20	76.09	82.08	95.52	112.6
70	44.58	47.68	49.24	53.66	56.11	59.13	64.67	71.29	77.26	83.32	96.95	114.3
71	45.38	48.52	50.09	54.56	57.03	60.08	65.68	72.38	78.42	84.56	98.37	116.0
72	46.19	49.36	50.94	55.46	57.96	61.04	66.69	73.47	79.59	85.80	99.80	117.6
73	47.00	50.20	51.80	56.35	58.88	61.99	67.71	74.56	80.75	87.05	101.2	119.3
74	47.81	51.04	52.65	57.25	59.80	62.95	68.72	75.65	81.92	88.29	102.7	120.9
75	48.62	51.88	53.51	58.15	60.73	63.90	69.74	76.74	83.08	89.53	104.1	122.6
76	49.43	52.72	54.37	59.05	61.65	64.86	70.75	77.83	84.25	90.78	105.5	124.3
77	50.24	53.56	55.23	59.96	62.58	65.81	71.77	78.93	85.41	92.02	106.9	125.9
78	51.05	54.41	56.09	60.86	63.51	66.77	72.79	80.02	86.58	93.26	108.4	127.6
79	51.87	55.25	56.95	61.76	64.43	67.73	73.80	81.11	87.74	94.51	109.8	129.3
80	52.69	56.10	57.81	62.67	65.36	68.69	74.82	82.20	88.91	95.75	111.2	130.9
81	53.51	56.95	58.67	63.57	66.29	69.65	75.84	83.30	90.08	96.99	112.6	132.6
82	54.33	57.80	59.54	64.48	67.22	70.61	76.86	84.39	91.24	98.24	114.1	134.3
83	55.15	58.65	60.40	65.39	68.15	71.57	77.87	85.48	92.41	99.48	115.5	135.9
84	55.97	59.50	61.27	66.29	69.08	72.53	78.89	86.58	93.58	100.7	116.9	137.6
85	56.79	60.35	62.14	67.20	70.02	73.49	79.91	87.67	94.74	102.0	118.3	139.3
86	57.62	61.21	63.00	68.11	70.95	74.45	80.93	88.77	95.91	103.2	119.8	140.9
87	58.44	62.06	63.87	69.02	71.88	75.42	81.95	89.86	97.08	104.5	121.2	142.6
88	59.27	62.92	64.74	69.93	72.82	76.38	82.97	90.96	98.25	105.7	122.6	144.3
89	60.10	63.77	65.61	70.84	73.75	77.34	83.99	92.05	99.41	107.0	124.0	145.9
90	60.92	64.63	66.48	71.76	74.68	78.31	85.01	93.15	100.6	108.2	125.5	147.6
91	61.75	65.49	67.36	72.67	75.62	79.27	86.04	94.24	101.8	109.4	126.9	149.3
92	62.58	66.35	68.23	73.58	76.56	80.24	87.06	95.34	102.9	110.7	128.3	150.9
93	63.42	67.21	69.10	74.50	77.49	81.20	88.08	96.43	104.1	111.9	129.8	152.6
94	64.25	68.07	69.98	75.41	78.43	82.17	89.10	97.53	105.3	113.2	131.2	154.3
95	65.08	68.93	70.85	76.33	79.37	83.13	90.12	98.63	106.4	114.4	132.6	155.9
96	65.92	69.79	71.73	77.24	80.31	84.10	91.15	99.72	107.6	115.7	134.0	157.6
97	66.75	70.65	72.61	78.16	81.25	85.07	92.17	100.8	108.8	116.9	135.5	159.3
98	67.59	71.52	73.48	79.07	82.18	86.04	93.19	101.9	109.9	118.2	136.9	160.9
99	68.43	72.38	74.36	79.99	83.12	87.00	94.22	103.0	111.1	119.4	138.3	162.6
100	69.27	7~.25	75.24	80.91	84.06	87.97	95.24	104.1	112.3	120.6	139.7	164.3

References

[1] A. Jajszczyk, “ The ASON approach to the control plane for optical networks”, International Conference on Transparent Optical Networks, IEEE ICTON 2003, Vol. 1, pp 87-90, Nov 2004.

[2] ITU-T Rec. G.8080/Y.1304, "Architecture for the Automatically Switched Optical Network (ASON)," November 2001 (and Revision, January 2003).

[3] Chunsheng Xin, Yinghua Ye, Ti-Shiang Wang, S. Dixit, Chunming Qiao, Myungsik Yoo, “On an IP-centric optical control plane”, IEEE Communications Magazine, Vol. 39, pp. 88-93, Sep. 2001.

[4] S. Sanchez-Lopez., J. Sole-Pareta, J. Comellas, J. Soldatos, G. Kylafas, M. Jaeger, “PNNI-based Control Plane for automatically switched optical networks”, Journal of Lightwave Technology, Vol. 21, pp. 2673 – 2682, Nov. 2003.

[5] J. Solé-Paretam, X. Masip-Bruin, S. Sànchez-López, S. Spadaro, D. Careglio, “Some Open Issues in the Optical Networks Control Plane”, in Proceedings of International Conference on Transparent Optical networks, IEEE ICTON 2003, Vol. 1, pp. 76-81, Fev. 2004.

[6] RFC 3945 – Generalized Multi-Protocol Label Switching (GMPLS) Architecture.

[7] P. Pedroso, J. Sole-Pareta, D. Careglio, M. Klinkowski, “Integrating GMPLS in the OBS networks control plane” ICTON, Vol.3, pp. 1-7, Aug. 2007.

