ISCTE IUL
Instituto Universitário de Lisboa

Departamento de Ciências e Tecnologias da Informação

JA(G)OBS SIMULATOR: IMPLEMENTATION OF THE MAIN FEATURES OF THE ROUTING PROTOCOL

Rui Pedro Miranda Batalha

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, Assistant Professor,
ISCTE-IUL
Co-orientador
Pedro Miguel Pedroso, PhD. Candidate,
UPC, Spain
Co-orientador
PhD. Davide Careglio, Associate Professor,
UPC, Spain
September 2011

Muito obrigado a todas as pessoas que me ajudaram durante este ano (e os outros de curso) e que fizeram isto tudo valer a pena. Pai, Mãe, Beto, Sónia, Primos (uns por me darem guarida e outros por reverem o trabalho), Pedro, Professor Cancela e um especial beijo à minha prima Sara. O outro agradecimento especial à “inkipa” por termos passado esta fase tão especial da nossa vida juntos, vão-me deixar bastantes saudades.

The present document was accomplished by the Master’s student Rui Pedro Miranda Batalha from the Instituto Superior de Ciências do Trabalho e da Empresa, Instituto Universitário de Lisboa (ISCTE-IUL) in collaboration with the Telecommunications Department of Universitat Politècnica de Catalunya (UPC).

The academic services of ISCTE and UPC

Index

List of Figures

List of Tables

Abstract

Optical Burst Switching (OBS) is an optical switching paradigm that has been re-gaining attention in the last few years after its boom around the year 2000. This paradigm is able to bring together the present technology (avoiding the optical buffer technology hurdles) and what is envisaged for future networks (packet-based optical switching). However it lacks a well-defined control plane that can keep up with quality of service (QoS) demands by Internet applications and end-to-end connectivity among multiple switching domains controlled by a single control instance.

Generalized Multi-Protocol Layer Switching (GMPLS) is a technology that can give the missing link to OBS. It is the extension of the Multi-Protocol Layer Switching (MPLS) which was designed for IP networks to introduce fast forwarding and Traffic Engineering (TE). GMPLS evolves from MPLS to deal with non-IP networks, e.g. SDH and WDM. However, it does not handle OBS so far.

This Master Thesis contributes towards this GMPLS-OBS interoperability by the development of some features to the Java Event-Driven Simulator of the GMPLS-OBS architecture called JA(G)OBS. This thesis comes up in sequence of another UPC-ISCTE Master Thesis of João Baião from September 2010, who implemented some of the basic features of the GMPLS Signaling protocol in the simulator. In particular, this Master Thesis will focus on the implementation of the GMPLS Routing protocol basic features in the simulator and to deploy a Graphical User Interface (GUI) for the simulator. The GMPLS Routing protocol considered in the simulator is the Open Shortest Path First with Traffic Engineering (OSPF-TE) which is one of the standards of GMPLS routing.

A comutação óptica de rajadas é um paradigma que tem vindo a ganhar muita atenção nos últimos anos. Este paradigma consegue conciliar a tecnologia do presente com a rede de *backbone* do futuro. Contudo, falta ao OBS um plano de controlo que consiga suportar os requisitos respeitantes à qualidade de serviço e protecção de erros.

Assim, o GMPLS poderá ser o elo que faltava nas redes OBS. O GMPLS é a extensão do MPLS onde este último foi desenhado para redes IP. Podemos dizer que o GMPLS é a actualização do MPLS para redes não-IP (SDH, WDM, etc.), sendo que é desejado que também tenha uma extensão para OBS, fazendo que seja possível ter uma rede GMPLS-OBS.

Esta dissertação teve como objectivo ajudar a equipa de investigação da UPC a implementar um simulador Java baseado na arquitectura GMPLS-OBS. Contudo, este documento não apresenta toda a documentação sobre simulador, visto que esta dissertação é a continuação do trabalho efectuado por João Baião e Pedro Pedroso no simulador, que incidiu na parte de Sinalização no simulador. Assim, esta tese incide na implementação do Encaminhamento (este é feito através do OSPF-TE) e Interface Gráfica do Simulador.

Keywords: OBS, GMPLS, Simulator, JA(G)OBS, Java, Event-Driven Simulator, Control Plane.

Chapter I – Thesis Overview

1.1. Introduction

The demands of today’s network are increasing fast; let us consider for instance all the applications inside a house that are connected to the Web. This is increasing the demand of throughput per person, creating a problem: how to efficiently provide it? This problem is even bigger when we think about the core networks. Such networks make use of optical technology (i.e. optical fibers) to transmit huge amounts of data at high speed and from which is required high reliability at the same time. So the way a core network switches and manages the traffic must satisfy such demands in a proper and effective way.

The future goal is to send IP packets through the core networks with the minimum redundancy (i.e. IP-over-WDM). One of the switching technologies studied nowadays to achieve the IP-over-WDM architecture is Optical Circuit Switching (OCS). This technology uses the same principle of the electrical circuit switching in telephone networks. The problem with OCS is when a connection is established, only the users who established it can introduce traffic in the circuit. The two users probably do not have enough load to use the entire channel. This is ineffective in terms of resource management since the traffic is more data based than voice based nowadays. Optical Packet Switching (OPS) is envisioned as the future optical switching technology. However, OPS need optical buffering and optical logic processing [1] making this choice infeasible. Subsequently an intermediate switching technology has been researched. This technology is the Optical Burst Switching (OBS). OBS combines the advantages of OCS and OPS. Hence, it can achieve much higher bandwidth utilization than OCS and with no need of buffering as the OPS do [1]. This topic will be better explained in chapter II.

Another problem is related with the lack of a Control Plane (CP) that can handle the specifications in optical networks. One of the choices that could be made is Generalized Multiprotocol Label Switching (GMPLS). This is a choice to take in account because it supports various types of networks e.g. Time Division Multiplexing (TDM), Wavelength Division Multiplexing (WDM) and IP Networks among others. Using GMPLS improves the dynamic and intelligence of the optical network. This architecture will simplify the networks protocols and will be more cost effective [9].

Hence, the goal is to have a GMPLS-OBS architecture that can cope with present and future challenges of core networks. The goal explained in [3] is to maintain the separation of Data and Control Plane (CP) given by both technologies, albeit the CP will be hybrid, based on the GMPLS and OBS interoperability. This will provide a better control in the core networks. In order to test the feasibility of this paradigm, a simulator has been designed to implement such architecture. This tool will provide the scientific community with results to better assess GMPLS-OBS solution.

This thesis is part of a broader research project that has been developed for some years now and it will not be entirely concluded with this thesis either. This being said, this thesis aims to contribute with the implementation of some features to the JA(G)OBS simulator and therefore contributing also to its general development.

The underlying basis of this thesis is the work done in [3], [9] and [46]. This document is organized as follows:

- Chapter I is composed by the introduction, motivation and the objectives proposed for this thesis.
- Chapter II comprises the description of the state of the art, where it is presented the technologies used in this thesis, namely OBS and GMPLS. It also includes the roots of GMPLS, a small explanation of the GMPLS messages, OBS procedures and how the GMPLS-OBS architecture is implemented.
- Chapter III is composed by the description of the implementation strategy for the basic features of the OSPF-TE Routing protocol in JA(G)OBS, the Graphical User Interface, the decisions and upgrades made in the code. It also includes an analysis of other simulators of GMPLS and OBS, e.g. GMPLS Lightwave Agile Switching Simulator (GLASS) [11], NCTUns [34], OBS model for OMNeT++ [22] and The Network Simulator - ns-2 (OBS-ns) [35].
- Chapter IV has the conclusions of this thesis.

1.2. Motivation

The paradigm in optical networks is changing. OBS is a solution capable of accommodating these changes, due to its granularity, bandwidth flexibility and separation of control and data planes. However, by itself, the CP of OBS cannot ensure reliability and Quality of Service (QoS) that networks nowadays need. GMPLS can ensure these missing elements to the CP. With this hybrid CP we will ensure QoS and other benefits to the network via Traffic Engineering (TE).

So it was proposed to implement a GMPLS-OBS network. The GMPLS was chosen instead of doing a new Control Plane from scratch since the goal of GMPLS is to achieve a unique control over multiple switching domains as shown in Fig. 1. Therefore, GMPLS has to be extended because so far it does not handle OBS.

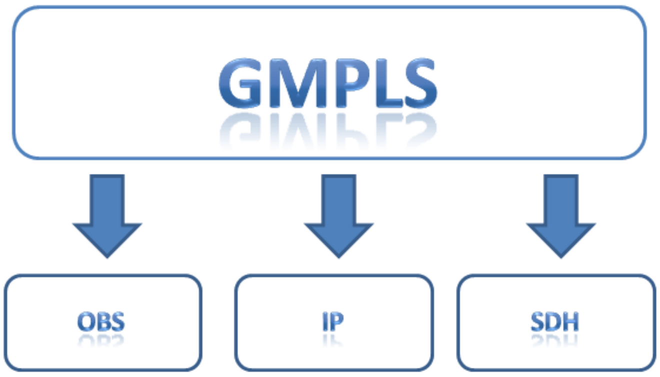


Figure 1 – GMPLS as global control plane

In order to achieve this, GMPLS must be extended properly to handle OBS switching domain.

Like all new solutions, the interoperability between two different technologies has some setbacks that have to be dealt with. In [46] these problems have been analyzed and some possible solutions were proposed. These solutions have to be tested to see if they are reliable and feasible. To test these solutions a simulator is used, that represents the demands of a GMPLS-OBS network. Despite of the many OBS simulators (NCTUns [34], ns-2 [35], JAVOBS [10] and others) and one of GMPLS (GLASS [11]), none of these simulators have the two technologies working together. Therefore, no simulator is suitable for testing the GMPLS-OBS approach.

In such a way, the Broadband Communication research group (UPC-CBA) decided to implement a new simulator. This new simulator must have most of the OBS and GMPLS tools implemented so they can communicate with each other and be as close as possible to reality.

The deployment of this simulator started from the JAVOBS simulator [10] (also a simulator from the same research group). JAVOBS is a Java based application designed exclusively to simulate OBS networks on top of the JAVANCO framework [39]. It implements the event-driven model together with fixed-increment time progression [49]. JAVOBS also implements almost all of the schedulers and schemes to schedule and dispatch a burst.

JAVANCO is a framework that, in its fundamental structure, has several packages offering a variety of features including graphical visualization, support for disk serialization of topologies and execution of common graph algorithms. It is thanks to these core packages that the user can rapidly develop and test network planning procedures through the construction of simulation models [10].

Since these two tools have so many good and usable features implemented, it was decided to add a new layer on top of them. So, our goal was to extend the simulator to enable OBS and GMPLS to communicate with each other, making a completely new simulator for GMPLS-OBS network, called JA(G)OBS.

In Fig. 2, we depict the implementation of a new layer in the JAVANCO framework (Fig.3).

The main motivation of this master thesis is to contribute with the implementation of the basic features of the GMPLS routing protocol and contribute to the general development of the simulator. This will help to predict possible problems and find their solutions in a controlled environment of a simulator.

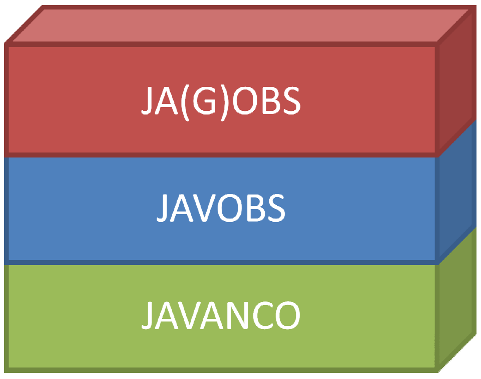


Figure 2 – GMPLS-OBS Simulator Layers

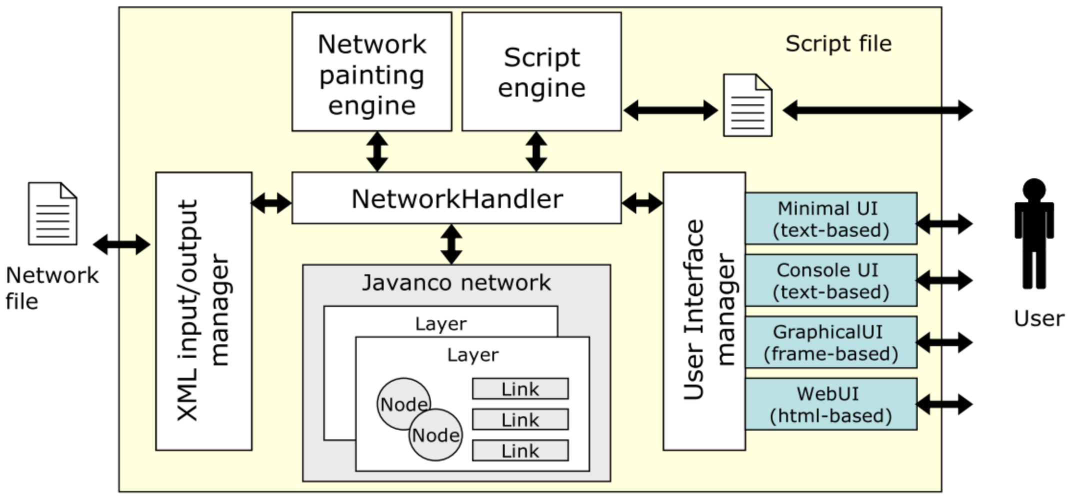


Figure 3 – General view of the JAVANCO Architecture

1.3. Objectives

The main objective of this thesis consists in the implementation of some key elements in the JA(G)OBS simulator. So, to meet the objectives it was crucial to deliver the following key elements:

1) Implementation of the basic features of Open Shortest Path First –Traffic Engineering (OSPF-TE) [27] routing protocol in JA(G)OBS. Up to date, only the Resource Reservation Protocol – Traffic Engineering (RSVP-TE) [28] was implemented from the GMPLS set of protocols. This allows a distributed routing computation.

2) Implementation of a Centralized Path Control Element (PCE). Operators and Network Providers are very fond of this element because a central node can give more accurate paths [45]. In fact, a central node running the PCE has the view of the *current* condition of the network. In our case, the goal was to test two different architectures (centralized and distributed) in the simulator.

3) Since JAVOBS/JAVANCO framework [39] has a few years now, some Java libraries used are becoming obsolete. One of the tasks proposed was to check the entire framework to change obsolete libraries for new ones. This was included in a bigger perspective of debugging all the Java classes for possible errors or better implementations of an algorithm.

4) Implementation of a Graphical User Interface (GUI). This will help people interested in using the simulator by making all the variables that the user can manipulate in a user-friendly interface.

5) And finally to carry out a simulator performance analysis of the simulator. Since this is a new simulator we are interested in finding out its performances capabilities, not only regarding reliability but also resource consumption. We have analyzed the CPU load, memory and time consumption in different network topologies which allow us to obtain reliable and extensive data regarding the simulator´s behavior.

Chapter II – State of the Art

2.1. Optical Burst Switching

Optical Circuit Switching (OCS) relies on a well known paradigm. It uses the same idea as the electrical circuit switching in telephone networks. Also it didn't have to use memory and it is easy to implement. However, OCS is not very effective when the traffic is data. This is a problem because traffic nowadays is almost data based. OCS works by making a closed circuit between the sending node and the receiving node. This approach is ineffective because the circuit spends more time without sending information than the time is sending it. This happens because the data is normally sent in bursts unlike voice traffic that is uniform. Thus, a new paradigm must be introduced, the OPS. Packet switching is more efficient in data transport than OCS. This statement is supported by IP networks using electrical switching.

Although OPS is the main target, it is currently unavailable because of the inexistence of optical buffer and optical logic processing [1]. Nowadays the only type of optical buffers is the Fiber Delay Lines (FDL). When some extra time is needed to process the burst, the node will introduce the bursts in the FDL which will increase the delay of it. This technique is effective because instead of a burst being lost, it will be delayed. Nevertheless inserting a burst in a FDL will add physical impairments to the burst. Other problem is that we cannot randomly access the bursts in the FDL, so a burst has to wait for its turn and the priority will not have any influence whatsoever on the decision.

This is where OBS [50] is introduced bringing together the advantages of OCS and OPS. OBS can achieve much higher bandwidth utilization for data networks than OCS and needs almost no optical buffers as OPS do [1]. In [15] a comparison between OCS and OBS was made. This comparison shows that OBS has a better performance, indicating that it is a good step towards a better optical network.

OBS works by sending a burst control packet (BCP), with an offset time[1], to the other nodes in the path before sending the information burst (Fig.4). The offset time allows the nodes to reserve the resources needed along the path before the burst arrives [2]. The BCP is sent normally out-of-band, it is not sent in the same wavelength (λ) as the data burst (Fig. 5). BCP processing is done electrically since it cannot be processed optically due to the lack optical technology.

OBS is more sophisticated than OCS because the same resources can be used for two or more different connections more efficiently, increasing the throughput offered by the network. This is accomplished by using the offset time to manage the resources better than having a static path like the one used in OCS.

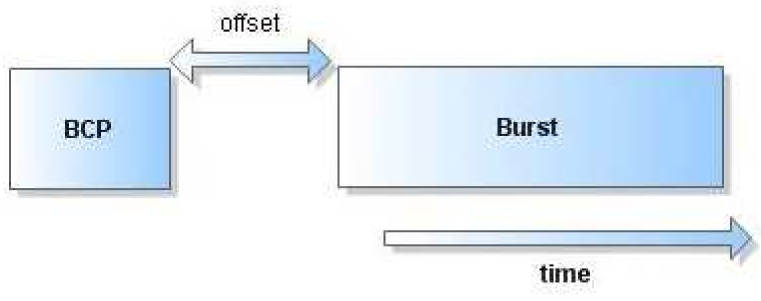


Figure 4 – OBS time diagram

Normally OBS uses two reserving schemes: 1) Just Enough Time (JET) and 2) Just In Time (JIT). The basic difference of these two schemes is that JET has a field in the BCP with the offset time. This makes possible for the node just to reserve the wavelength λ for the time it is using the resources. The JIT reserves the resources when the BCP arrives at the node. So, a small difference in the BCP packet makes JET more resource efficient than JIT without using more complexity in the packet. However, JET algorithm is more complex than JIT algorithm i.e. needs more resources of the node to be processed. This indicates that the simpler JIT reservation scheme appears to be a good choice for the foreseeable future [48]. These two schemes use the one way reservation, which dramatically increases the throughput of the network without compromising the burst, since the bit error rate (BER) is minimal in an optic fiber communication system (BER≈10^{-12}). This eliminates

[1] Offset time - is the time between the BCP and the data burst.

the Round Trip Time (RTT) in the beginning of the burst. More detailed work about these two schedulers was done in [12], [13] and [14].

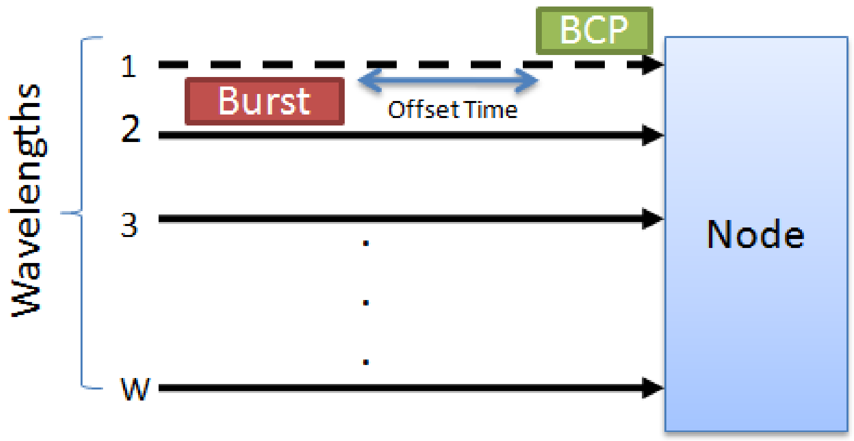


Figure 5 – BCP out of band

2.2. Generalized Multi-Protocol Layer Switching

The GMPLS is the generalization of the MPLS. The MPLS is a protocol designed to give Fast Forwarding abilities to IP networks. The forwarding of the packets is based on labels. The node doesn't have to check the entire header but only the label that is substantially smaller. The label is checked and the node sees in the list where the packet must be forwarded, making the routing decision faster than to inspect the all IP header.

