NTNU – Trondheim
Norwegian University of
Science and Technology

Modifications to and performance evaluation of output scheduling in 3LIHON nodes

Andrea Marchini

Master of Telematics - Communication Networks and Networked Services (2
Submission date: December 2013
Supervisor: Norvald Stol, ITEM
Co-supervisor: Walter Cerroni, University of Bologna

Norwegian University of Science and Technology
Department of Telematics

Problem description

In the latest years, communications have become more and more demanding, due to increasing needs of transferred data volumes and speed requirements. Recent research show that future core networks will have to handle both circuit and packet switched traffic. An introduction on recent network and traffic topology evolution and future requirements is therefore advisable.

This work will focus on integrated hybrid optical networks. We start by explaining the three main categories to classify hybrid optical networks: client-server, parallel and integrated. More specifically, the most important architectures proposed so far for the latter are ORION, OpMiGua and 3-LIHON; they should be presented as well.

3-LIHON and its output scheduling algorithm have been previously studied by Gaia Leli in her master thesis (2012), so that work can be extended by investigating and trying to improve the performance of SM/BE traffic under heavy system load. In this condition short SM/RT packets tend to frequently interrupt SM/BE packets, thus increasing the loss of SM/BE packets. Some possible ways to remedy this could be for example:

- retransmitting interrupted packets;

- resuming interrupted packets. More in details, a proper (Fragment End) optical code can be used, in addition to the tail and header ones. These packets will not be influenced by preemptive GST and SM/RT ones (as they both have priority and will be properly switched at the following node, according to their type and optical code, if present), so their transmission can be accomplished, even after an interruption, on the same wavelength of the optical fiber. This can be performed by adding a register for each wavelength at both transmitter and receiver. At the following switch, fragmented packets will be reassembled.

- putting FDLs at the output of the OPS that manages SM/RT traffic could be another possible way to reduce the effects of preemptive packets over SM/BE ones, in order to decrease the probability of wavelength contention between SM/RT and SM/BE packets. These FDLs however should not be long enough to compromise the real time requirements of SM/RT traffic.

Moreover, if results given by the above techniques are not as effective as expected, an intermediate architecture between OpMiGua and 3-LIHON can be studied. A possible architecture could be obtained by removing the OPS managing SM/RT traffic. This would lead to some other changes, like:

- exploiting the EPS to handle both SM/RT and SM/BE packets;
- having the FDL at the OXC output equal to the maximum length of a SM/RT packet
- employing only one output queue at the EPS, which involves reordering incoming packets according to their length. Since most SM/RT packets are shorter than SM/BE ones, usually they will be prioritized.

Evaluation of this method also needs to take into account drawbacks and consequences from packet reordering.

As the considered systems are very complex, an analytic approach is challenging. This study will therefore focus on simulations to evaluate performances. More specifically, the software selected to assess them is DEMOS, a system for discrete event modelling on Simula.

Abstract

Optical networks became very important in the latest years, thanks to their high traffic capacity, because of the continuous growth of total exchanged data and the request of real-time communication.
Since they have (relatively) long adaptation times, this technology is not inherently the most suitable to carry bursty traffic like the current one. Hybrid networks try to get the best from optical circuit switching and optical packet switching.
This work is focused on an hybrid network architecture called **3LI-HON (3-Level Integrated Hybrid Optical Network)**. It has three different quality of service (QoS) levels, in order to meet different requirements:

- Guaranteed Service Type (**GST**): resembling an optical circuit switched service, it does not allow data loss.
- Statistically Multiplexed Real Time (**SM/RT**): resembling an optical packet switched service, it ensures none or very small delay inside the network, it allows a low data loss ratio and bandwidth contention.
- Statistically Multiplexed Best Effort (**SM/BE**): resembling an optical packet switched service with very small overall packet loss but no guaranteed delay inside the nodes.

In a 3LIHON node, SM/BE traffic interrupted by packets with higher priority is dropped. This means that time and resources spent sending the SM/BE packet until the interruption are wasted.