[8] Oscar Pedrola, Josep Sole-Pareta, Davide Careglio, Miroslaw Klinkowski, “JAVOBS: A Flexible Simulator for OBS Network Architectures” , Journal of Networks. Vol.5, pp. 117-122, February 2010.

[9] Jue, J.P, Yang, W-H, Kim, Y-C, “Optical packet and *burst* switched networks: a review”, Communication, IET, Vol. 3, pp. 334-352, 2009.

[10] C. Qiao and M. Yoo, "Optical Burst Switching - A New Paradigm for an Optical Internet", Journal of High Speed Networks, Special Issue on Optical Networks, Vol. 8, pp.69-84, 1999.

[11] J. Buysse, M De Leeenheer, C. Develder, B Dhoedt, P. Demeester, “Cost-Effective Burst-over-Circuit-Switching in a Hybrid Optical Network”, Fifth International Conference on Networking and Services, ICNS ’09, pp. 499-504, Apr. 2009.

[12] S.J.B. Yoo, “Optical Packet and Burst Switching Technologies for the Future Photonic Internet”, Journal of Lightwave Technology, Vol. 24, pp. 4468-4492, Dec. 2006.

[13] M. A. Aydin, T. Atmaca, H. Zaim, O. C. Turna, V. H. Nguyen, "Performance Study of OBS Reservation Protocols," aict, Fourth Advanced International Conference on Telecommunications, pp.428-433, 2008.

[14] Joel J. P. C. Rodrigues and Binod Vaidya, “Evaluation of Resource Reservation Protocols for IP over OBS Networks”, 11th International Conference on Transparent Optical Networks (ICTON 2009), pp. 1- 4, Ponta Delgada, Açores, Portugal, June 28 – July 02, 2009 (Invited).

[15] J.Y. Wei, and R.I. McFarland Jr., "Just-in-time signaling for WDM optical burst switching networks", Journal of Lightwave Technology, Vol. 18, pp. 2019-2037, Dec 2000.

[16] Myungsik Yoo, Chunming Qiao, “Just-Enough-Time (JET): a high speed protocol for bursty traffic in optical networks.”, IEEE conferences, pp. 26-27, Aug. 1997.

[17] SuKyoung Lee, Sriram, K., HyunSook Kim, JooSeok Song, “Contention-based limited Deflection Routing Protocol in Optical Burst-Switched networks”, IEEE Journal on Selected Areas in Communications, Vol. 23, pp. 1596 – 1611, Aug. 2005.

[18] SuKyoung Lee, Sriram, K. HyunSook Kim, JooSeok Song, “Connection-based limited deflection routing in OBS networks“, GLOBECOM, Vol. 5, pp. 2633-2637, Dec. 2003.

[19] D. Colle, J. Cheyns, C. Develder, E. Van Breusegem, A. Ackaert, M. Pickavet, P. Demeester, “GMPLS Extensions for Supporting Advanced Optical Networking Technologies”, Proceedings of 2003 5th International Conference on Transparent Optical Networks, Vol. 1, pp. 170-173, Jun. 2003.

[20] RFC-3473, Generalized Multi-Protocol Label Switching. *(*GMPLS*)* Signaling Resource Reservation Protocol-Traffic. Engineering *(*RSVP-TE*)* Extensions.

[21] RFC 3472, GMPLS - Constraint-based Routed Label Distribution Protocol (CR-LDP) Extensions.

[22] Open Shortest Path First (OSPF) Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS).

[23] Intermediate System to Intermediate System (IS-IS) Extensions in support of Generalized Multi-Protocol Label Switching (GMPLS).

[24] Label Switched Paths (LSP) Hierarchy with Generalized Multi-Protocol Label Switching (GMPLS) Traffic Engineering (TE).

[25] RFC 3209 - RSVP-TE: Extensions to RSVP for LSP Tunnels.

[26] P. Rozycki, J. Korniak, “Influence of the Control Plane Architecture on QoS in the GMPLS Network”, Conference on Human System interactions, pp. 1033-1037, May 2008.

[27] Jordi Perelló, Luis Velasco, Fernando Agraz, Salvatore Spadaro, Gabriel Junyent, Jaume Comellas “A Comparison of In-Fiber and Out-Of-Fiber GMPLS-Based Control Plane Configurations: Benefits, Drawbacks and Solutions”, ICTON, pp. 101-104, 2008.

[28] Pedro Pedroso, “Interoperable GMPLS/OBS Control Plane: Functional Architecture and Protocol Extensions Proposal”, Pedro Pedroso Master’s dissertation, 2008.

[29] P. Pedroso, D. Careglio, R. Casellas, M. Klinkowski, and J. Solé-Pareta, "An interoperable GMPLS/OBS Control Plane: RSVP and OSPF extensions proposal", in *Proceedings of the 6th International Symposion on Communication Systems, Networks and Digital Signal Processing* (CSNDSP08), pp. 418-422, Graz, Austria, July, 2008.

[30] J.Triay, J. Rubio, C. Cervelló-Pastor, “An Optical Burst Switching Control Plane Architecture and its Implementation”, Proceedings of the 12th Open European Summer School EUNICE 2006. Stuttgart, pp. 1-7, Sep. 2006.

[31] [Online] Generalized Multiprotocol Label Switching (GMPLS), *International Engineering*Consortium, pp. 1-25, www.iec.org.