MPLS then evolved to give Traffic Engineering (MPLS-TE) [37]. TE in MPLS has a focus on measurement and control functions [37]. This facilitates the efficiency and reliability of the network operations while simultaneously optimizing network resource utilization and traffic performance. So MPLS, amongst other things, helps to ensure QoS in IP networks. This is possible because MPLS promotes minimization of packet loss; minimization of the delay; maximization of the throughput and enforcement the service level agreements.

Since the MPLS can only manage IP networks, it was a logical step to make it a more general protocol because other types of networks need also a control plane. GMPLS was the answer. It can support other types of networking that are not IP based, e.g. time-division (SDH/SONET), spatial switching and wavelength switching. This new features makes it flexible for a CP capable for handling a unified way multiple switching [32]. Also GMPLS adds RSVP-TE and OSPF-TE which have been revised to better handle and distribute TE which was not possible with standards protocols.

At this point, it is crucial to highlight that GMPLS is not a protocol, but rather a group of protocols that can be separated in three categories: 1) Routing, 2) Signaling and 3) Link Management.

1. The routing can be done by Intermediate System to Intermediate System- Traffic Engineering (IS-IS-TE) or OSPF-TE. Since OSPF-TE is more sophisticated and it is preferred by vendors, this is the one used in the simulator. Since one of the focuses of this thesis is to make the routing in the simulator functional, the OSPF-TE will be explained later in this chapter (section 2.3.2).

2. The signaling can be done by: Constraint-based Routing Label Distribution Protocol (CR-LDP) or RSVP-TE. The signaling part of the GMPLS lies outside of the scope of this thesis, since was already implemented in JA(G)OBS [30]. Therefore, we shall no further elaborate on it.

3. The Link Management is done by the Link Management Protocol (LMP). Once again it does not lie within the scope of this thesis, for detailed information about this subject see [16].

2.3. GMPLS Routing

2.3.1. Link State Routing

The LSR_1 is one of the classes of protocols that were elaborated to disseminate data about the state of the network. Many types of routing protocols are based on LSR_1, for instance, the Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF) are two examples. The LSR_1 came out to substitute the Distance Vector Routing (DVR). Two main issues were identified in DVR, namely i) the metric used was the message queue in memory, and not the time it takes to get from node A to B. This became a problem when the bandwidth became higher, although this problem could be circumvented. However ii) the count to infinity problem was an issue that couldn’t be circumvented properly, only partial solutions were developed [5] (Poisoned Reverse and Split Horizon). This overthrew DVR chances of maintaining his throne.

Thus, according to [4], there are five main ideas underlying LSR_1 success:

1. **Discover its neighbors and learn their network addresses.** When a node is initialized it will send a Hello packet to the neighbor nodes. The neighbor node will respond with his information and name. These two parameters must be unique in the network to maintain a correct topology.

2. **Measure the delay or cost to each of its neighbors.** For the LSR_1 it is very important for a node to know the delay to each one of its neighbors. Without this metric the main idea of the protocol wouldn´t be accomplished. The most common way to determine the delay is sending an ECHO packet to the neighbors, they respond as soon as the message is received (minimizing the processing time). Then the node only has to calculate the RTT and estimate the propagation delay. This process is very similar with the ping message of the Internet Control Message Protocol (ICMP) [29] protocol.

   However the cost is a controversial subject because it can mean the actual money that is spent to use the path, but it can mean packet lost probability in a line just to name a few examples. So, in short, this is up to the network manager how to measure the cost. This is why nowadays we use the Traffic Engineering tools. These tools give more flexibility for the telecommunications engineers to create new metrics and new ways to manage a network. TE will be explained later on.

3. **Construct a packet telling all it has just learned.** The packet must have the identification of the sender, sequence number of the packet, the packet age and the list of neighbors with the respective delays. The propagation of this information is the hard part, because in some situation sending them periodically is the answer, but when a significant event occurs, it is useful to send information right away. The way we send the information is important but is an open matter in LSR_1.

4. **Send this packet to all other routers**. This is probably one of the most important steps in the process, although flooding information can be a big problem when using LSR_1 based

protocols, because the packet can loop around in the network if certain measures won´t be taken. The nodes in the network must keep record of the sequence number of the packets it sends. This will prevent the message looping in the network. Since this method is not flawless, for example a node can *crash* and the information that was contained there is lost. Because of this problem, the field Age in this type of protocol is very important. If a node restarts, it will start to send message with the ID=0. Although this packet won´t be rejected by other nodes because the field Age is not equal to 0. This means the packet is "younger" than the previous packet sent (the age of the packet starts with a certain number and is decremented every 10 seconds until it reaches 0 and is discarded).

5. **Compute the shortest path to every other router**. This last step normally is calculated by the Dijkstra algorithm or a variant of it that has some other factors into account, e.g. QoS. When a new packet is received the information that is contained in it is processed and compared to the one in the graph, if it has a shortest path the graph will be updated, if not, the packet is sent to the other neighbors without changing anything.

2.3.2. Open Shortest Path First

The OSPF was created when the Routing Information Protocol (RIP) [5] protocol, which was based in DVR, started to show weaknesses. This happened when the networks became bigger due to the problems explained earlier. Thus, this new protocol (OSPF) had to support several requirements:

- Had to be open source, i.e. everyone who wants to use this protocol is allowed to do so without restrictions.
- Had to be able to support and compute different types of metrics mentioned earlier.
- The protocol had to be dynamic, adjusting itself to constant network changes.

The OSPF-TE protocol is one of the routing protocols in GMPLS, as was mentioned early in this thesis. This is one of two protocols that are distributed and designed to run in Autonomous System (AS). Since OSPF was designed specifically for the Transmission Control Protocol\Internet Protocol (TCP/IP) internet environment [8], IETF had to add more capabilities so that routing could be done in other types of networks.

Thus, the Traffic Engineering (OSPF-TE) is used. TE is the study of the optimizations of the performance in a telecommunications network, using statistics and the interpretation of nodes behavior to increase the overall performance and reliability in a network.

The extension in OSPF was made because of the different demands in different networks. For instances, there are networks that the routing should be made by measuring the load in each node, which is our case. To use this type of metric there are a special type of Link State Advertisements (LSA). They are called the Opaque Link State Advertisements because in some nodes the information contained in it is unreadable for them, so these nodes only redirect the packets without reading the information in it. This subject will be approached later on in this chapter.

Next the type of packets used by OSPF to manage the routing in a network will be introduced.

2.3.3. Types of Packets of OSPF

In OSPF there are five types of packets and each one was designed to do a specific task in the network:

1. **Hello:** The *Hello* packet is the type 1 packet of OSPF protocol which allows the communication between adjacent (neighbor) nodes in the network. These packets have two main purposes i) to ensure bi-directional paths and ii) to create adjacencies (sharing policies) between the nodes. The bi-directionality is established by including in the packets a list of all nodes seen sending a hello packets recently (except for the first hello packet). This message is sent in different ways depending on the kind of network we are working on. Since the thesis case is a broadcast network, this is the only type that is going to be referenced on from now on. On a broadcast network, when a node first enters a network it will broadcast a hello packet to the entire network. This packet is the only one in OSPF that the node is responsible to be sent to all nodes in the network. When the others nodes receive the packet, they will respond with other hello packet, but this one includes the list of active nodes. With this information the node can start his adjacencies sending other types of packets. To maintain the adjacencies, hello packets will be sent periodically through the network in periods agreed by the nodes. In Fig. 6 there is an example of the process described.

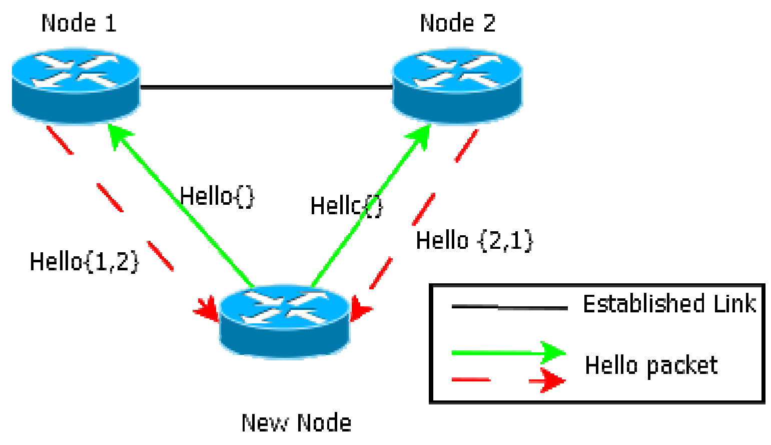


Figure 6 – New Node entering a network

2. **Database Description**: This packet distributes the database between the nodes. When the nodes finished the process of discovering each other, they have to complete the formation of an adjacency, i.e. complete the submission of information of the nodes. To do so, the nodes have to transmit the information they have about the network. They will start sending type 2 OSPF packets, the database description packet. This type of packet describes the contents of the Link State Advertisement (LSA) in the database (LSA will be explained later on). Usually one packet does not have enough space to fit all LSAs gathered by the node, so a string of packets may be required to send the entire database. In the beginning of the transmission the nodes have to decide who will be the master and the slave on this information sharing. Normally the node that is connected the longest time in the network will be the master because it normally has a more detailed database. This type of packet is only distributed by the neighbors to spread LSAs to other adjacent nodes; this reduces the load because it reduces the number of packets that pass through a node.
3. **Link State Request**: When the database is exchanged between the nodes, some of them consider the information that was given is old and they need new information. In this case, the node will ask for more up-to-date information. Resending all LSAs in the database is unpractical and introduces unnecessary load in the network. Thus, the node will send a Link State Request (LSR_2) with the information of the LSA that is required. The response to this type 3 packet is made by a Link State Update packet.

4. **Link State Update**. This packet has the basic principle: to spread information along the network. However the behavior of this message is different depending on the type of network we are using. The network can be event based and time trigger based. Since this packet is the only one that is implemented in the simulator because we assume that the network is static i.e. no more nodes or links will be introduced in the middle of a simulation, so implementing other types of packets at this point will not have an impact on the results. The Link State Update (LSU) will be better explained in the next topic of this chapter.

5. **Link State Acknowledgment**: The Link State Acknowledgment is the type 5 of the OSPF packets; they exist to make the flooding of information of OSPF reliable. The Link State Acknowledgment can be sent in two different ways, delayed or direct to the neighbor. The delayed has the advantages of facilitate the piggybacking of various Link State Acknowledgments in the packet, i.e. enables a single Link State Acknowledgment packet to indicate various acknowledgments to several neighbors at once. This also randomizes the pattern of sent messages. The transmission of the packet must start (and end) before a pre-established time interval to prevent needless retransmissions.

   The Link State Acknowledgments are sent directly to the neighbor that were the source of the duplicated LSU, on the contrary to that we would expect this is the way it is done instead of sending a Link State Acknowledgment for all LSU received. This method is used because it saves bandwidth and acknowledges can be made by the LSU message.

More detailed information about OSPF messages can be found in appendix A.

The next topics will explain in more detail the LSU and its components since this was the only packet implemented in JA(G)OBS.

Link State Update Packet

The link state update is the type 4 of the OSPF packet and it is used to disseminate the information in the entire network. The mechanism used by this message is not a typical flooding, but a Split Horizon flooding, where a node only needs to ensure that the packet traveled to its neighbors except through the one from which it has received the message before, this happens with all other packets with the exception of the hello packet.

This method has some advantages 1) it sends less packets to network (this is useful preventing network congestion) and 2) it is less likely that packets loop in the network. If the LSU was not received properly, the sender node receives a Link State Acknowledgment. Instead of flooding the information again the node will only send the LSU to the neighbor who didn't receive it.

There are two ways to configure the flooding of the LSUs 1) the time trigger event and 2) the event trigger event:

1) The time trigger sends LSUs in a pre-established timer, normally this timer is configured to be 10 seconds. So the node has to inform every 10 seconds of its links situation. These messages have always to be sent even if there isn't any new information about the link state. They have to send the messages to ensure the other nodes that it is online and cannot be discarded from the forwarding table. This could be a problem if the network is congested and the packets do not arrive on time. This will cause the node to be erased from the forwarding table of the others when it is not suppose to, causing problems in the node because it will assume that links are offline when in fact they are online. Despite of this fact, this is considered a reliable way to exchange information because these situations are almost nonexistent.

2) In the Event trigger, if we assume that the network is stabilized, i.e. no more nodes are entering or exiting the network, the only OSPF packet that travels through the network is the LSU. In Fig. 7, it is shown how the LSU flooding mechanism works. The node A will receive a path acknowledgment (RESV message) from the receiving node (the event). This is a signaling message from the GMPLS protocol RSVP-TE. At this point it will update its database with the LSA created using the information in the signaling message and flood a new LSU packet, with the LSA inside to its neighbors.

When its neighbors (node B and D) receive the LSU they will perform the same steps, but with a small difference; they will not send the message to the link they receive it from, performing a Split Horizon. This behavior doesn't allow the messages to loop in the network preventing unnecessary load in the network. This last procedure is repeated until the LSU was seen two times in the same place, when this happens the LSU will be discarded making impossible for the message to loop. This is how the event trigger works. One event triggers the spreading of a Split horizon flooding.

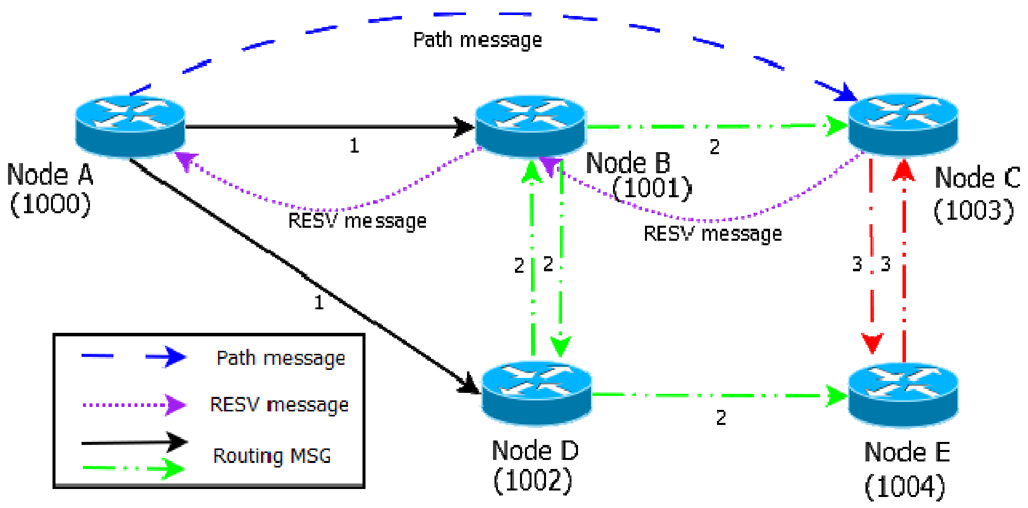


Figure 7 – Example of LSU Spreading

The format of the LSU packet is described in appendix A.

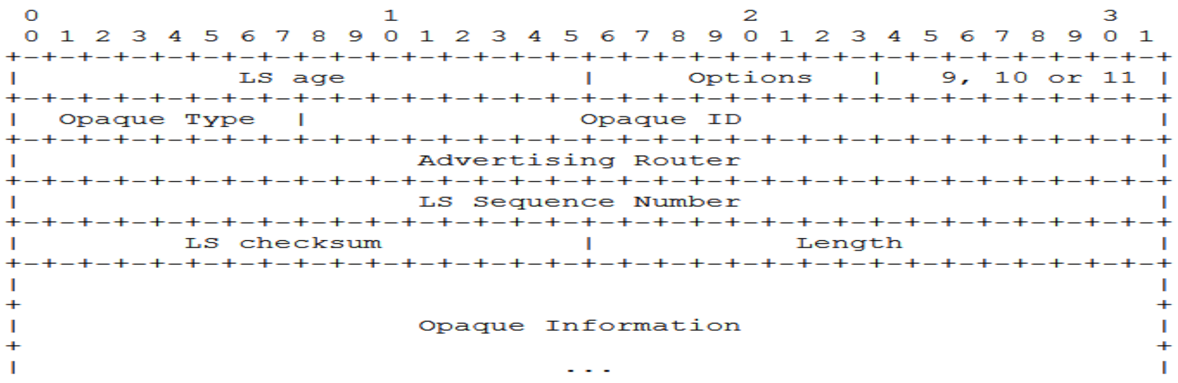
The next topics will explain the content in a LSU packets i.e., the Link State Advertisement (LSA).

Link State Advertisement

The Link State Advertisement (LSA) allows the communication between nodes, because without the LSA it would be impossible to disseminate information over the network. This container is where the information is stored for transmission, unit of data [8]. LSAs are like packets, each has a header (Fig. 8) that has important information that helps the node compute it. There are eleven different types of LSAs, each one with a specific purpose. Since we only implement the type 10, it is the only one detailed in this thesis. Type 1 and 2 are in appendix A, the rest can be found in [7] and [8].