In this work we try to avoid this behaviour, by implementing and comparing three new output scheduling algorithms. They are different versions of the standard 3LIHON node:

- **3LIHON-RS**: SM/BE interrupted packets are resumed once higher priority traffic has been sent.
- **3LIHON-RT**: SM/BE interrupted packets are retransmitted.
- **3LIHON-2R**: since GST packets are usually much longer than SM/RT ones, resuming an interruption caused by GST traffic leads to a long wait. This architecture resumes SM/BE packets interrupted by SM/RT traffic and retransmits them if they have been interrupted by GST traffic.

A proper (Fragment End) Optical Code can be used to implement and signal packets interruption and resuming, Every 3LIHON version has been tested with increasing traffic loads, but never so high to overload the node. Three different simulators have been written in order to study the behaviour of the different architectures, by using the **Simula** programming language and its *context class* **DEMOS**, specifically intended for discrete event modelling. The node input part has just been modelled, since it does not directly influences results.

As a function of system load, performances of these new architectures have been analysed, paying attention especially to SM/BE packets mean delay, SM/BE queues, delay distribution of SM/BE traffic, wavelengths utilization, GST and SM/RT interruptions over SM/BE traffic, standard deviation of SM/BE delay and SM/RT packet loss.
It has been observed that 3LIHON-2R does not have the best absolute performances, but is the best trade off among all the systems and is very well balanced when system load increases. It is the architecture showing the lowest mean SM/BE packet delay. 3LIHON-RS proved to have the lowest SM/BE queue values and less interruptions of SM/BE traffic. Furthermore, it has the best wavelengths utilization performances. 3LIHON-RT achieved the worst results and is the first architecture overloading.

Keywords

Scheduling
Integrated hybrid
Performance evaluation
Optical networks

To my family, myself and the future.

Contents

List of abbreviations

3LIHON	3-Level Integrated Hybrid Optical Network
GST	Guaranteed Service Type
SM/RT	Statistically Multiplexed Real Time
SM/BE	Statistically Multiplexed Best Effort
FDL	Fiber Delay Line
OXC	Optical Cross Connect
OPS	Optical Packet Switch
EPS	Electronic Packet Switch
FIFO	First In First Out
WRON	Wavelength Routed Optical Network
QoS	Quality of Service
DPT	Detect Packet Type
OC	Optical Code
DWDM	Dense Wavelength Division Multiplexing
ROADM	Reconfigurable Optical Add-Drop Multiplexer

Introduction

Motivation and current work

In the last decades, communication technology has changed a lot. At the beginning, telephony was the main service, but then it has been superseded by data, which are now the biggest portion of traffic, also as a result of the advent of the Internet.
Since the beginning, the Internet has provided best effort services and this has been sufficient until recent times, but lately there has been the need of traffic differentiation, i.e. QoS management, due to new applications and the growth of users and transferred data volume.

The evolution in fiber transmission technology, thanks to e.g. Dense Wavelength Division Multiplexing (DWDM), reconfigurable optical add-drop multiplexers (ROADMs) and all-optical crossconnects (OXCs), made available a huge transmission capacity, so that now optical devices and systems carry the largest amount of data, while electronics introduce intelligence to the data and control plane. The problem is that optical networks are not suitable for bursty traffic, thus resulting in a transport inefficiency, mainly because of their (relatively) slow adaptation times. This leads to the concept of hybrid networks, which try to obtain the merits of different technologies - optical circuit-switching and optical packet-switching - by combining them into one architecture, while avoiding their disadvantages.

The hybrid network architecture we will focus on in this work is called 3-Level Integrated Hybrid Optical Network (3LIHON) [6]. The aim of this work is to get a fully functional simulation model of a basic

3LIHON node, implement resume and retransmission for interrupted packets and study the performances for outgoing traffic towards the output wavelengths.

Problem definition and goal

Problem definition

In a standard 3LIHON node, SM/BE traffic suffers from losses and delay. We study the behaviour of three new 3LIHON architectures at different traffic loads. The scheduling algorithms introduced avoid losses and manage the delay in different ways.
We want to observe SM/BE packets delay and its characteristics, such as mean value, distribution and standard deviation, besides wavelength utilization, SM/BE queue length and interruptions.