Link State Advertisement – type 10

This type of LSA is used on a network that uses traffic engineering. It is one of the few types that are known as opaque LSA. The opaque LSA can be used in a network with nodes that cannot read this type of information, for them it is opaque. They will distribute the packets without reading its content. This is the reason why this type is called *opaque*. Opaque LSAs provide a generalized mechanism to allow for the future extensibility of OSPF [7]. Since the TE network sends more types of information than the other types of network, like the bandwidth used, thus the use of opaque-LSA is mandatory (no other type of LSA can handle this type of information). Type 10 is used in our case because in a real life situation this type will flood the information only to the determined area and not to the entire network. The Fig.8 shows that an opaque-LSA header is different from the common LSA structure.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LS age             |     Options   |   9, 10 or 11 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Opaque Type  |               Opaque ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Advertising Router                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      LS Sequence Number                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         LS checksum           |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                      Opaque Information                       |
+                                                               +
|                              ...                              |
```

Figure 8 – Link State Advertisement – type 10 [8]

Description of the several message fields:

Opaque Type	This field has the values of 0 to 127 for IETF Consensus action and 128 to 255 for experimental actions. In our case this value is 1, because it is the number used for Traffic Engineering LSA.
Opaque ID	This field has an arbitrary value for maintaining various LSA-TE.
Opaque Information	This topic is still under discussion

2.4. GMPLS-OBS Architecture

In the previous sections it was explained GMPLS and OBS separately and what were the motivations to have them. In this section it will be explain how the two interact with each other.

The GMPLS and OBS technologies have some years now, however an interoperability solution has been continuously postponed. The main problem of these two technologies together relies on the different signaling schemes they use to do the resource reservation. OBS signaling consists in a one way signaling procedures while GMPLS requires a two way procedure.

The proposal made in [3] uses a hybrid control plane (Fig. 9), where both GMPLS and OBS have to perform signaling and routing functions. One of the problems with this solution is the communication between the two CPs. There are two solutions on the table for the connection itself: i) using two separate networks (Fig.10a)). This has the advantages of relieving the management and control processes between the nodes, making the network more resilient to failures i.e. if one of the networks begins to mal function the other can carry on without being too strongly harmed. Lastly, the cost of the nodes will decrease because they will be less complex. The main disadvantage relies on the fact of maintaining two different networks instead of only one. To compensate this fact we can ii) aggregate the GMPLS network to work (Fig. 10b)) in a wavelength of the optical fiber. However this solution has the problem of using resources that could be used to transfer data. So this problem doesn’t have a straightforward solution, instead it has to be designed on a case by case basis. In this thesis two separated networks were used.

The other problem is the integration between the two technologies. This problem is more complex and it is the main focus of [3] and [9]. Because the OBS and GMPLS work in two very different time scales, they will have different responsibilities in the control plane of the network. Since OBS works in the timescale of micro/milliseconds it is responsible for the Resource Reservation and the “current” network resource availability because these tasks require a fast decision. The GMPLS is responsible for the tasks that are more stable (timescale of minutes or hours), the Network Topology and the Virtual Topology Management. More information related to this topic in [3].

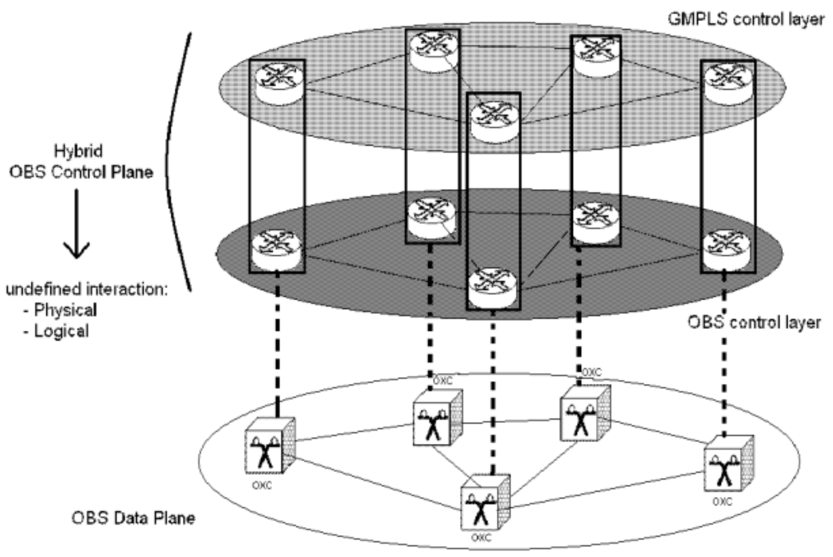


Figure 9 – Different Planes in GMPLS-OBS network [8]

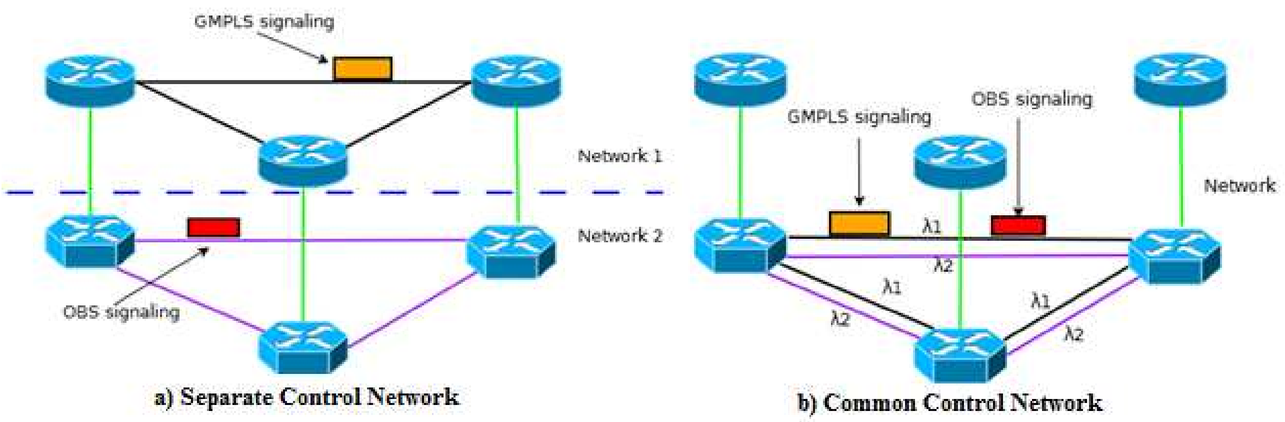


Figure 10 – Two physical topologies of GMPLS-OBS network

OBS Background Task

These tasks are the responsibility of GMPLS.

1. **Virtual Topology Management -** It is responsible for maintaining and tear downing Label Switch Paths (LSPs) between a pair of edge nodes [3]. Despite of this, no resource reservations is made. The reservation is done in a different time scale. The LSPs are maintained to complete a path between the edge nodes. The maintenance of LSPs in OBS adds protection, restoration, link management and QoS.

2. **Network Topology Information –** This part includes two different pieces: 1) *Information dissemination* which is responsible of dissemination of the resources information along the network. The OSPF-TE is responsible for this action because the network is relatively stable and for this reason there aren't many modifications. 2) *Path Computation* this part has the task of computing and distributes the best paths possible, decreasing the percentage of Burst Loss. The way of doing this is still an open matter. The proposal made in [3] says that to support the dynamic routing we need to flood frequent network-resource-update messages without modifying the signaling protocol. Instead it is proposed to give more dynamism to the LSP selection when the BCP is created. The explicit route is based on a given traffic intensity matrix. An intelligent and careful network planning based on such traffic patterns and statistical behavior (TE functions) helps us to better define a set of static, explicit routes and a set of limited dynamic, explicit routes for time-limited traffic demands. This approach does not require accurate network resource availability information and therefore does not incur in high instability. This feature is also being operated in GMPLS since the time is not a problem, because of the scheme presented and also secures the advantages of GMPLS.

OBS Specific Tasks

These tasks are OBS responsibility.

1. **Resource Reservation –** which is responsible for reserving the bandwidth necessary to transmit the bursts, because the bursts have a timescale of micro/milliseconds. The Burst Control Packet has the necessary routing information for the core nodes to re-arrange the switching matrix and the burst is sent to the right path. This mechanism assures the best

results without compromising the path in the LSP.

2. **Network Resource Availability Information –** is responsible for gathering and disseminating the "current" resource availability of the network. This is very important for a good network functioning because if it is well engineered the burst loss will decrease dramatically. In order to be faster, this process has to be done optically, precious milliseconds wasted in O/E/O will decrease the efficiency of the protocol, although, the process to do it isn't yet defined. The proposal in [3], one of the basis of this thesis, says to implement this feature in the OSPF-TE protocol, the LSAs associated with each established LSP must be used. For this to work a specific extension made in an *opaque* LSA and some modifications are needed. LSAs information has to resume the status of the links and the core nodes, e.g. how much traffic is in a link and how many wavelengths are being used. This metric is not exact because the node won't have accurate states of all nodes in the network (there are always new events occurring in the network). However this does not affect the performance of the proposal because *normal*ly the information gathered in the node when the BCP is formed is enough to prevent collisions of the bursts.

Like it was said before, some changes were proposed for the GMPLS and next it will be referred and explained what types of changes were proposed in [9].

RSVP-TE

In this protocol normally we can only have one label request per message [9], but this restricts to one LSP request at the time, so we can only reserve one wavelength. This is very inefficient because a burst may need more than one wavelength. So it is proposed TE tunnels[2] that can have one or more LSPs depending on the traffic demand. This is done with just two messages sent (Path and RESV) per LSP, making it more efficient in terms of load (using fewer messages than the normal protocol) and solving the problem of having more than one wavelength for LSP.

[2] TE tunnels - The traffic that flows along a label-switched path is defined by the label applied at the ingress node of the LSP, these paths can be treated as tunnels [51].

OSPF-TE

In the proposal as explained before, we can have one or more wavelengths per TE tunnel. Some of the wavelengths can be shared between two or more nodes. So the authors of [3] introduced a new state that is called *Shared* which defines the wavelengths that are being used for more than one tunnel. The wavelength 4 in the Fig. 11 exemplifies a *Shared* wavelength. This wavelength has to be announced in a different way. The nodes that need to create new LSPs have the possibility to use these shared links if necessary. However the standard LSA messages do not have enough space to have information about this aggregations and the state of the links. So the authors pick up a previous hypothesis [21], adding two or three more bits in the LSA that can have extra information about the state of the link, instead of the On/Off state. With this we can introduce the link state status and more information that can be useful for computing the best possible path.

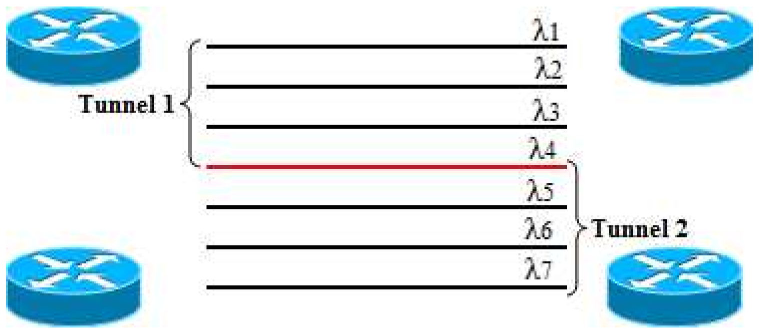


Figure 11 – Shared Wavelength

Chapter III - Event-Driven JA(G)OBS Simulator

The JA(G)OBS simulator is an event-driven tool to simulate GMPLS-OBS networks. It was built on top of JAVOBS [10] and it is an evolution of the JAVANCO framework [39]. The goal is to introduce a fully functional and independent new layer on the simulator that can communicate with the OBS layer, making simulations possible with this new architecture.

The GMPLS controller comprises a Signaling and Routing modules and interfaces for them to communicate with each other and with the OBS controller. Note that the LMP was not implemented (left for future work). The controller´s components, configuration and communication (between the controllers) of the simulator are shown in Fig 12.

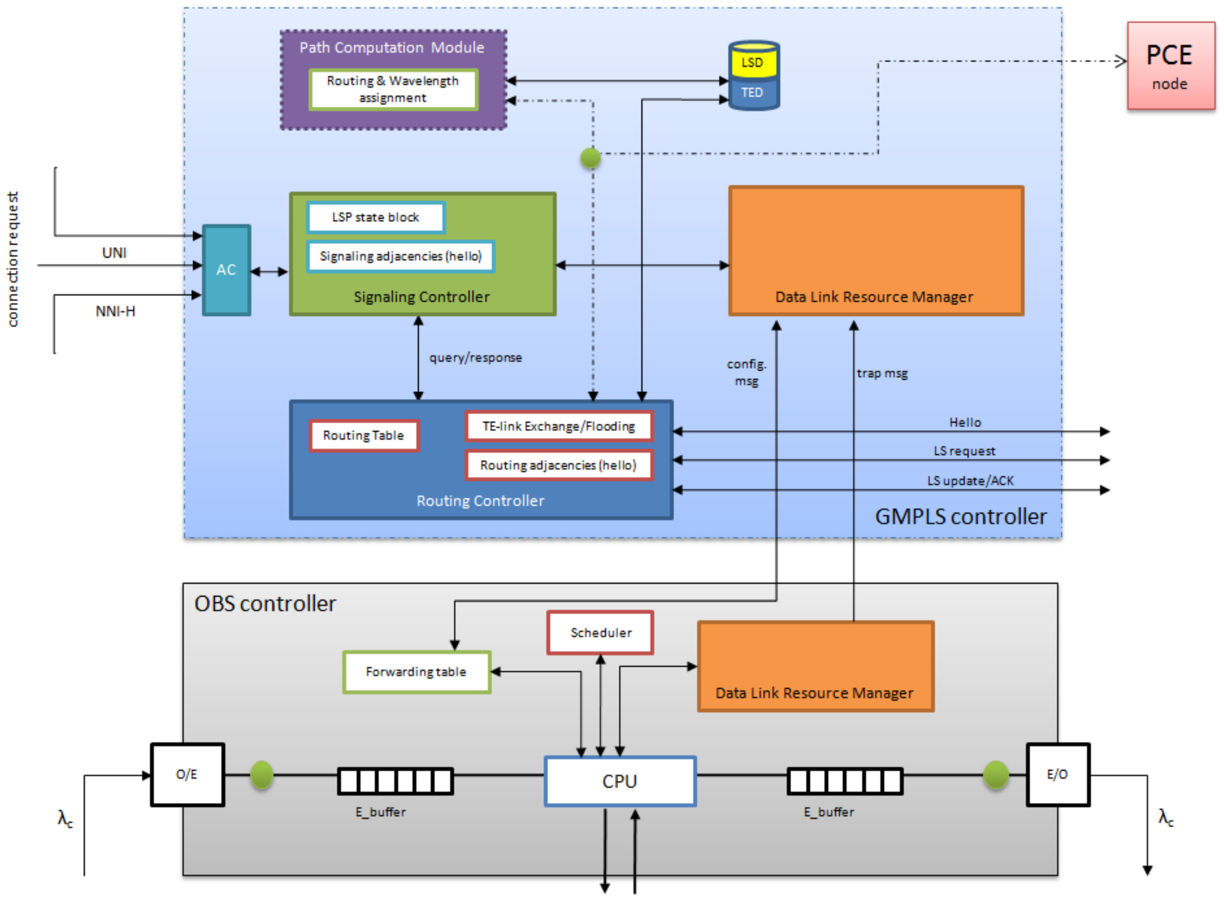


Figure 12 – GMPLS/OBS simulator architecture

The Fig.12 also shows the architecture of a GMPLS-OBS node in JA(G)OBS simulator. In this paragraph it is briefly explained the tasks made by each module. In orange it is shown the Data Link Resource Manager. This module is responsible to manage the wavelengths available in each node. In purple is the Path Computation Module, this is where the paths are computed with the information stored in the database (LSD/TED) and with the information just gathered by the Routing Controller (in blue). The Routing Controller has the function of create, flood and receive LSU messages when necessary. The Signaling Controller (in green) has the job of signaling new paths established in the network. The forwarding table in OBS controller has the table of entries used by the Data Link Resource Manager to help compute the wavelengths available in each node. The other blocks in the OBS Controller simulate the jobs done by hardware level in the node.

We focus our attentions on the dissemination of the LSU and signaling messages because, as said before, these messages are the ones who carry the information necessary for the network to operate. We also implemented various interfaces for the new types of messages that can be introduced without almost any effort, e.g. the interface *LinkStateAdvertisement*, which is ready to integrate more than the opaque LSA that we use now. It was also our goal to make the simulator as modular as possible because this will make future works much easier. For example, in the middle of this thesis the shortest path algorithm was changed for a new one without almost any code changes. This is only possible because of the modularization and interfaces available in the simulator.

The work developed during this thesis was focused on creating the Routing Controller module in blue in Fig. 12, PCE node and the communication between them and the other components of the simulator.

The next topics will explain in detail what was done on the simulator.

3.1 Signaling Implementation

The construction of the GMPLS layer was done in parts. The first part was to implement the Signaling protocol which was done by Pedro Pedroso and João Baião. Details about the implementation can be seen in [30]. Regarding the signaling structure there were two changes made in the simulator during this thesis:

1) Implementation of the Path Computation Element, which is not part of the GMPLS architecture so it will be explained later;

2) Implementation Yen Shortest Path [24] to fit in our simulator. We decided that implementing the algorithm from scratch was very time consuming and the results wouldn´t differ from the ones that we have now. Also the implementation of Yen Shortest Path in [25] was shown to be robust and fast. So we modified it to work with our simulator. This was possible, as said before, because the modularization introduced in the simulator. The introduction of this algorithm was almost straightforward.

3.2. Routing Implementation

The routing in the simulator is done only by OSPF-TE. IS-IS-TE protocol is out of the scope of this thesis. The routing is responsible to guarantee that the local Database (LSD/TED in Fig.12) at the GMPLS controllers is up to date. In Fig. 13 it is shown how the flooding of one message is done when a LSP is created between the node A and C.

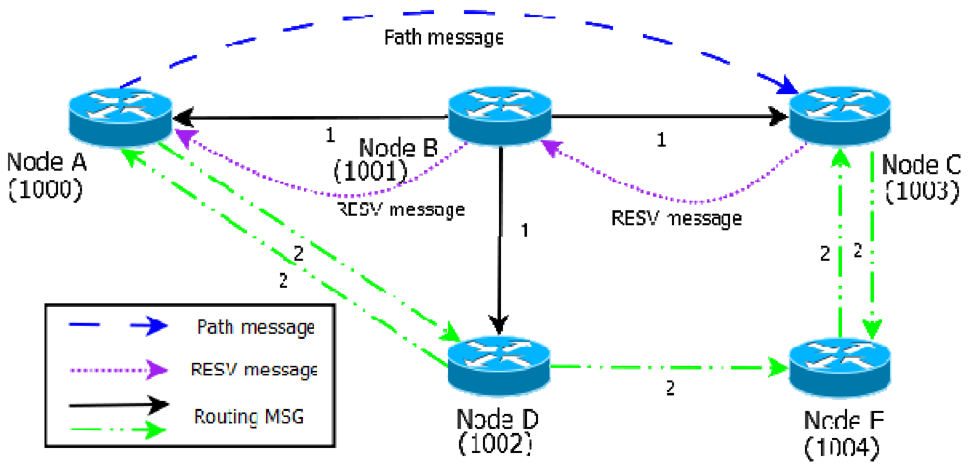


Figure 13 – Flooding Procedure at Node B

When node B receives a RESV message (signaling message) it has to update its local database and flood the information because a path was reserved. It is crucial to maintain the local database up-to-date because future routings will be made based on the information on it. So the task to be performed by node B is to spread the information to its adjacent nodes, node A, C and D. To do this, it has to create a new message first (i.e. *GMPLS_RoutingMsg*) – see Fig.14. The message is formed by a Header and an *Opaque* LSA. On its turn, *Opaque* LSA has also and Header and Type-Length-Values on it. The TLV has three variables:

1) Type: which determines what kind of LSA it is, opaque or not.

2) Length: this variable has the size of the variable value.

3) Value: it has the information about what we want to spread. In our case there is a list of TLVs that have the values of the QoS, load and wavelengths status. The number of TLVs on the LSA depends on the size of the information that we want to spread. This is the closest structure that could be obtained to [7].

When the message is created, the information that we want to spread is introduced, in this case a path was established between A and C. Once this happens, when node B receives a RESV message, meaning that a path was established, it needs to send the information to its neighbors. Therefore Step 1 on Fig. 13 is done, i.e. the routing message has been received in the adjacent nodes. Now they have to do what the node B did before: flood the message to its neighbors, except to the one it was received from (split horizon).

Let us take the example of node D. This node has to send the message to the node A and E. Because the process is parallel, the node A is also sending the same message to the node D, so the nodes have to discard the second message that passes through them with the same message ID. This avoids that the message enters in a loop. This is the beginning of the process of the dissolution of the message from node B because when a message is seen twice in the same place it is assumed that all its adjacent nodes have received this information already. The Fig.7 (section 2.3.3) shows the continuation of the process when the message arrives to node A.

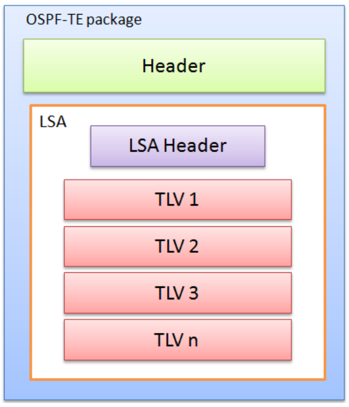
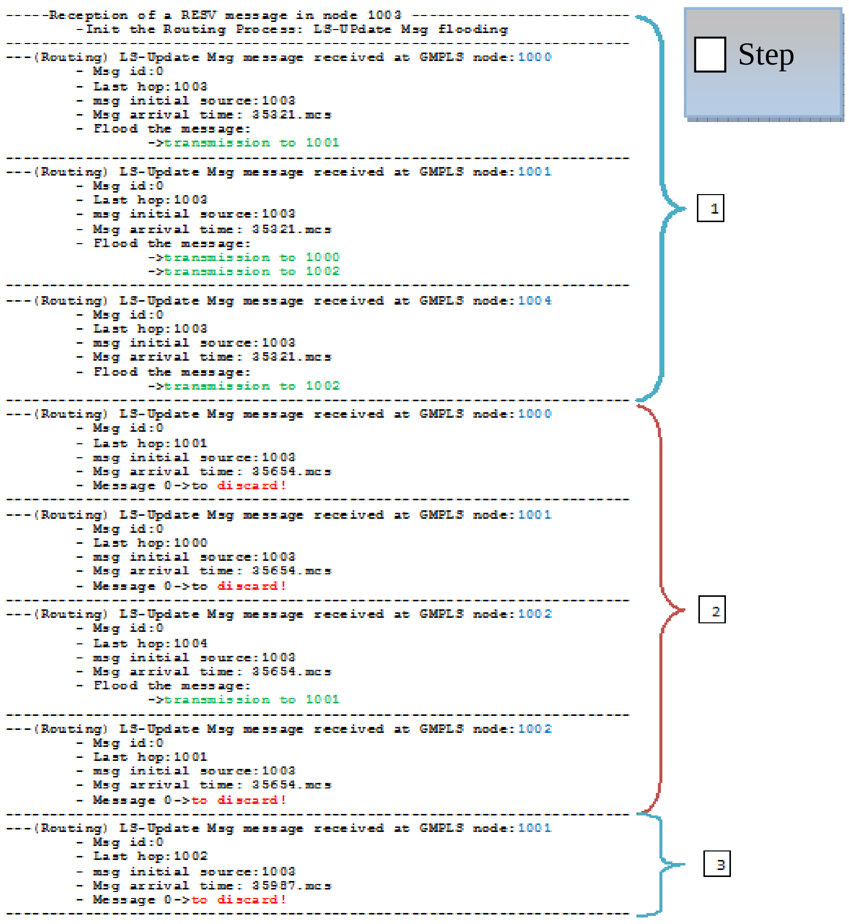


Figure 14 – Structure of GMPLS Routing message.

This implementation was successfully made in JA(G)OBS. In Fig. 15 there is the modified output of the simulator, concerning the LSU flooding. This figure was modified for better explanation purposes only. The unmodified output can be seen in appendix B.



```
-----Reception of a RESV message in node 1003 -------------------------
        -Init the Routing Process: LS-UPdate Msg flooding
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1001
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1000
                ->transmission to 1002
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1004
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1002
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1001
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1000
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1004
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Flood the message:
                ->transmission to 1001
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1001
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
-----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1002
        - msg initial source:1003
        - Msg arrival time: 35987.mcs
        - Message 0->to discard!
-----------------------------------------------------------------------
```

Figure 15 – LSU flooding in JA(G)OBS

The process described in Fig. 15 is similar to the one described in the paragraphs above. In the first step, node 1003 (node D of Fig.13) receives a RESV message that triggers LSU flooding. Then it has to send it to is adjacent nodes 1000, 1001 and 1004 (nodes A B and E of Fig. 13). In blue, we can the nodes that received successfully the LSU at the first step.

In green we see that those nodes flood the message to their neighbors. In the second step some of the nodes received the same message twice from a different node, for example the node 1000 receives the same message, although now it is from node 1001 which flooded to its neighbors in the first step.

In red, it can be seen that in those nodes the message is duplicated and is discarded. The only node that received the message for the first time was node 1002 (node C Fig.13). Also in the second step it can seen that node 1002 sent the message to node 1001, this happened because the message from node 1003 was processed first and subsequently the node 1002 didn´t had in its database that node 1001 sent the same message at the same time. So in the third step it can be seen the end of the process with node 1001 discarding the message sent by 1002.

3.3. PCE Implementation

The central Path Computation Element (PCE) is a special node that *sniffs* all the information in the network to have a complete view of the “current” state of the nodes and links in the network. The notion behind this architecture is that a central node with all the information can compute better paths than using only distributed computation (as standard GMPLS) (i.e. with a local PCE at each node). The two main disadvantages of using the PCE is the more traffic generated in the network and with PCE there is a point of failure in the network, if the PCE fails the all network start to malfunction, this doesn’t happen with GMPLS.

The process of asking for a new path is a bit different from the one of normal signaling. The Fig. 16 shows how the process is done when a burst arrives to node A. First the node A has to send a Path Computation Request message [47] requesting a path to the PCE node. Because the PCE node connection is based on Transmission Control Protocol (TCP) [23], the first time this connection is established is much slower than sending directly to the node due to the fact that TCP uses the “three way handshake”. Despite this fact, when the PCE node receives the message, it will compute a path with the extra information that it has from the network and sends back a message with a path. When node A receives this Path Computation Reply Message [47], it has to confirm that the path is not occupied with another burst, keep in mind that other nodes are asking for resources in parallel of this one. The only way to confirm that the path is still available is to send a Path message to where the burst final node is (node C). Only when the RESV message comes from it, the process can continue with the routing (explained in the routing implementation section) and the actual sending of the burst.

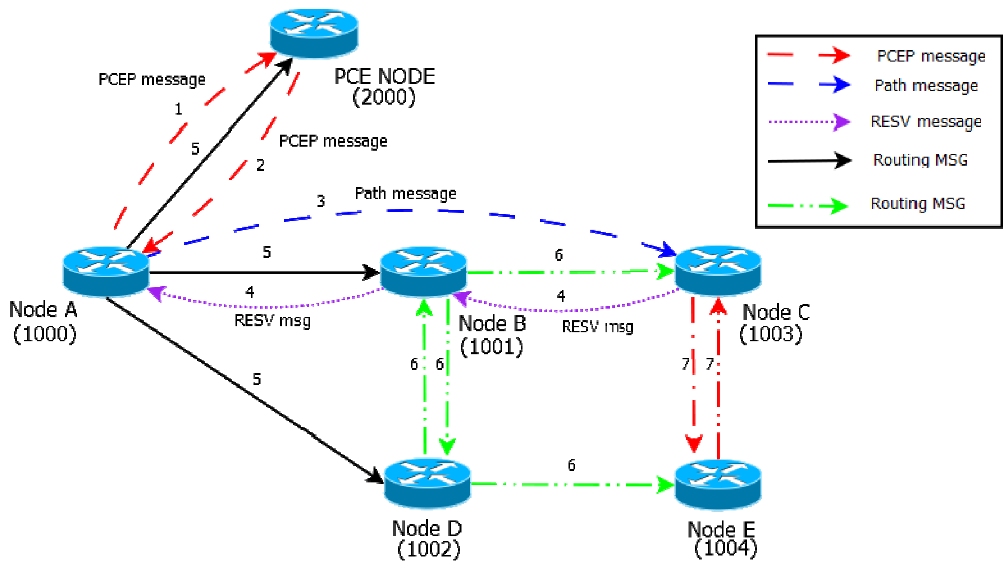
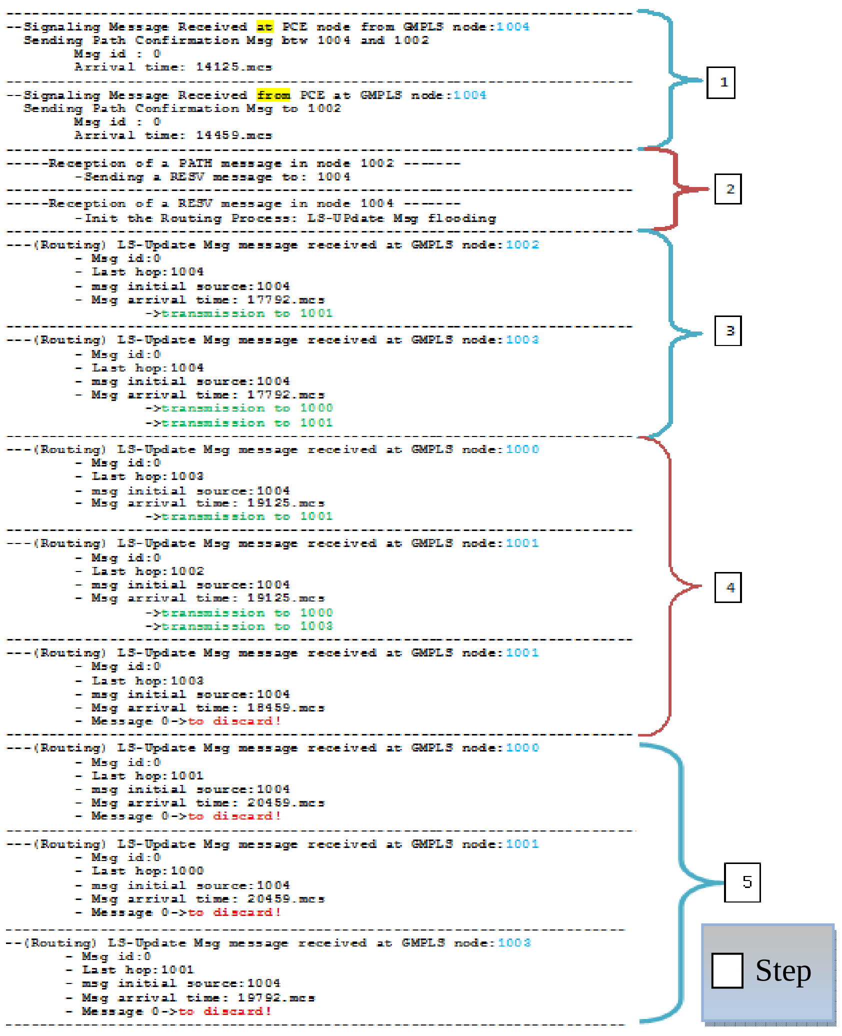


Figure 16 – Signaling and Routing with a central PCE node

Figure 17 we have the modified output (to better explain it) in JA(G)OBS when a node asks for a path to central PCE. This figure was modified for explanation purposes only. The unmodified version can be seen in appendix B.



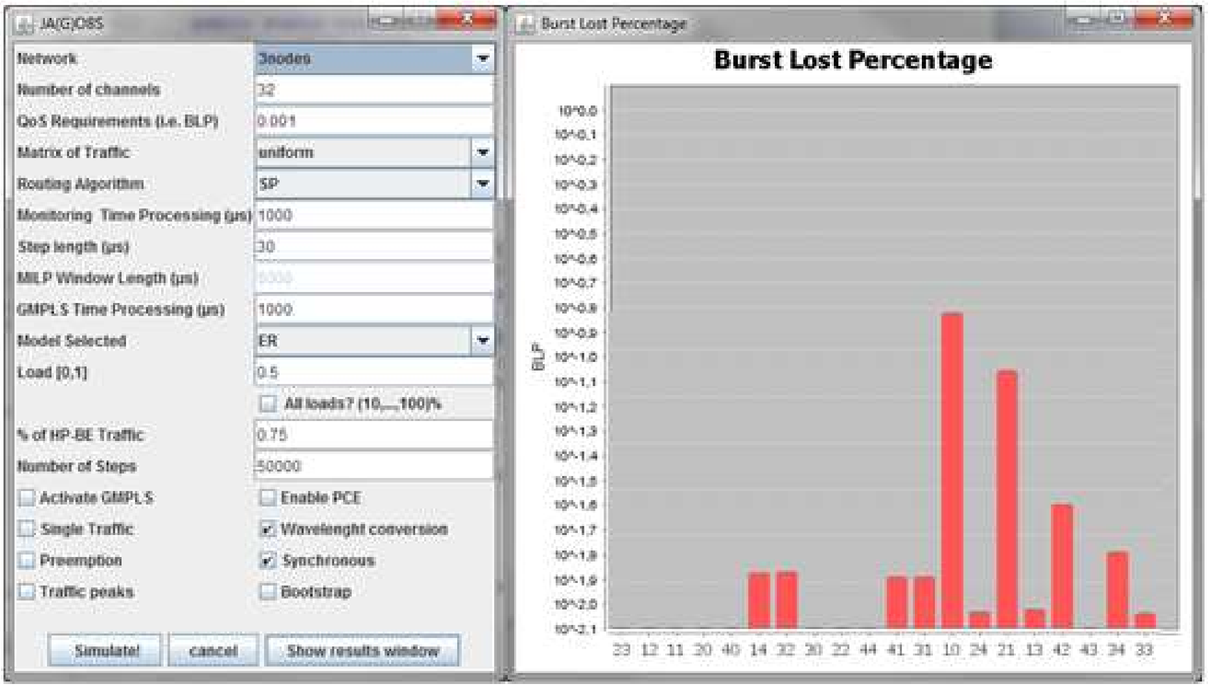
```
----------------------------------------------------------------------
--Signaling Message Received at PCE node from GMPLS node:1004
  Sending Path Confirmation Msg btw 1004 and 1002
        Msg id : 0
        Arrival time: 14125.mcs
----------------------------------------------------------------------
--Signaling Message Received from PCE at GMPLS node:1004
  Sending Path Confirmation Msg to 1002
        Msg id : 0
        Arrival time: 14459.mcs
----------------------------------------------------------------------
-----Reception of a PATH message in node 1002 -------
        -Sending a RESV message to: 1004
----------------------------------------------------------------------
-----Reception of a RESV message in node 1004 -------
        -Init the Routing Process: LS-UPdate Msg flooding
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1004
        - msg initial source:1004
        - Msg arrival time: 17792.mcs
                ->transmission to 1001
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1003
        - Msg id:0
        - Last hop:1004
        - msg initial source:1004
        - Msg arrival time: 17792.mcs
                ->transmission to 1000
                ->transmission to 1001
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1003
        - msg initial source:1004
        - Msg arrival time: 19125.mcs
                ->transmission to 1001
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1002
        - msg initial source:1004
        - Msg arrival time: 19125.mcs
                ->transmission to 1000
                ->transmission to 1003
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1003
        - msg initial source:1004
        - Msg arrival time: 18459.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1001
        - msg initial source:1004
        - Msg arrival time: 20459.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1000
        - msg initial source:1004
        - Msg arrival time: 20459.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
--(Routing) LS-Update Msg message received at GMPLS node:1003
        - Msg id:0
        - Last hop:1001
        - msg initial source:1004
        - Msg arrival time: 19792.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
```

Figure 17 – JA(G)OBS output when PCE is active

In the first step we can see the exchange of messages occurring between node 1004 (node E of Fig.16) and the PCE node. First, node 1004 asks for a path and PCE node re-sends the same message but with the information in it. However the node has to confirm this path with the receiving node, which is node 1002. So in the second step we can see that node 1002 successfully receives the signaling message and responds to it with a RESV message confirming that the path is still available. The other steps follow the same sequence as explained in the section above (Routing implementation), i.e. the node 1004 starts to send LSU messages to its neighbors. It should be noticed that this was a direct path, i.e. the path had only one node in the way (node 1001). If the path had more than that, the routing messages would start in *n-1* of the path (considering that the path has *n* nodes).

3.4 Graphical User Interface

One of our goals was also to develop a Graphical User Interface (GUI) so that all users who want to use the simulator can do it smoothly without any effort. The GUI is based on Java Swing libraries because the entire simulator is based on Java. This is preferable rather than adding more dependencies to the simulator. For example, if the GUI was written in GTK+ [38] it would be richer graphically speaking and lighter in terms of memory consumption. However it would have many more dependencies that would have to be installed in the computer for the GUI to run properly. Since this was not the purpose of the project, we focused on the general solution since Java can run on many platforms (Linux, Windows©, MacOS© and Solaris). In this case, we just needed to install Java Virtual Machine, JFreechart [26] libraries and the JAVANCO Libraries for the simulator to work. The GUI is composed by two windows as shown in Fig.18.



Network	3nodes
Number of channels	32
QoS Requirements (i.e. BLP)	0.001
Matrix of Traffic	uniform
Routing Algorithm	SP
Monitoring Time Processing (μs)	1000
Step length (μs)	30
MILP Window Length (μs)	5000
GMPLS Time Processing (μs)	1000
Model Selected	ER
Load [0,1]	0.5
	☐ All loads? (10,...,100)%
% of HP-BE Traffic	0.75
Number of Steps	50000
☐ Activate GMPLS	☐ Enable PCE
☐ Single Traffic	☑ Wavelenght conversion
☐ Preemption	☑ Synchronous
☐ Traffic peaks	☐ Bootstrap

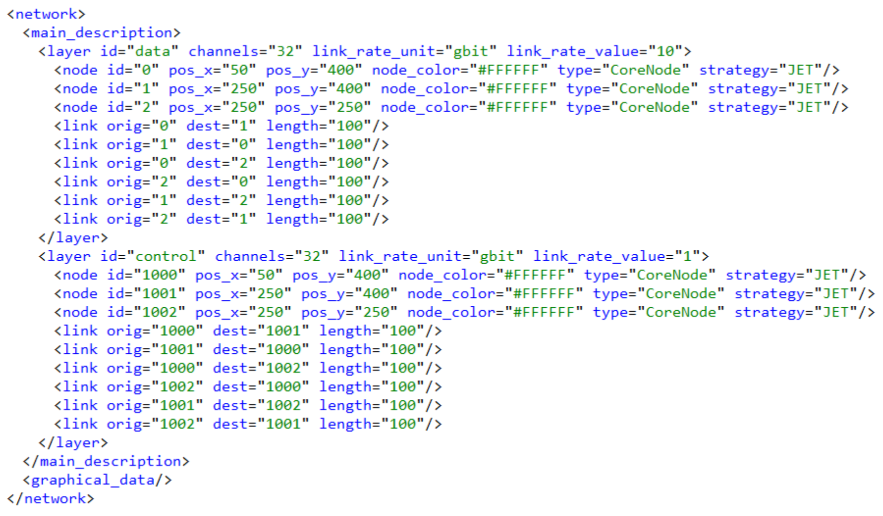
Figure 18 – Graphical User Interface of JA(G)OBS

The left window is composed by:

Network	This combo box gets the name of the XML files in the directory *\JAVANCO\default_graphs*. The network configuration is contained in these files. Figure 19 shows an example showing how the file is organized. The introduction of information on the file is crucial to achieve a good simulation. If the parameters were not introduced correctly in the XML file the simulator could malfunction.
Number of channels	The user can define on this field how many channels (λ´s) are available per fiber.
QoS Requirements (i.e. BLP)	The user will introduce on this field the burst lost probability he/she wants to have in the network.
Matrix of Traffic	This field uploads what type of traffic matrix the user wants: Uniform or Non-uniform.
Monitoring Time Processing	This is processing time of the GMPLS to the physical layer.
Step Length	This field coordinates the Step time of the simulator. The simulator step time is how much time each node has to execute a task before the CPU time has to pass to another node. This time has to be the shortest time defined in the simulator.
MILP Window Length	This is the time that the PCE node has to aggregate various requests using the MILP algorithm.
GMPLS Time Processing	This is the amount of time the GMPLS node takes to process each message.
Model Selected	In this combo box the user can choose the different kinds of models the simulator has to offer.
Load	In this parameter the user can define how much load he/she wants in the network. If he/she wants to simulate more than one load, the only thing that he/she needs to do is to click on the *checkbox* all **loads** and the simulator will simulate the loads from 10 to 100% with 10% increments.
% of HP-BE Traffic	This lets the user introduce the amount of ratio he/she wants for High Priority Traffic. Thus, the rest will be channeled to Best Effort Traffic.
Activate GMPLS	Since the simulator can only use an OBS layer, the user can choose between simulating the two layers, GMPLS-OBS or just one OBS.
Enable PCE	In this checkbox the user has the opportunity to choose if he/she wants the centralized Path Computation Element or not. If this check box is selected, the **Activate GMPLS** is automatically activated. It is assumed that the network needs a control layer to communicate with the PCE. Also when the PCE is enable the D-MILP protocol can be activated by the user.

Single Traffic	This field inserts only one kind of message without the knowledge of HP or BE.
Wavelength Conversion	This check box gives the user the opportunity to choose if the OBS node has the ability to convert a wavelength if needed to complete a path.
Preemption	This field when selected allow for HP traffic to be inserted in wavelengths that were reserved for BE traffic.
Synchronous	This field makes the traffic generation synchronized or not with the step.
Traffic Peaks	This forces the simulator to allocate traffic peaks in a certain period of time in a group of links.
Bootstrap	This allow for the configuration of the network be by bootstrap.
Simulate!	Start simulator.
Cancel	Shutdown button
Show results window	This button shows the other window with the results.

The right window shows a graphic bar with the burst lost probability (BLP) by LSP. All the other results gathered from the simulator are saved in text files that can be used by the user without any problem or specific program.