Goal

The goal of this work is to study and compare the performances of a 3LIHON node with different algorithm for output scheduling, especially for what concerns SM/BE packets. The main figures observed are interruptions and delay for SM/BE packets, efficiency of the different scheduling algorithms and wavelength utilization at different loads. Packet loss probability for SM/RT traffic is of minor interest.

Outline

The outline of the present work is the following:

- Chapter 1 explains the concept and the architecture of a 3LIHON node, as well as the three modified architectures.
- Sections from 2.1 to 2.4 illustrate how the node has been modelled and how the different sources and types of packets have been implemented in the simulator.
- Section 2.6 describes how the new introduced architectures have been implemented.

- Section 3.1 and all its subsections present results and comparisons between the three different architectures.
- Finally, Chapter 4 illustrates conclusions for this work and possible future works.

Chapter 1

Purpose and architecture of a 3LIHON node

1.1 Circuit switching and packet switching

Like in the ”old” telephony, circuit-switching allocates resources for communication between two parties in a reserved way, so that they have the complete availability of the connection for the entire duration of the conversation. This requires a previous signalling phase, in order to set-up the appropriate circuit, performed with the exchange of messages; thus, it is necessary at least the round trip time of the connection and, in large networks, this can take a relative long time, making the circuit-switched technology not suitable for the transport of bursty or highly variable traffic.
With the advent of the Internet and the consequently raised demand, there had been the need of upgrading the copper networks and replacing them with optical fibers, to provide the necessary bandwidth, that increased further on thanks to optical amplifiers and the deployment of Dense Wavelength Division Multiplexing. At first, these links were used as point-to-point interconnections, but now we are able to provide real optical networking, employing optical switches. This evolution has been possible by using so-called lightpaths (i.e. a wavelength passes through a node transparently, thus relieving the router

of the additional work of inspecting the transit traffic) and Optical Cross Connects (OXC, which switch the optical signal). Another step has been made with the advent of Automatically Switched Optical Networks (ASON), because the lightpath can be set-up by a control plane, saving the operator from doing it [2]. Besides the relatively long set-up time, circuit-switched networks have another drawback: there will probably be an inefficient usage of the bandwidth, since a single wavelength can carry up to a few ten Gbit/s.

In packet-switching, data are sent in packets and this can also be applied to the optical world, thus talking of Optical Packet Switching (OPS). Packets are formed by a header and a payload: the former contains the routing information and is processed electronically, the latter is kept in the optical domain. This allows a more efficient usage of the network resources, because a wavelength is occupied only when necessary, suiting better to a highly variable traffic like the present one, and has had a boost with the IP protocol. Furthermore, it is possible to share the wavelengths on the links thus having a statistical multiplexing gain. Packet-switching has some disadvantages, too, as routers and switches along the path have to process the transit traffic, so congestion may happen and consequent delays can occur, which are difficult to predict. Also, possible contentions at intermediate nodes need to be solved by queuing packets until enough resources are available [5]. In the optical domain, the only way, at the moment, to queue packets is by using a Fiber Delay Line (FDL), which introduces a fixed delay.

In an integrated hybrid optical network, all network technologies share the same bandwidth resources in the same network simultaneously. Thus, in a network with OPS and wavelength switching, the node can choose whether to send the traffic in a wavelength-switched mode or in a packet-switched mode. Its choice can be determined by a congestion situation or by QoS requirements (the wavelength path are used for high-priority traffic). Usually, dynamic traffic is sent in packet-switched mode, whereas wavelength-paths are preferred for more smooth, stable traffic. This means also that each node must have a wavelength-switched device and a packet-switched device. Integrated hybrid optical networks are the most bandwidth-efficient, but also the most complex.

1.2 QoS differentiation

The 3LIHON architecture proposes three service levels able to support different QoS needs, each one associated to a different switching technology. They are respectively:

- Guaranteed Service Type (GST): similar to a circuit-switched service, it does not allow information loss inside the network;
- Statistically Multiplexed Real Time (SM/RT): similar to a packet switched service, it guarantees none or very limited delay inside the network, but tolerates some packet loss inside the nodes and bandwidth contention;
- Statistically Multiplexed Best Effort (SM/BE): similar to a packet switched service, it does not guarantee delay inside the nodes and allows very small packet loss, e.g. by employing link-level packet retransmission.