```xml
<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="JET"/>
      <node id="1" pos_x="250" pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JET"/>
      <node id="2" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JET"/>
      <link orig="0" dest="1" length="100"/>
      <link orig="1" dest="0" length="100"/>
      <link orig="0" dest="2" length="100"/>
      <link orig="2" dest="0" length="100"/>
      <link orig="1" dest="2" length="100"/>
      <link orig="2" dest="1" length="100"/>
    </layer>
    <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="JET"/>
      <node id="1001" pos_x="250" pos_y="400" node_color="#FFFFFF" type="CoreNode" strategy="JET"/>
      <node id="1002" pos_x="250" pos_y="250" node_color="#FFFFFF" type="CoreNode" strategy="JET"/>
      <link orig="1000" dest="1001" length="100"/>
      <link orig="1001" dest="1000" length="100"/>
      <link orig="1000" dest="1002" length="100"/>
      <link orig="1002" dest="1000" length="100"/>
      <link orig="1001" dest="1002" length="100"/>
      <link orig="1002" dest="1001" length="100"/>
    </layer>
  </main_description>
  <graphical_data/>
</network>
```

Figure 19 – 3 node network configuration XML file

3.5 Performances Analysis

In this subsection is presented the Central Processor Unit (CPU) and memory consumption in the simulator. These tests were done in an AMD© Phenom© II X4 945 processor with 4 GB of RAM in a Microsoft© Windows© 7 64 bit operating System. The simulations were done with 10000 Steps of the simulator, with two different algorithms: 1) Shortest Path and 2) K-Shortest Path (with K=2) on two different networks: 1) German50 Fig. 20 and 2) NSFNET Fig. 21. In the Table 1 we can see main specifications of the networks.

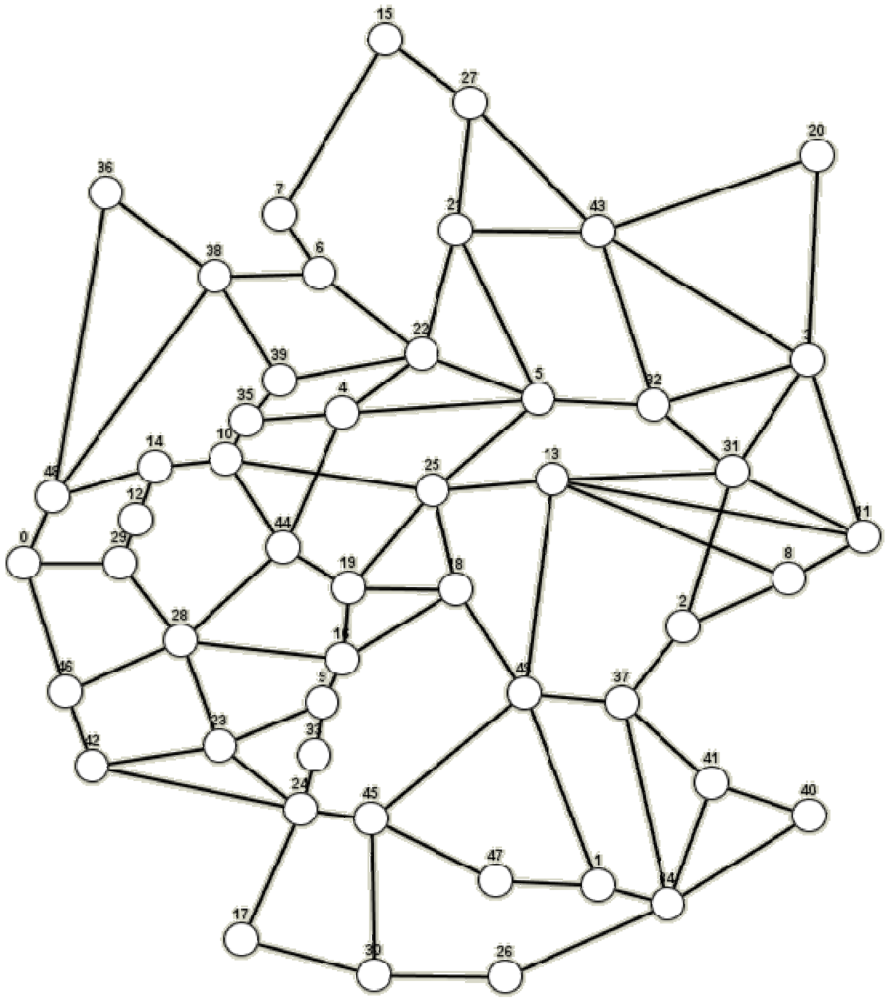


Figure 20 – German 50 network topology

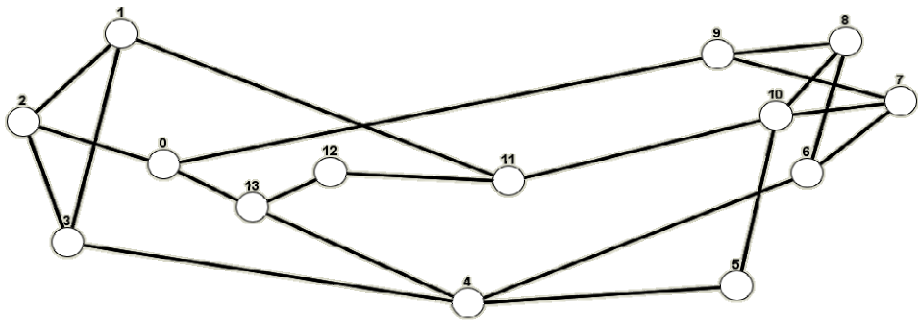


Figure 21 – NFSNET network topology

Name	Number of Nodes	Number of Data Links	Number of Control Links
NSFNet	14	42	42
GERMAN50	50	176	176

Table 1 – Network Specifications

CPU Consumption

The CPU consumption over the time is presented in Fig. 22 and 23. The Shortest Path algorithm is used in the two different networks.

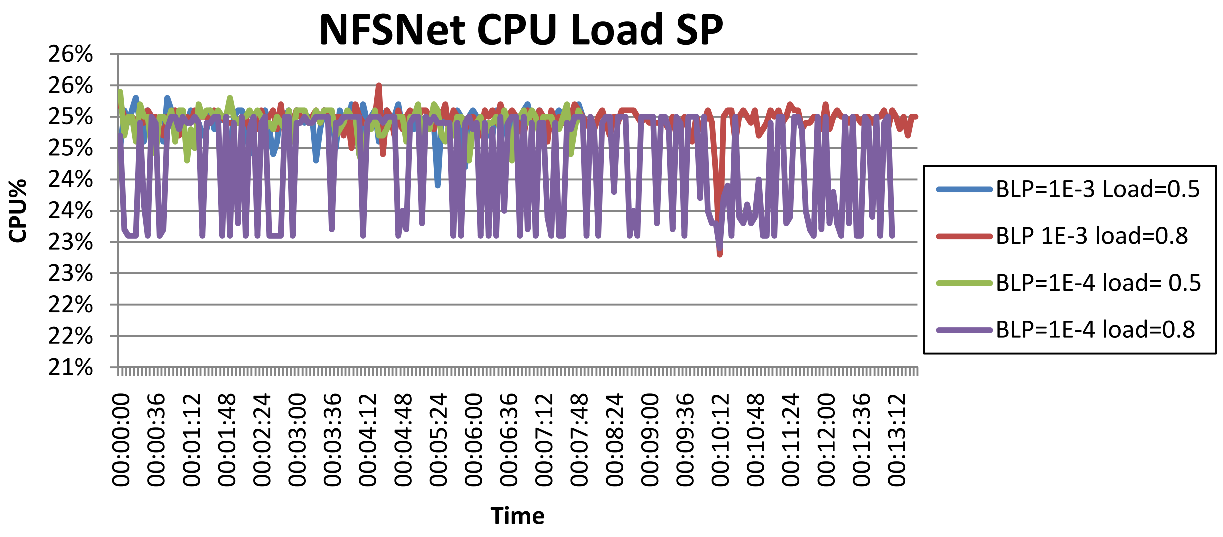


Figure 22 – CPU load in NFSNet with SP

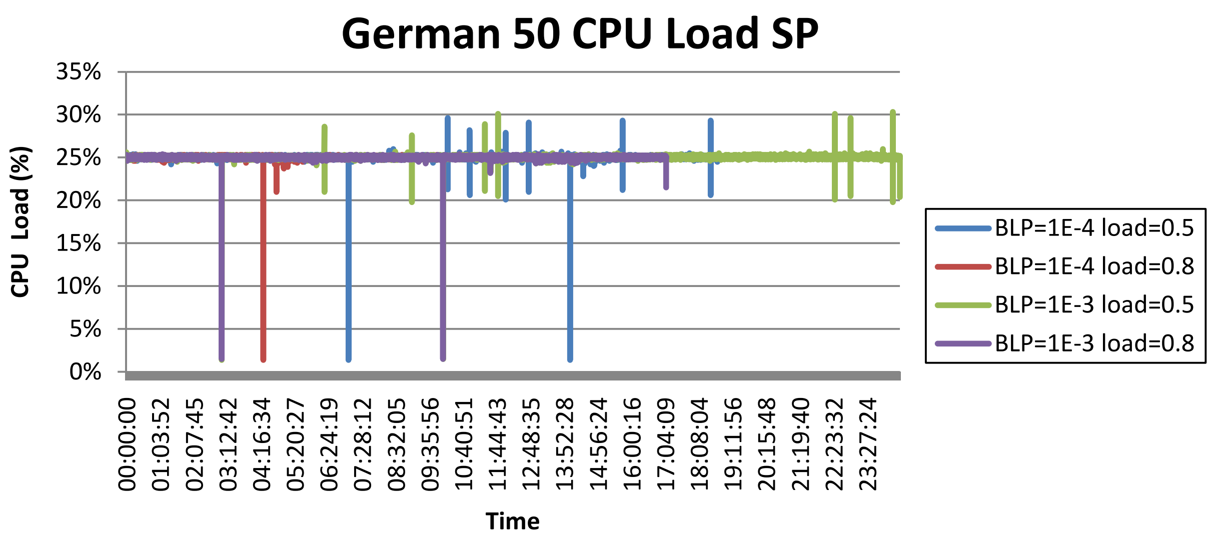


Figure 23 – CPU load in Germa50 with SP

In these two figures it is shown that the simulator oscillates near 25% of the CPU load, which is good because if the system has enough memory the user can run simulations in parallel to save time. Another thing that is noticeable is that the simulation time of German50 network is longer than the NFSNet, German50 toke 23 hours and 13 minutes and NFSNet only toke 13 minutes. This fact can be explained because there are 3.5 times more connections and 26 more nodes in the German50. So, the simulator has to make much more calculations per node, making it much more time consuming. However this simulation proves that the simulator only uses a quarter of the CPU load, even when the simulations are very time consuming. So one conclusion made is that the simulator does not use more CPU load when the simulations are long. In appendix D is shown the graphics for the K-SP simulations.

Memory Consumption

Figure 24 and Fig. 25 it is presented the results of memory consumption in the two networks using the Shortest Path Algorithm and with a Burst Lost of 10^{-3}.

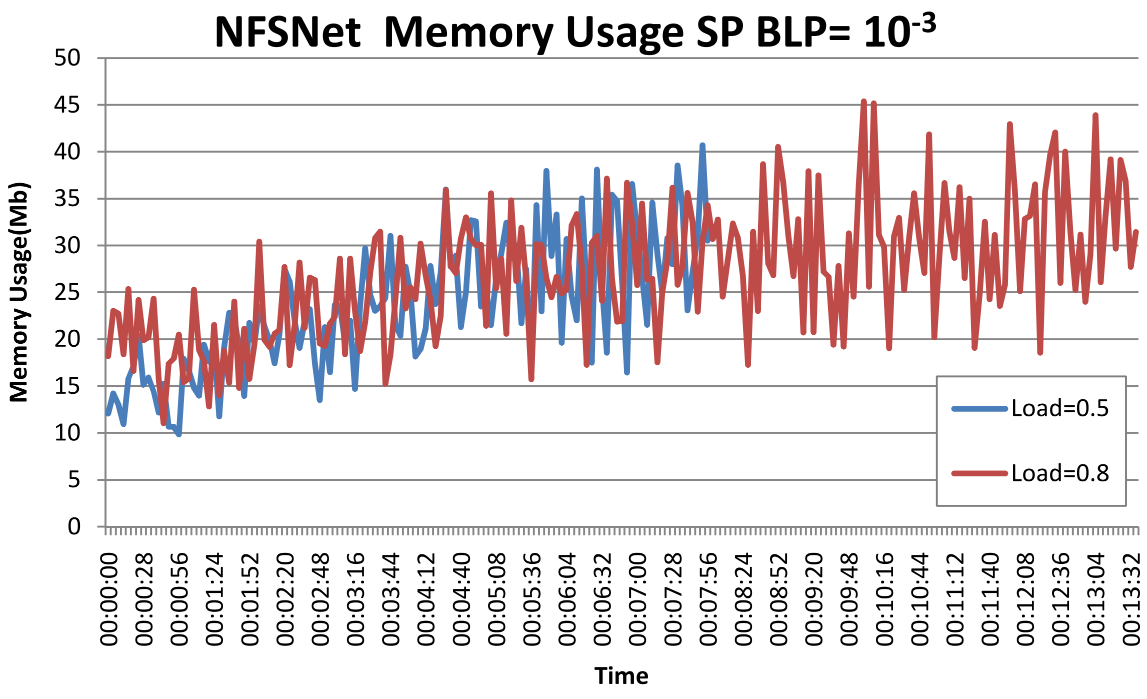


Figure 24 – Memory Consumption in NFSNet with BLP= 10^{-3} SP algorithm

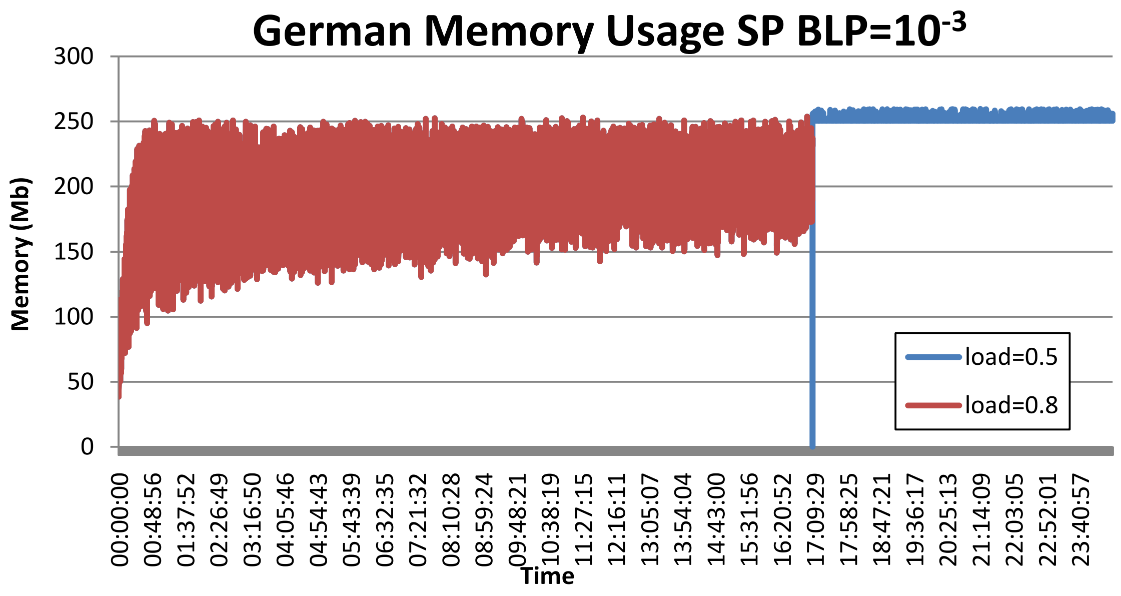


Figure 25 – Memory Consumption in German50 with BLP= 10^{-3} SP algorithm

Figure 24 it is shown that the graphic curve is not stable, as it has a lot of peaks. This happens because the Java Garbage Collector cleans the memory without us giving that command, as expected in Java programming. However, the simulator doesn't go above the 45 megabytes of memory used, which for a program developed in Java is very low memory usage.

Nevertheless, in Fig.25 it is shown that the memory usage increases when the number of nodes also increases. This is normal, because there are more nodes, links and classes loaded in the memory. Thus, an average of 250 megabytes of memory usage is observed in the German50 network. The curiosity of Fig.25 is that the Garbage Collector is cleaning less and less the memory over the time. In this case, the garbage collector cannot clean the objects from the memory because there aren't any objects being discarded, i.e. all objects are being used in the simulation. So it can be concluded that the memory usage of the simulator is low, taking in account that is programmed in Java. Also that, in this case, the Garbage Collector is doing what is supposed to do, cleaning the unnecessary objects from the memory. In the appendix D it is shown the graphics for the BLP = 10^{-4} using both the SP and the K-SP algorithms.

Network	Algorithm	BLP	Load	Steps	Simulation Time	Max MEM Used	Min MEM used
NFSNet	SP	0,0001	0,5	10000	0:08:00	40	9
NFSNet	SP	0,0001	0,8	10000	0:13:21	44	13
NFSNet	SP	0,001	0,5	10000	0:08:03	41	10
NFSNet	SP	0,001	0,8	10000	0:13:45	45	11
NFSNet	K-SP	0,0001	0,5	10000	0:08:04	41	11
NFSNet	K-SP	0,0001	0,8	10000	0:12:13	49	15
NFSNet	K-SP	0,001	0,5	10000	0:09:13	40	10
NFSNet	K-SP	0,001	0,8	10000	0:12:45	47	12
German50	SP	0,0001	0,5	10000	13:21:29	260	33
German50	SP	0,0001	0,8	10000	18:43:35	260	40
German50	SP	0,001	0,5	10000	24:30:01	260	5
German50	SP	0,001	0,8	10000	17:06:51	254	38
German50	K-SP	0,0001	0,5	10000	26:02:00	250	4
German50	K-SP	0,0001	0,8	10000	20:23:56	255	19
German50	K-SP	0,001	0,5	10000	32:11:50	259	17
German50	K-SP	0,001	0,8	10000	25:37:05	260	20

Table 2 – Table of results of the simulations

In Table 2, the summary of the performance results of the simulations is shown. It can see that the memory usage is similar in all the simulations made. Also it is shown that the K-SP algorithm is generally more time consuming in the German50 network, this happens because K-SP algorithm has to do more calculations than the SP to dispatch a burst. This is not noticeable in the NFS network because it has fewer connections between the nodes, therefore doing fewer calculations.

3.6 Related Work

In this section, it is described some OBS-related and GMPLS-related simulators, namely 1) OBS-ns [35], 2) NCTUns [34] and 3) OBS model for OMNeT++ [22] and 4) GLASS [11].

OBS-ns

The OBS-ns simulator is an extension of the Network Simulator 2 (ns-2) simulator. OBS-ns is an event-driven simulator that is built on ns-2. Because of this, it is still necessary to write a script on OTcl to specify all the parameters in the network. This simulator introduced some extensions to the script to implement the following features:

- Fiber-Delay-Lines (FDL);
- The size of the Burst Header Packet (BHP);
- Burst Control Packet (BCP);
- Timeout specifications.

The simulation output is organized in statistic files and trace files that can be read with any text editor. One of the pros of this simulator is the interoperability between a Nam animator [40] that can read the output files and create an animated GUI of the network state.

The cons are:

- All the code was written in C++, this adds complexity for developers and users that want to install it.
- The study in [41] was unable to ascertain what type of resource reservation was implemented in it.

There is no information whatsoever regarding the possibility of GMPLS or any CP being deployed.

NCTUns

The NCTUns is a more mature simulator, since it is on version 4.0. It was implemented to be a simulator and an emulator of different kinds of networks. The OBS network is supported by different modules that are included in the simulator. The user can specify the behavior of the nodes, burst assembly, wavelength channel and conversion, control packet processing time and contending burst algorithm.

Another pro of this simulator is the GUI that allows the user to construct and configure his/her models. The GUI can also do an animation of the packet transfer which is very useful for the user to see what his/her work is doing in a more user friendly way.

The cons are:

-Being written in C++;

-Being difficult to install in a machine [41];

-There is only one Reservation Scheme (JET), so the user cannot see the difference in the performance between two different reservation schemes.

However the GUI and the adaptability make it a good tool to make some experiences in OBS networks.

OBS model for OMNeT++

As the authors say in [22], OMNet++ is not a simulator by itself, but more of a framework with tools to make any kind of simulator, in their case an OBS simulator. The structure of the simulator is explained in [22], but it basically consists of two types of nodes: 1) the Edge node that is based on the Router module of OMNet++ and has to assembly module to convert packets in to bursts and disassemble modules to do the opposite task. 2) The Core node basically is only responsible for the routing of the packets in an optical way between the sender and the receiver.

The simulator is made up of modules, this is why it very interesting because other modules can be implemented almost effortlessly. In our point of view this is a very interesting characteristic of this simulator, it is also highlighted that the user can configure each node separately, this enables the simulation of different network configurations.

The drawbacks of this simulator are:

- It only has implemented JET for resource reservation, despite of this the authors claim that JIT can be easily introduced in the simulator.
- The forwarding table is static, i.e. the routing is done always on the same fiber and does not take to account the conditioning of the network;
- It was implemented in C++.

Finally it should be stated that the simulator uses the proposal made by [19] that uses labels to the forwarding of the bursts.

GMPLS Lightwave Agile Switching Simulator

The GLASS (GMPLS Lightwave Agile Switching Simulator) has been developed to support the R&D work in the field of Next Generation Internet (NGI) networking with GMPLS-based WDM optical network and Internet traffic engineering with DiffServ-over-MPLS [11]. The GLASS was implemented on the Scalable Simulation Framework (SSF) [42].

SSF framework was implemented to be a discrete event simulation platform. It provides an interface for programmers to create simulators avoiding the problems of threads and synchronization. Also SSF provides a tool, used in GLASS, which is called SSFNet [42]. This makes tools available for network simulation to programmers, i.e. allows the programmer implement to protocols like IP, TCP, OSPF and others out of the box. Despite the fact that SSF has an implementation of OSPF, it had to be upgraded in GLASS because SSF only supports static OSPF and does not support Traffic Engineering (TE) features. For this reason, in GLASS the algorithm was upgraded to handle TE, also other algorithms were developed from scratch to handle the features documented in [11]:

1) Differentiated service (DiffServ): This has the ability to differentiate the process of a packet that arrives in the router, i.e. depending on the type of packet the processing is a different processing. GLASS defined 4 categories of traffic and processes the packets depending on the category, so the packets can be queued right away, buffered (giving space

to more priority packet) or in the worst case scenario, dropped;

2) Per hop behavior (PHB): Because the essence of GMPLS is that every node has to decide for itself what it is supposed to do. So the DiffServ together with the PHB algorithm decides what the destiny of the packet is queue, buffer or dropped;

3) GMPLS-TE: In GLASS two signaling protocols were implement the Constraint-based Routing Label Distribution (CR-LDP) and the RSVP-TE. In [11] doesn´t explain how these protocols are implemented, only that the Type-Length-Value (TLV) has many different types of metrics. For the MPLS routing part, it was only modified the OSPF of SSFNet to OSPF-TE as mentioned early;

4) MPLA in OAM for performance monitoring and fault restoration.

5) and GMPLS-based signaling for WDM optical network, link/node failure model and fast restoration from link or node failure.

GLASS also has LMP which supports the control channel management, link property correlation and link connectivity verification. The LMP establishes and maintains the control channels connectivity between neighboring nodes by exchanging hello protocol packets for fast keep-alive, control channel availability and status monitoring. This feature is still not supported in JA(G)OBS. Also an important aspect of this simulator is the way the data is inserted in the simulator, it uses Domain Modeling Language (DML) which is a standard of data files. Also in the last versions of GLASS a GUI was included, with this GUI networks can be built effortlessly, which in my point of view, is a plus. This fact allied output files ready, out of the box, for Microsoft Excel or OpenOffice Calc makes it a very good I/O of data in the simulator when compared with the other simulators.

The only problem in GLASS is the following:

1) The project seems to be abandoned. Since it only fully works in Java 1.4 and with some changes in the code works on 1.5. This is a big problem because at this time Java 7(1.7) is almost ready and the simulator users must know how to do Java programming to make these changes.

In summary GLASS is a very good simulator for GMPLS in WDM because it is easy to use and has almost every tool needed.

Chapter IV - Conclusion

This thesis discusses the interoperability between GMPLS and OBS control plane. It presents the two technologies separately and the challenges and benefits of putting the two working together.

However, this thesis gives more focus on GMPLS routing algorithm (OSPF-TE), since it was the main contribution made to JA(G)OBS.

Regarding the simulator, it was shown how LSUs messages from OSPF-TE were successfully implemented and also how JA(G)OBS is a versatile simulator due to its modularity. In addition, the implementation of the PCE, was also done which allowed us to simulate two different policies (distributed and centralized path computation) in the same simulator.

A series of performance tests were also presented, showing that JA(G)OBS can cope with various network sizes (in terms of nodes and links) without using a large amount of computational resources, as shown in Chapter III (Performance Analysis).

As such, it is concluded that JA(G)OBS is a robust and viable tool that can handle a GMPLS-OBS network without much effort, making it a good choice for this kind of network. Other conclusion that was made is that GMPLS-OBS is a good solution for the current and future networks, since it accomplishes the necessary requirements for a network.

Despite of being a robust and reliable simulator, JA(G)OBS is still not finished. Our goal was not to make all features operable right away, but instead make a reliable simulator with fewer features. Regarding future work, the simulator will benefit if the following key features are implemented:

1. Implementation of the remaining four OSPF-TE packets, namely Hello packet, Database Description packet, Link State Request and Link State Acknowledgment packet. This will add more features to the simulator that aren't supported in the current version and could be interesting to study the full behavior of OSPF-TE in this architecture.
2. Implementation of a Pareto distribution [43], [44] algorithm. This will make the traffic in the simulator bursty-based inside the network and give a different approach to the simulations based on the Engset probability of congestion formula [52], [53].

References

[1] Keping Long *et al*., "A GMPLS-based OBS Architecture for IP-over-WDM Networks, Network architectures, management, and applications No4 , COREE, REPUBLIQUE DE (2006), pp 63540H.1-63540H.10, September 2006 .

[2] T. Battestilli *et al.*, "Introduction to Optical Burst Switching", Communications Magazine, IEEE, pp S10-S15, August 2003.

[3] P. Pedroso *et al*., "Integrating GMPLS in the OBS Networks Control Plane", Transparent Optical Networks, 2007. ICTON '07. 9th International Conference, pp 1-7, August 2007

[4] S. A. Tanenbaum, "Computer Networks, 4th edition", 2003.

[5] C. Hedrick *et al.*, "Routing Information Protocol", RFC 1058, June 1998.

[6] L. Berger *et al*." Generalized Multi-Protocol Label Switching Signaling Functional Description", RFC 3471, January 2003.

[7] R. Coltun, "The OSPF Opaque LSA Option", RFC 2370, July 1998.

[8] J. Moy *et al*., "OSPF Version 2", RFC 2328, April 1998.

[9] P. Pedroso *et al.*, "An interoperable GMPLS/OBS Control Plane: RSVP and OSPF extension Proposal", Communication Systems, Networks and Digital Signal Processing, 2008. CNSDSP 2008. 6th International Symposium, pp 418-422, July 2008.

[10] O. Pedrola *et al*., "JAVOBS: A Flexible Simulator for OBS Network Architectures", Journal of Networks, Vol.5, pp 256-264, February 2008.

[11] Y. Kim *et al*., "GLASS (GMPLS Lightwave Agile Switching Simulator)- A Scalable Discrete Event Network Simulator for GMPLS-based Optical Internet", August 2002.

[12] M. Yoo *et al*, "Just-Enough-Time (JET): A High Speed Protocol for Bursty Traffic in Optical Networks", IEEE/LEOS Technologies for a Global Information Infrastructure, Vertical-Cavity Lasers, Technologies for a Global Information Infrastructure, WDM Components Technology, Advanced Semiconductor Lasers and Applications, Gallium Nitride Materials, Processing, and Devi, pp 26-27, August 1997.

[13] S. J. Ben Yoo, "Optical Packet and Burst Switching Technologies for the Future Photonic Internet", Journal of Lightwave Technology, VOL. 24, pp 4468- 4492, December 2006.

[14] Z. F. Syahid *et al.*," Comparison of JET and JIT Protocols in OBS Networks with Bursty Internet Traffic ", the 4 Th International Conference TSSA, pp 54-58, December 2007.

[15] F. Xue *et al.*, "Performance Comparison of Optical Burst and Circuit Switched Networks", Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC, pp 3-6, March 2005.

[16] A. Farrel *et al*. "GMPLS: architecture and applications", 1st edition, 2005.

[17] A. Manolova *et al.*, "The OBS Control Plane: GMPLS Integration or Not?", IX Workshop in G/MPLS Networks, Girona Spain, 2010.

[18] H. Guo *et al.*, “Proposal of a multi-layer network architecture for OBS/GMPLS network interworking”, Network Architectures, Management and Applications V, SPIE, November 2007.
[19] C. Qiao *et al.*, “Labeled Optical Burst Switching for IP-over-WDM Integration”, IEEE Communications magazine, Vol. 38 No. 9, pp 104-114, September 2000.
[20] A. Manolova *et al.*, ”Advantages and Challenges of the GMPLS\OBS Integration”, VI GMPLS Workshop, Girona, Spain, pp 133-144, April 2007.
[21] R. Martínez, “Experimental GMPLS-based routing for dynamic lightpad provisioning and recovery in all-optical WDM networks”, PhD dissertation, Universitat Politècnica de Catalunya, April 11, 2007.
[22] F. Espina, “OBS network model for OMNeT++: A performance evaluation”, SIMUTools’ 2010 Proceedings of 3rd International ICST Conference on Simulation Tools and Techniques, Article No. 18, March 2010.

[23] Information Sciences Institute University of Southern California, “Transmission Control Protocol”, RFC 793, September 1981.

[24] J. Y. Yen, “Finding the K Shortest Loopless Paths in a Network”, Management Science, Vol. 17, No. 11, Theory Series, pp. 712-716, July 1971.

[25] “K-shortest-paths”: http://code.google.com/p/k-shortest-paths/#Implementation, accessed at May, 2011.

[26] “JFreeChart” : http://www.jfree.org/jfreechart/index.html, accessed at May, 2011
[27] D. Katz *et al*, “Traffic Engineering (TE) Extensions to OSPF Version 2”, RFC 3630, September 2003.
[28] R. Aggarwal, “Extensions to Resource Reservation Protocol - Traffic Engineering (RSVP-TE) for Point-to-Multipoint TE Label Switched Paths (LSPs), RFC 4875, May 2007.
[29] J. Postel, “Internet Control Message Protocol DARPA Internet Program Specification”, RFC 792, September 1981.
[30] J. Baião, “GMPLS-Controlled OBS Network Simulator: Implementation of signaling protocol”, Master Dissertation, Instituto Superior das Ciências do Trabalho e da Empresa – Instituto Universitário de Lisboa, October 2010.
[31] OSPF Packet Types: http://sites.google.com/site/amitsciscozone/home/important-tips/ospf/ospf-packet-types , accessed at September 2010.
[32] E. Mannie *et al.*, “Generalized Multi-Protocol Label Switching (GMPLS) Architecture”, RFC 3945, October 2004.
[33] P. Pedroso , “ Interoperable GMPLS/OBS Control Plane: Functional Architecture and Protocol Extensions Proposal”, Master dissertation, Universitat Politècnica de Catalunya, July 2008.
[34] SimReal Inc., "NCTUns", http://nsl10.csie.nctu.edu.tw/, accessed at March 2011.
[35] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/, accessed at March 2011.

[36] E. Rosen *et al.*, “Multiprotocol Label Switching Architecture”, RFC3031, January 2001

[37] D. Awduche *et al.*, “Requirements for Traffic Engineering Over MPLS”, RFC 2702, September 1999.

[38] “GTK+” : http://www.gtk.org/ , accessed at May 2011.

[39] S. Rumley *et al.*: Software tools and methods for research and education on optical network, in COST action 291 final report, in press.

[40]”Nam animator”: http://www.isi.edu/nsnam/nam/, accessed at May 2011.

[41] V.Soares *et al.*, “OBS Simulation Tools: A Comparative Study”, Communications Workshops, 2008. ICC Workshops '08, pp. 256-260, May 2008.

[42]” Scalable Simulation Framework”: http://www.ssfnet.org/homePage.html, accessed at May 2011.

[43] James W. Stoutenborough, Paul Johnson, "Pareto Distribution", April 17, 2006, http://pj.freefaculty.org/stat/Distributions/Pareto-02.pdf .

[44] Morris H. DeGroot, “Optimal Statistical Decisions”, pp. 41, 2005.

[45] Lu Shen *et al.*, “Centralized vs. distributed connection management schemes under different traffic patterns in wavelength-convertible optical networks”, Communications, 2002. ICC 2002. IEEE International, August 2002.

[46] P. Pedroso *et al.*, “A GMPLS/OBS Network Architecture Enabling QoS-aware End-to-End Burst Transport”, 12th IEEE International Conference on High Performance Switching and Routing to be held in Cartagena, Spain, July 4 - 7, 2011.

[47] JL. Le Roux *et al.*, “Path Computation Element (PCE) Communication Protocol (PCEP)”, RFC 5440, March 2009.

[48] J. Teng *et al.*, “A Comparison of the JIT, JET, and Horizon Wavelength Reservation Schemes on A Single OBS Node”, Proc. of the First International Workshop on Optical Burst Switching, San Francisco, December 2003

[49] Z. Rosberg *et al.*, ”Blocking probabilities of optical burst switching networks based on reduced load fixed point approximations”, in Proc. IEEE Infocom 2003, San Francisco, CA, March 2003.

[50]C. Qiao and M. Yoo, "Optical Burst Switching (obs) - a new paradigm for an optical internet", Journal of High Speed Networks, vol.8, no. 1, pp. 69-84, March 1999.

[51] D. Awduche *et al.*,” RSVP-TE: Extensions to RSVP for LSP Tunnels”, RFC 3209, December 2001

[52] J. Boucher, “Traffic System Design Handbook: Timesaving Telecommunication Traffic Tables and Programs”,1st edition ,1993

[53] A. Zalesky *et al.*, “Engset formula for bufferless OBS/OPS: when is and when isn’t lengthening the off-time redundant?”, Global Telecommunications Conference, December 2009

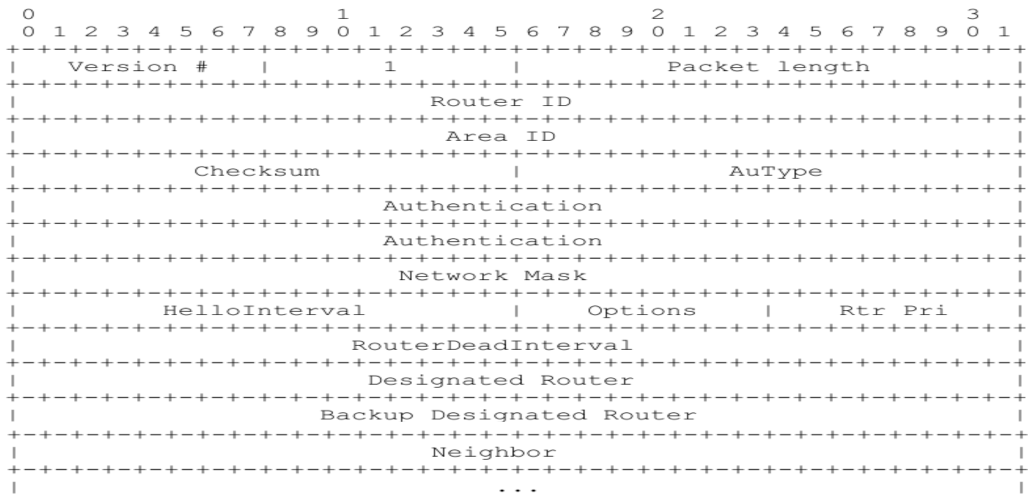
Appendixes

A – Types of Packets in OSPF

A.1. Hello Packet

Figure 26 illustrates the structure of the Hello packet and the name of each one of the fields.

When a hello packet is received, the node has to validate the Network Mask, *HelloInterval* and *RouterDeadInterval* before the actual processing of the packet, only if this fields match those previously agreed is the packet accepted. In case the packet is not rejected, the rest of the packet will be examined to see if there is any update on the information in comparison for one that is in the list. In case there is an update, the information will be stored in the database and sent to the other adjacent nodes, this will readjust the timer that is connected to receiving/sending of hello packet. The figure 26 shows the configuration of the Hello packet. This packet isn't implemented in the simulator because it is assumed that the network is already connected and there are no nodes connecting when the network is operating. In a future work this feature can be implemented.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |       1       |         Packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Router ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Area ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Network Mask                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         HelloInterval         |    Options    |    Rtr Pri    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     RouterDeadInterval                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Designated Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Backup Designated Router                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Neighbor                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

Figure 26 – Hello Packet [8]

Description of the several message fields:

Message Field	Description
Version	Field where the version of the packet is inserted.
Type	This field is where the type of packets is been transmitted, there are 5 types of OSPF packets; 1(Hello), 2(Database description), 3(Link State Request), 4(Link State Update) and 5(Link State Acknowledgement).
Packet Length	The field where the total size of the packet is inserted, it makes sense to put the size of the packet which can be variable, depending how much information is putted in there.
Router ID	The field where the ID of the node that created the packet is introduced.
Area ID	The area where the node belongs.
Checksum	This field is where a checksum travels to guarantee that the packet doesn´t have errors produced by the transmission of the packet.
AuType	The field where the type of authentication is inserted. There are 3 types of values in the authentication; 0(without password), 1(plain text password) and 2(Message-Digest algorithm 5 cipher).
Authentication	The field where the information to confirm the integrity of the packet is inserted.
Network mask	This field has the subnet mask of the advertising OSPF interface.
HelloInterval	This field has the time interval that Hello packet must be sent. This interval is by default in a broadcast network of 20 seconds.
Options	This field has the type of extra capabilities that the node can perform.
Rtr Pti	This field has the priority for the node to become a *Designated Router*, if this field is 0, the router is not capable to become a *Designated Router*.
DeadInterval	This field has the number of seconds that if the neighbor node doesn't respond is considered out of the network or *down*. The standard value in this field is 120 seconds in a broadcast network.
Designated Router	This field has the ID of the node that is the *Designated Router*, if there isn't one yet, the field should be 0.
Backup Designated Router	This field has the ID of the *Backup Designated Router*, this field like the previous is 0 if there is no *Backup Designated Router* chosen yet.
Neighbor	The Router IDs of all OSPF nodes from whom a valid Hello packet has been seen on the network.

A.2. Database Description Packet

Figure 27 illustrates the structure of the database description packet. When the packet is received it will be associated with a node to see if it is considered active or not, this process helps to discard packets with old information. Considering that the packet is accepted, the fields I, M, MS, Options and DD sequence number must be stored to be compared to future packets. This comparison is made because the packet isn’t immediately rejected if these fields are equal.

Once again this type of packet was not implemented. Since the simulator was working on event trigger, all the information was carried in Link State Update messages.

Since OSPF packets have the same 6 fields, and they were explained in the hello packet, it will not be explained in further OSPF packets.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |       2       |         Packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Router ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Area ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Interface MTU         |    Options    |0|0|0|0|0|I|M|MS
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     DD sequence number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                             -+
|                                                               |
+-                      An LSA Header                          -+
|                                                               |
+-                                                             -+
|                                                               |
+-                                                             -+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

Figure 27 – Database Description Packet [8]

Description of the several message fields:

Interface MTU	This field contains the size (in bits) that the packet could have.
Options	This field has the same purpose as the one in the Hello packet, to announce extra capabilities in the node.
I	This field has the information about the sequence of the packaging. If the packet is the first of a sequence, the field has the number 1.
M	This field lets the receiving node know if there are anymore packets of this sequence of Database Description after this one. The field should be 1 if there are more packets for the announce sequence.
MS	This bit indicates if the sending node is the *master* in this connection.
DD sequence number	This field is used to sequence the collection of Database packets. The initial value should be unique. The sequence number then increments by 1 until the complete database description has been used to sequence the collection of DBD packets. The initial value should be unique. The sequence number then increments by 1 until the complete database description has been sent.
LSA Header	This field has the *link state Advertisement* (LSA) header, which is where the information about the connections is stored. This topic will be further explained in other topics.

A.3. Link State Request Packet

The LSR_2 is sent having a specific piece of the database, so it is more efficient than having to send the all database once again. When the neighbor responds, the packet may not contain all the LSA that are needed, in this case the node will update the list of requests and send another LSR_2, the cycle will continue in intervals of RxmtInterval[3] until all LSA in the list are updated and removed of the list. The Fig. 28 illustrates the structure of the links state request packet. This type of packet is defined by the LS checksum, LS age and LS sequence number although these fields are not specified in the Link State Request Packet itself. The router may receive even more recent instances in response. [8]. When a node receives a LSR_2, it will process the packet and see what the link state agreement (LSA) or LSAs that has been requested and send them on a link state update packet

[3]RxmtInterva l - The number of seconds between LSA retransmissions, for adjacencies belonging to this interface. Also used when retransmitting Database Description and Link State Request Packets.[8]

(LSU). The packet should not be put on the retransmission list, if the connection fails this LSR_2 will not be sent again, the neighbor will ask again for the LSA if it didn't get it from another node. If the node that requested the LSA doesn't have it on the database it should produce a BadLSReq[4] and restart the adjacency again for a full share on the database once again.

Once more this type of packet wasn't implemented because of the same reasons of the Database Description Message.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Version #   |       3       |         Packet length         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          Router ID                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           Area ID                             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |           Checksum            |             AuType            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Authentication                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Authentication                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          LS type                              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Link State ID                           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     Advertising Router                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                              ...                              |
```