1.3 Architecture of a 3LIHON switching node

Figure 1.1 presents a general 3LIHON node, with N input/output fibers carrying M wavelengths. Each of them has a Detect Packet Type (DPT) block, whose function is to identify the transport service of the incoming packets, in order to treat them accordingly. If a GST packet is detected, it is switched to an OXC and then forwarded to its pre-established wavelength on the output fiber. SM/RT packets are switched to an OPS and SM/BE to either an Electronic Packet Switch (EPS) or an OPS with electronic buffering. GST and SM/BE are expected to be the great part of the overall traffic and luckily OXCs and EPSs are today commercially available components, unlike

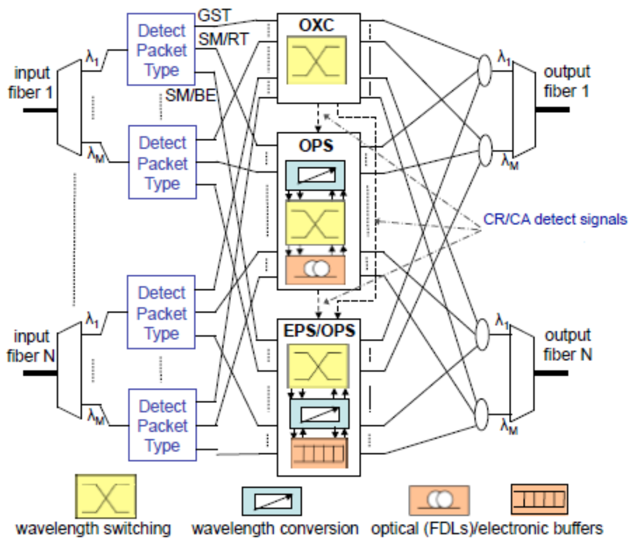

Figure 1.1: General scheme of a 3LIHON switching node [6]

the ones needed to handle SM/RT traffic. Switches managing SM/RT and SM/BE packets also perform Collision Avoidance (CA) among different types and contention resolution (CR) among packets of the same class.

In Figure 1.2 is shown the architecture of a DPT, implemented using OC detection, as OC encoders/decoders are passive components, so they do not increase the overall power consumption. This DPT sends all the unmarked traffic to the OXC, this means that GST traffic does not require OC marking, whereas SM/RT and SM/BE traffic need to be distinguished and are then marked with an OC header and tail. The header is used to send the packet to the proper switching subsystem by accordingly opening or closing the gates, the tail to set the DPT back to its default configurations, after the end of the transmitted packet. Thus OCs may be divided into two groups, i.e. employed for SM/RT and SM/BE traffic.

Figure 1.3 shows how CA is managed using Detect signals. A generic wavelength channel j on output fiber k, indicated with (j, k) is consid-

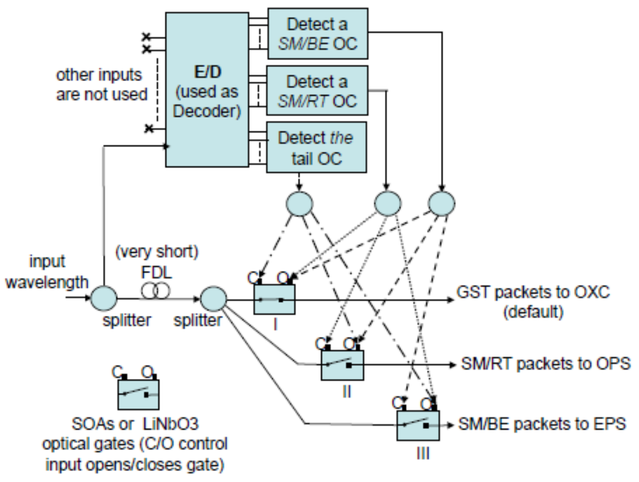

Figure 1.2: Detect Packet Type architecture implemented with OC detection [6]