Figure 28 – Link State Request Packet [8]

Description of the several message fields:

LS type	This field has the type of *link state Advertisement* requested. Since there are 11 types of LSAs, the packet must specify the type it is expecting because each type of LSA carries different types of information, this aspect will be further explained better later on.
Link State ID	This field identifies the routing domain that is being described. This parameter can have five different options, to see each option consult [8].
Advertising Router	This field has the ID of the node that requested the information.

[4] BadLSReq - A Link State Request has been received for an LSA not contained in the database. This indicates an error in the Database Exchange process. [8]

A.4. Link State Update Packet

In the Fig. 29 shows the structure of the LSU

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |       4       |         Packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Router ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Area ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            # LSAs                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                            +-+
|                             LSAs                              |
+-                                                            +-+
|                              ...                              |
```

Figure 29 – Link State Update Packet [8]

Description of the several message fields:

# LSAs	This field has the number of *link state Advertisement* carried by the packet.
LSAs	This field is where the information about the network is sent.

A.5. Link State Acknowledgement Packet

Description of the several message fields:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Version #   |       5       |         Packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Router ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Area ID                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |             AuType            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-                                                             -+
|                                                               |
+-                         An LSA Header                       -+
|                                                               |
+-                                                             -+
```

Figure 30 – Link State Acknowledgment Packet [8]

LSA Header	This field has the LSA header we want to confirm by sending this message. _

A.6. Link State Advertisement Packet

Figure 31 shows the Header of an LSA that is equal in all LSAs, except those that who are opaque which have some special field, as described earlier.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LS age             |    Options    |    LS type    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Link State ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Advertising Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     LS sequence number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         LS checksum           |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 31 – Link State Advertisement [8]

Description of the several message fields:

LS Age	This field has the time in seconds when the LSA was originated. The field is very useful to confront two packets that have similar information and one is older than the other helps to decide which LSA to discard.
Options	This field specifies the options of the OSPF that the node can support.
LS Type	This field is very important because it is where the link state is announced. There are 11 types of LSA. Each LSA has its own separate advertisement type.
Link State ID	This field identifies the routing domain that is being described. This parameter can have five different options, to see each option consult [8].
Advertising Router	This field has the ID of the router where the packet is originated.
LS Sequence number	This field has the sequence number of the LSA, the field is used to detect old or duplicated LSA. The router has to check this field to guarantee that the LSA database is up to date.
LS Checksum	This field has the checksum of the contents of the LSA, except the LSA age. The checksum is used to detect errors that can occur in the transmission of the LSA. The algorithm that is used on the LSA is the Fletcher's checksum.
Length	This field has the size of the LSA. The length field is important because the LSA can contain a variable number of information, depending on the type of LSA and the number of information that the LSA carries.

A.6.1. Link State Advertisement – Type 1

This type of LSA is used when node announces itself to other nodes that share the same metrics. This type of LSA is flooded through the network. This type of LSA was not implemented because it was assumed that all nodes knew their neighbors. This simplification doesn't affect our results.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    0    |V|E|B|        0      |            # links            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Link ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Link Data                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     # TOS     |            metric             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      TOS      |        0      |          TOS  metric          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Link ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Link Data                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

Figure 32 – Link State Advertisement- Type 1 [8]

The Fig. 32 represents a link State Agreement when the Link state type value is 1, which means that is a Router-LSA type.

Description of the several message fields:

V	This parameter is set to 1 means that is the endpoint of a Virtual link.
E	This parameter is set to 1 when the router is a boundary node. It means it is the point of entry of connections on an Autonomous System (AS).
B	This parameter is set to 1 when the router is a border router that has an external connection with other network.
# Links	This parameter has the number of links described in the LSA. This parameter is important because the number of links varies and the router needs to know how much information there is to compute.
Link ID	This parameter identifies where the link is connect. This value depends on the link type, if the router is directly connected to the router that originated the LSA the link ID will have the same ID that is on the LSA header. If it´s not the case, then this parameter can have 4 values: 1- Neighboring router's Router ID; 2- IP address of Designated Router; 3- IP network/subnet number; 4- Neighboring router's Router ID.
Link Data	This value depends on the link's Type field. For connections to stub networks, Link Data specifies the network's IP address mask. For unnumbered point-to-point connections, it specifies the interface's MIB-II *ifIndex* value. For the other link types it specifies the router

	interface’s IP address. This latter piece of information is needed during the routing table build process, when calculating the IP address of the next hop. [8]
Type	This parameter describes the type of connection that is described in the LSA, the parameter can have values between 1 and 4 1- Point-to-point connection to another router 2- Connection to a transit network 3- Connection to a stub network 4- Virtual link
# TOS	This parameter has the TOs metrics given in a link, if there is no addition of metrics in a link this parameter is 0.
Metric	This parameter has the cost of using the link.

A.6.2. Link State Advertisement – Type 2

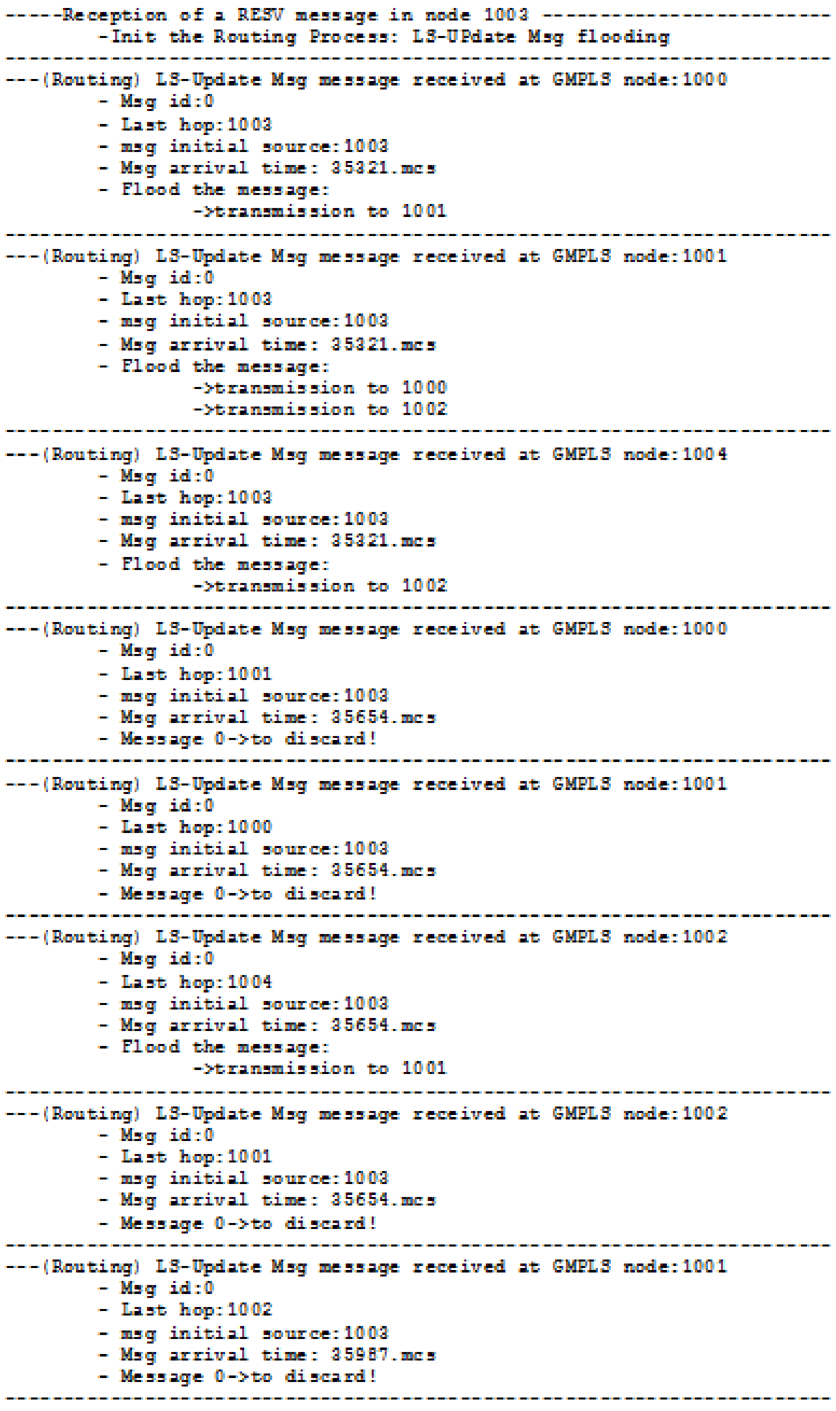
The Fig. 33 represents a link State Agreement when the Link state type value is 2, which means that is a network-LSA type. This type of LSA is generated for every broadcast for an area that has more than 2 routers. This network-LSA describes the routers that are connected, including the Designated Router which is the router that creates the LSA. This type of LSA wasn’t implemented for the same reasons mentioned in LSA-type1.

```
|                        Network Mask                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Attached Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
```

Figure 33 – Link State Advertisement- Type 2 [8]

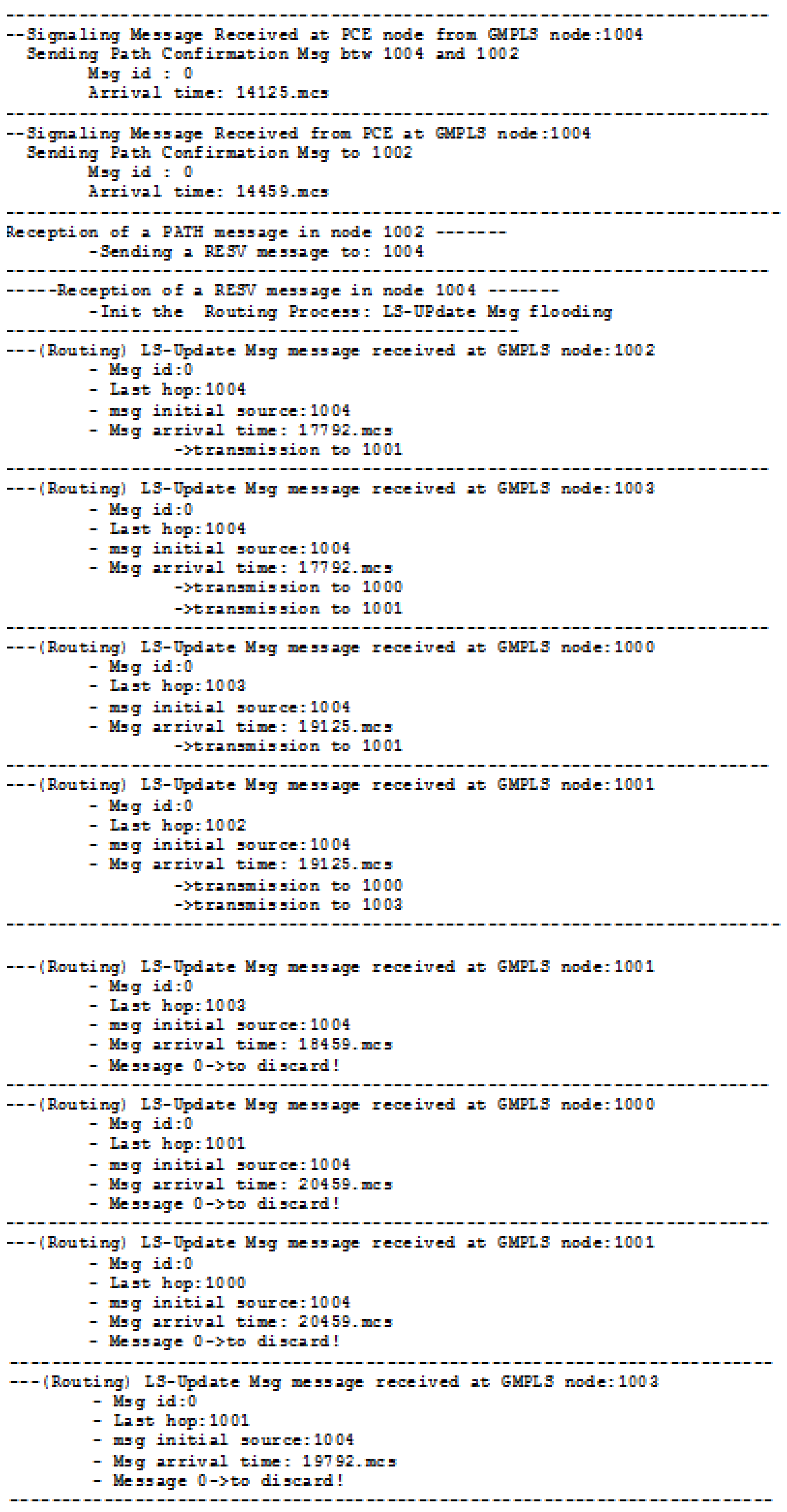
Network mask	This field has the address mask for the network. In this case study it will be considered to be 0 because we don´t have an IP based network.
Attached Router	This field has the ID of the routers that are attached to the network, keep in mind that the Designated Router also has the ID in this list.