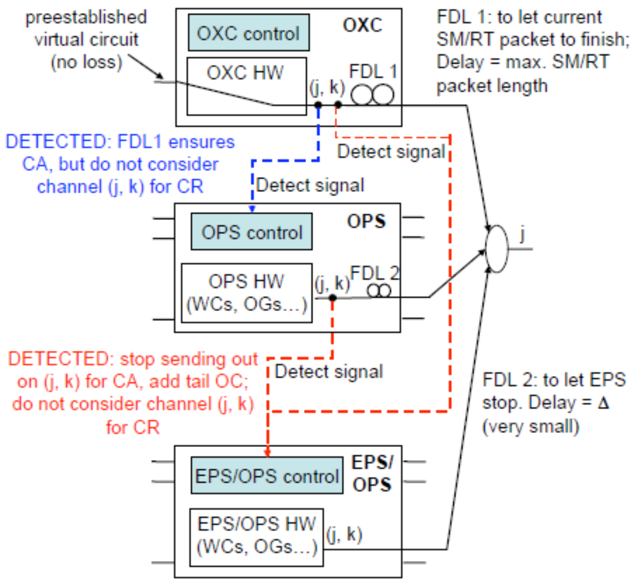
preestablished virtual circuit (no loss)
OXC control
OXC
OXC HW
(j, k)
FDL 1
FDL 1: to let current SM/RT packet to finish; Delay = max. SM/RT packet length
DETECTED: FDL1 ensures CA, but do not consider channel (j, k) for CR
Detect signal
Detect signal
OPS control
OPS
OPS HW (WCs, OGs…)
(j, k)
FDL 2
j
DETECTED: stop sending out on (j, k) for CA, add tail OC; do not consider channel (j, k) for CR
Detect signal
FDL 2: to let EPS stop. Delay = Δ (very small)
EPS/OPS control
EPS/ OPS
EPS/OPS HW (WCs, OGs…)
(j, k)

Figure 1.3: How collision avoidance is performed [6]

ered. GST packets need to be forwarded with maximum priority and without loss in circuits pre-established during the network planning phase or set up by the ASON, so, as soon as one of them arrives, a couple of Detect signals is sent by the OXC to the OPS and EPS control. GST packets are non-preemptive over SM/RT ones, i.e. there is no collision among their transport classes, this is why they enter a fixed length FDL1. Its length is chosen equal to the maximum length of a SM/RT packet, so that it can always be completely sent out on channel (j, k) before the beginning of the GST packet transmission. After having received a Detect signal, the OPS stops considering channel (j, k) for contention resolution for incoming packets and must use other wavelengths; furthermore, after having sent the GST packet, the OPS has to wait for a time equal to the delay introduced by FDL1 before starting to send SM/RT packets again, in order to let the GST packet finish its transmission through the FDL1 buffer. SM/BE packets are preempted by both GST and SM/RT ones to avoid further delays, so as soon as the EPS/OPS receives a Detect signal, it must stop transmission immediately, add a tail OC and not consider channel (j, k) for contention resolution for incoming packets. SM/BE packets that may have been interrupted can be dropped, resumed or later retransmitted by the EPS on a free wavelength. FDL2 is used so that SM/RT packets allow the EPS to stop its transmission and has a very small delay. Note that SM/RT packets should have a significantly shorter length than GST ones, so that the delay introduced by FDL1 does not affect them too much when passing through the OXC.

1.4 Modifications to the standard 3LIHON output scheduling

Three main changes have been made to the standard output scheduling of a 3LIHON node:

- **resuming (RS)**: transmission of SM/BE packets interrupted by higher priority traffic is then resumed, using the same wavelength.
  The concept which led to this modification was to avoid dropping interrupted SM/BE packets and therefore waste transmis-

sion capacity. Time and capacity spent until the interruption would just be wasted in the standard 3LIHON node, because the SM/BE packet is dropped. This would also cause a data loss.
In the 3LIHON-RS node, when a SM/BE packet is interrupted by higher-priority traffic it is not dropped, but simply suspended and then resumed again when the previously used wavelength gets free.

- **retransmission (RT)**: SM/BE packets interrupted by higher priority traffic are then put back in queue for later retransmission, using any free output wavelength.
  This alternative does not avoid wasting transmission capacity and time, because time and capacity spent until the interruption can not be got back. An advantage of this choice, compared to the 3LIHON-RS, is that an interrupted packet does not have to wait for a specific output wavelength to be free, but it can be sent through any available resource.

- **resuming/retransmission (2R)**: SM/BE packets interrupted by GST traffic are put back in queue for later retransmission, using a free output wavelength. SM/BE packets interrupted by SM/RT traffic are resumed, using the same wavelength.
  This is a compromise solution between the two previous versions. This scheduling algorithm is based upon the idea that GST packets are usually longer, but less frequent, than SM/RT packets, so it may be quicker to retransmit an interrupted SM/BE packet through another free wavelength, instead of resuming it and waiting for the complete transmission of the interrupting GST packet.
  Furthermore, it has been previously showed ([3]) that, at high loads, interruptions to SM/BE packets are caused mainly by SM/RT traffic. We thought that avoiding to retransmit all these SM/BE packets could improve overall performances.

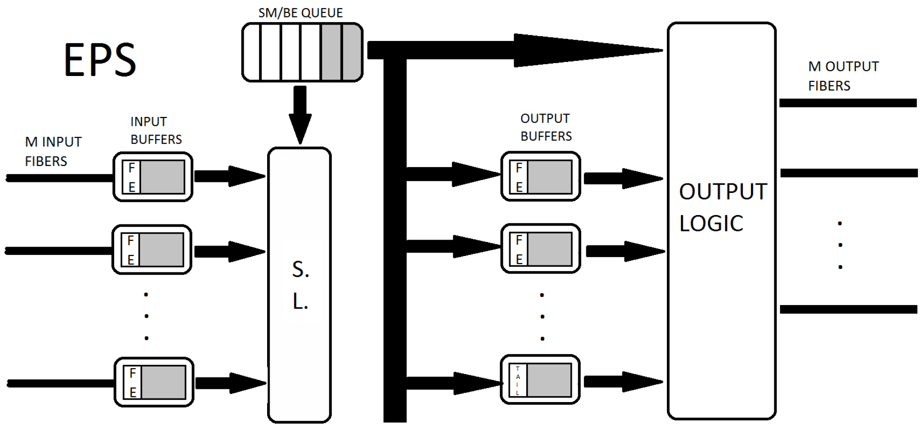

Figure 1.4: Modifications introduced to the standard 3LIHON node

1.4.1 Architecture of the modified 3LIHON node

Changes made to the standard output scheduling of the 3LIHON node also affect the node's architecture.
Since retransmission is quite simple to implement, because it only needs to put the packet at the back of the queue, we will now focus on resuming.

Introduced modifications can be seen in Figure 1.4. They only regard the EPS.
Since we are dealing with a cut-through switching typology, it is not possible to wait for an interrupted packet to be completely reassembled at the following node. Inside the EPS, an input and an output buffer have been introduced. These buffers hold a piece of a possible incoming or outgoing interrupted SM/BE packet. There is a couple of buffers for each wavelength.
It is also necessary to implement a new optical code, we call it Fragment-End Optical Code (FE-OC). This OC is put in the tail and is used to mark the end of each SM/BE fragment.
The input buffers receive incoming fragments coming from the previous node from the input fibers. Every buffer must be able to distinguish between tail OC and FE-OC, because the corresponding tail fragment can be received while another fragment is in the buffer.