B – JA(G)OBS output files



```
-----Reception of a RESV message in node 1003 -------------------------
        -Init the Routing Process: LS-UPdate Msg flooding
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1001
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1000
                ->transmission to 1002
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1004
        - Msg id:0
        - Last hop:1003
        - msg initial source:1003
        - Msg arrival time: 35321.mcs
        - Flood the message:
                ->transmission to 1002
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1001
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1000
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1004
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Flood the message:
                ->transmission to 1001
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1001
        - msg initial source:1003
        - Msg arrival time: 35654.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1002
        - msg initial source:1003
        - Msg arrival time: 35987.mcs
        - Message 0->to discard!
----------------------------------------------------------------------
```

Figure 34 – LSU flooding in JA(G)OBS unmodified



```
--------------------------------------------------------------------------
--Signaling Message Received at PCE node from GMPLS node:1004
  Sending Path Confirmation Msg btw 1004 and 1002
        Msg id : 0
        Arrival time: 14125.mcs
--------------------------------------------------------------------------
--Signaling Message Received from PCE at GMPLS node:1004
  Sending Path Confirmation Msg to 1002
        Msg id : 0
        Arrival time: 14459.mcs
---------------------------------------------------------------------------
Reception of a PATH message in node 1002 -------
        -Sending a RESV message to: 1004
--------------------------------------------------------------------------
-----Reception of a RESV message in node 1004 -------
        -Init the  Routing Process: LS-UPdate Msg flooding
-------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1002
        - Msg id:0
        - Last hop:1004
        - msg initial source:1004
        - Msg arrival time: 17792.mcs
                ->transmission to 1001
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1003
        - Msg id:0
        - Last hop:1004
        - msg initial source:1004
        - Msg arrival time: 17792.mcs
                ->transmission to 1000
                ->transmission to 1001
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1003
        - msg initial source:1004
        - Msg arrival time: 19125.mcs
                ->transmission to 1001
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1002
        - msg initial source:1004
        - Msg arrival time: 19125.mcs
                ->transmission to 1000
                ->transmission to 1003
---------------------------------------------------------------------------

---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1003
        - msg initial source:1004
        - Msg arrival time: 18459.mcs
        - Message 0->to discard!
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1000
        - Msg id:0
        - Last hop:1001
        - msg initial source:1004
        - Msg arrival time: 20459.mcs
        - Message 0->to discard!
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1001
        - Msg id:0
        - Last hop:1000
        - msg initial source:1004
        - Msg arrival time: 20459.mcs
        - Message 0->to discard!
--------------------------------------------------------------------------
---(Routing) LS-Update Msg message received at GMPLS node:1003
        - Msg id:0
        - Last hop:1001
        - msg initial source:1004
        - Msg arrival time: 19792.mcs
        - Message 0->to discard!
--------------------------------------------------------------------------
```

Figure 35 – JA(G)OBS output when PCE is active unmodified

C – Simulation Results

CPU results

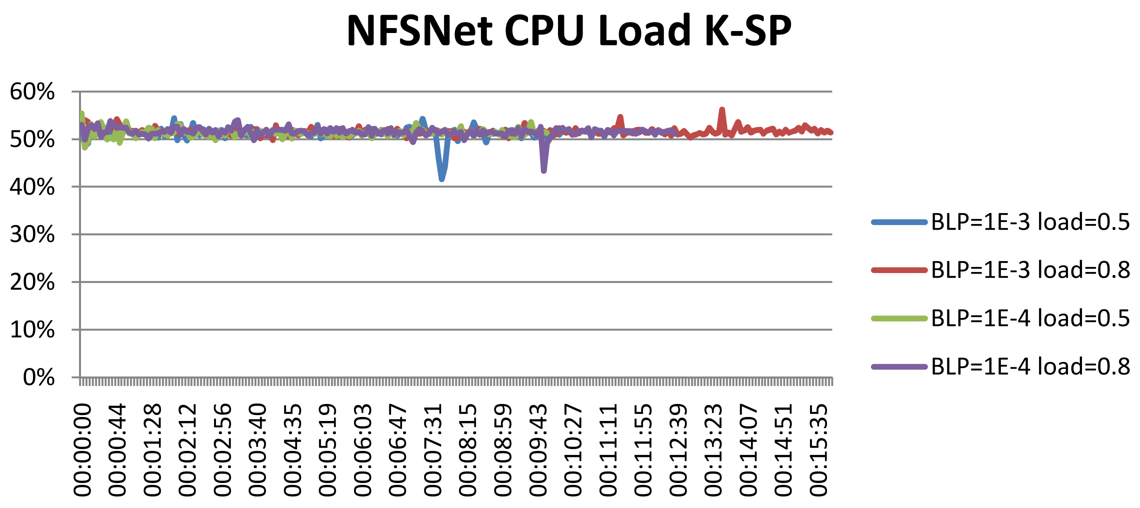


Figure 36 – CPU load in NFSNet with K-SP

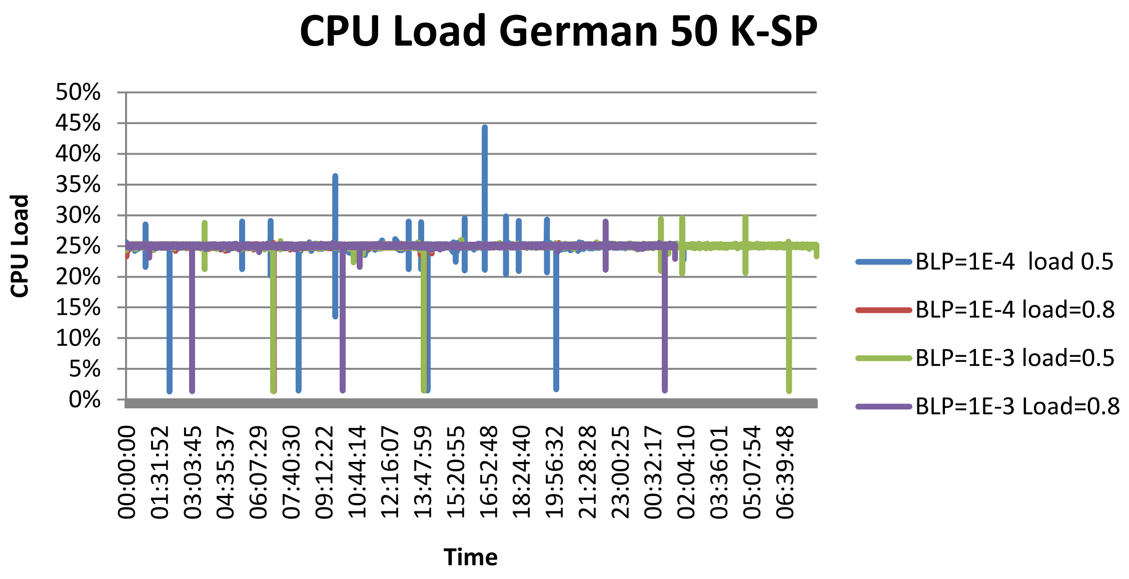


Figure 37 – CPU load in German50 with K-SP

It is noticeable that the behavior of the CPU doesn't change when we change from SP to K-SP algorithm.

Memory Results

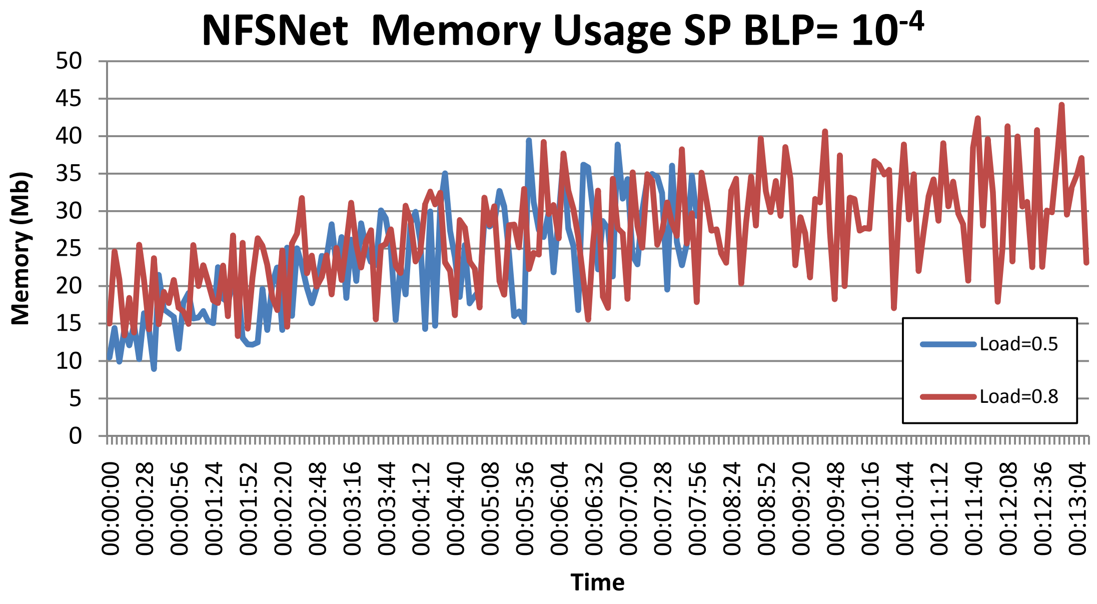


Figure 38 – Memory Consumption in NFSNet with BLP= 10^{-4} SP algorithm

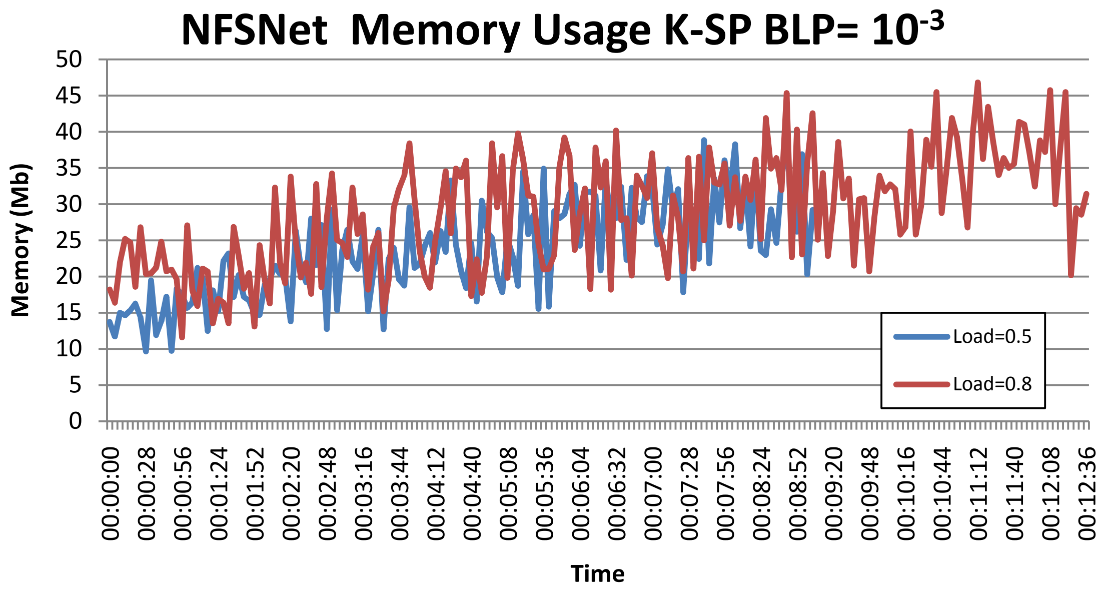


Figure 39 – Memory Consumption in NFSNet with BLP= 10^{-3} K-SP algorithm

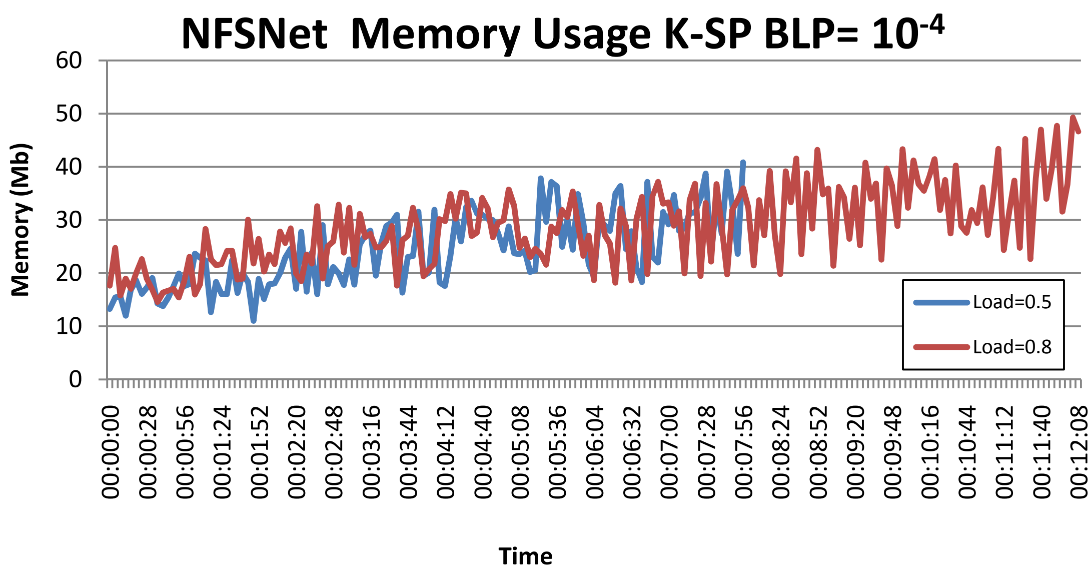


Figure 40 – Memory Consumption in NFSNet with BLP= 10^{-4} K-SP algorithm

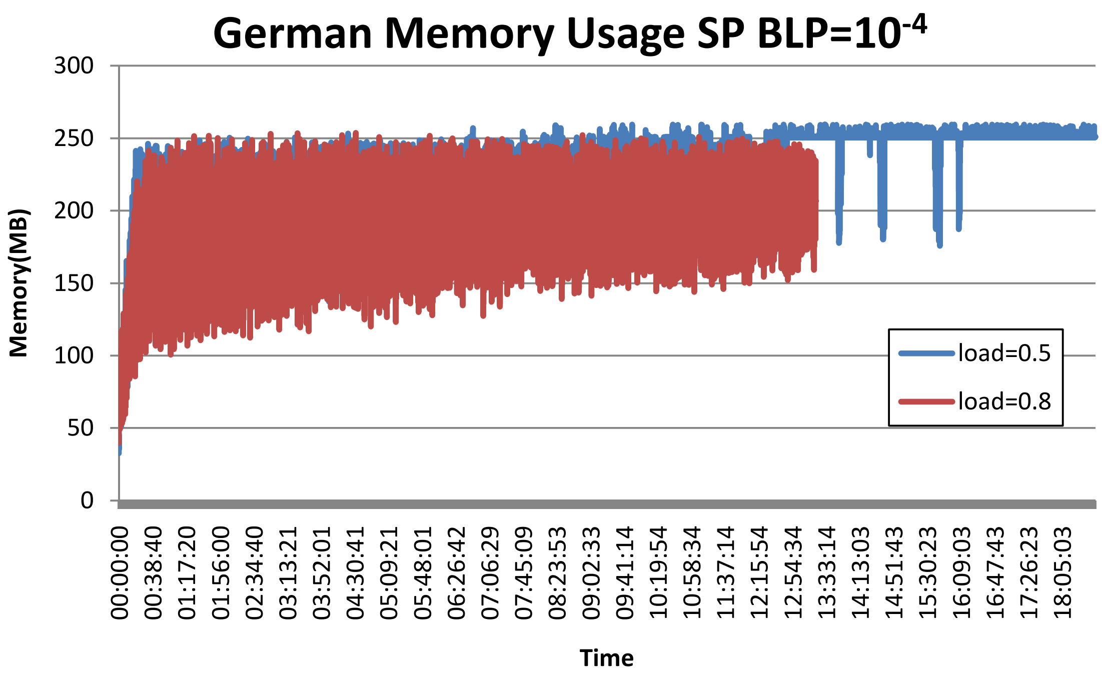


Figure 41 – Memory Consumption in NFSNet with BLP= 10^{-4} SP algorithm

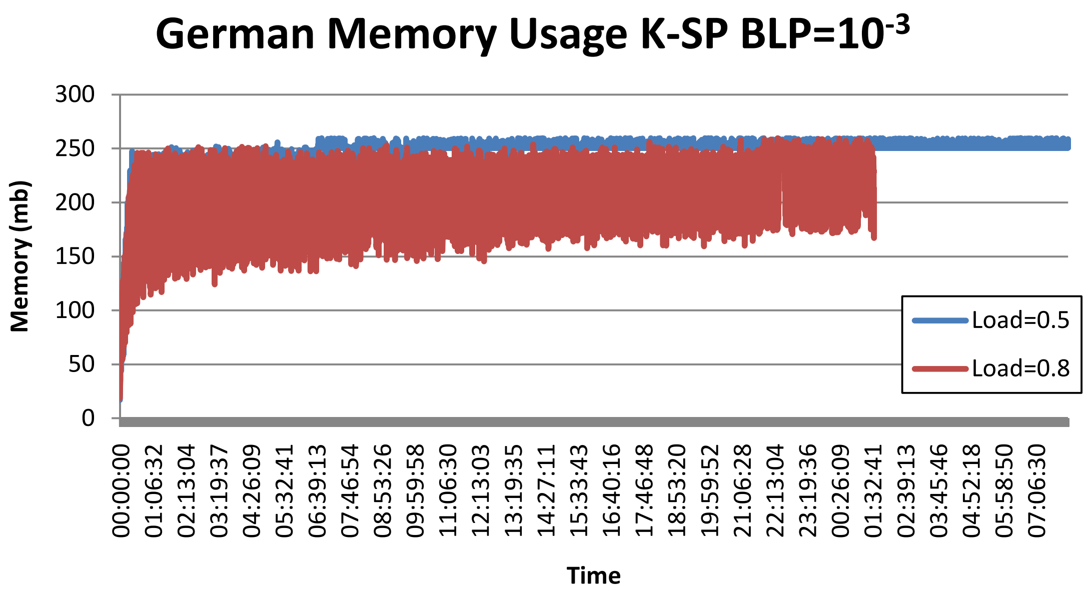


Figure 42 – Memory Consumption in German50 with BLP= 10^{-3} K-SP algorithm

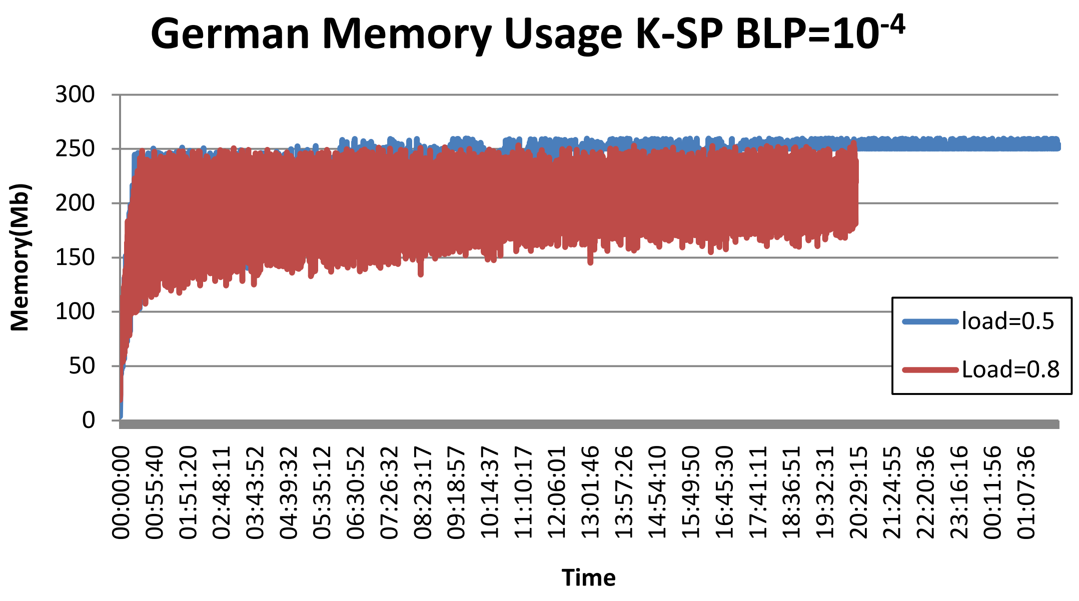


Figure 43 – Memory Consumption in German50 with BLP= 10^{-4} K-SP algorithm

It is noticeable that the memory behavior doesn’t change when we change from SP to K-SP algorithm.

List of Acronyms

BCP	Burst Control Packet
BE	Best Effort
BER	Bit Error Rate
BLP	Burst Lost Percentage
CP	Control Plane
CPU	Central Processor Unit
CR-LDP	Constraint-based Routing Label Distribution Protocol
DiffServ	Differentiated service
DVR	Distance Vector Router
FDL	Fiber Delay Line
GLASS	GMPLS Lightwave Agile Switching Simulator
GMPLS	Generalized Multiprotocol Label Switching
GUI	Graphical User Interface
HP	High Priority
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
IS-IS-TE	Intermediate System to Intermediate System with Traffic Engineering
JET	Just Enough Time
JIT	Just In Time
LMP	Link Management Protocol

LSA – Link State Advertisement

LSP – Label Switched Path

LSR_1 – Link State Routing

LSR_2 – Link State Request

LSU- Link State Update

MILP – Mixed Integer Linear Programming

MPLS – Multiprotocol Label Switching

NGI - Next Generation Internet

O/E/O – Optical to Electric to Optical

OBS – Optical Burst Switching

OCS – Optical Circuit Switching

OPS – Optical Packet Switching

OSPF – Open Shortest Path First

OSPF-TE – Open Shortest Path First with Traffic Engineering

PCE – Path Computation Element

PHB – Per Hop Behavior

QoS – Quality of Service

RAM – Random Access Memory

RIP – Routing Information Protocol

RSVP-TE – Resource Reservation Protocol with Traffic Engineering

RTT – Round Trip Time

SDH – Synchronous Digital Hierarchy

SSF - Scalable Simulation Framework

TCP	– Transmission Control Protocol
TDM	– Time Division Multiplexing
TE	– Traffic Engineering
TLV	- Time-Length-Value
UPC	- Universitat Politècnica de Catalunya
UPC-CBA	- Broadband Communication research group
VoIP	– Voice over Internet Protocol
WDM	– Wavelength Division Multiplexing
XML	- eXtensible Markup Language