The scheduling logic (S.L. in the picture) is in charge of collecting these incoming fragments and put them inside the node's logic, for possible reassembling. This aspect has not been covered, since this work focuses on the output scheduling (so we may see the studied node as an input node). Since it may happen that the SM/BE queue is empty, the scheduling logic also has to decide whether to take packets from the input buffers or the queue. In case the reassembly is not possible, the fragment has to be forwarded as it is.
While sending a SM/BE packet, an interruption may occur, thus the remaining fragment must be put inside the output buffer corresponding to the output wavelength being used. Once again, it is the scheduling logic that takes this decision.
During normal operations, it will probably happen to have some output buffers busy and some packets inside the queue, waiting to be scheduled. It is now the output logic that is in charge of deciding which packet to take and send through the output wavelength.
The 3LIHON-RS scheduling is probably more difficult to implement using a cut-through architecture rather than a store-and-forward architecture, since there is no chance to store fragments in buffers and reassemble them, but every fragment or packet must be immediately forwarded. This will also lead to more complicated logics.

Chapter 2

Node modelling, code and scheduling algorithms

2.1 Node modelling

The node architecture is showed in Figure 1.1, it consists of N input/output fibers, carrying M wavelengths.
The incoming traffic includes GST, SM/RT and SM/BE packets in various percentages.
GST packets have reserved wavelengths available and do not experience any loss; their arrival process is described by Poisson distribution, whereas their inter-arrival process is described by a negative exponential distribution. They are then collected in bursts whose length is constant and forwarded through the network by the OXC.
Unlike GST packets, SM traffic may experience loss since it has no wavelength reserved and is fitted in gaps between GST packets. Its inter-arrival process and the SM packets length are both described by a negative exponential distribution, whereas the arrival process follows Poisson distribution.
The simulation model presented in this work is asynchronous and unslotted.
We assume no FDL available in the OPS for SM/RT traffic and one buffer with FIFO priority order for SM/BE traffic.

2.2 Code description

The programming language chosen to implement the simulator of the 3LIHON node is Simula, with the help of DEMOS, a context class for discrete event simulation.
Simula, developed in the 1960s in Norway for simulation purposes, has been the first object-oriented programming language; DEMOS stands for *Discrete Event Modelling on Simula* and is a context class created by Graham Birtwistle [1]. A context is a package written in Simula which extends the language towards a specific problem area. DEMOS was meant to help beginners in discrete event simulation by augmenting Simula with a few building blocks, which provide a standardised approach to a wide range of problems.

2.3 Sources characterization

Three different kind of sources have been created, in order to simulate the arrival of three different kind of packets: GST, SM/RT and SM/BE.

Since GST packets have pre-defined paths along the network, 32 GST sources are created, one for each wavelength (M=32). In this way fixed links are realized and we assure the delivery of each GST burst. Each source makes the inter-arrival time follow a negative exponential distribution, whose mean value is function of the length of the GST packet and the bit channel rate. We assume a bit channel rate of 1Gbit/s for each wavelength. We can then obtain a specified load of GST traffic on each wavelength.

The SM/RT traffic is generated by a single source and the inter-arrival time follows a negative exponential distribution. The mean value is function of the mean length of a SM/RT packet, of the bit channel rate and of the relative load of SM/RT traffic.

A single source generates SM/BE packets, whose inter-arrival time follows a negative exponential distribution, with mean value given by the mean length of a SM/BE packet, the bit channel rate and the relative load of SM/BE traffic.

2.4 Packets characterization

Each type of traffic represents different services with different qualities in the future network, so it is reasonable to assign them different packet lengths [6].

GST bursts have constant distribution with a fixed mean value of 40000 bytes.
SM/RT packets length follows a negative exponential distribution with a mean value of 40 bytes.
The mean length for SM/BE packets is related to the one above and is modelled by a negative exponential distribution as well. In this work we consider a mean value equal to 40 times the mean length of a SM/RT packets, that is 1600 bytes.
According to the 3LIHON node structure, the FDL length after the OXC should be equal to the maximum SM/RT packet length, so both are set to 5 times the SM/RT mean packet length. This also means that any possible SM/RT packet longer than that will be cut off to this maximum value.

2.4.1 Guaranteed Service Type (GST) traffic description

GST traffic has been implemented in DEMOS as shown in Figure 2.1. Two new entities has been introduced:

- **GST generator**

  This entity generates GST packets in a loop. There is one generator for each wavelength and, since GST packets have the highest priority, they can not be interrupted by packets of other classes. Once a GST packet has been generated, it is immediately scheduled and the corresponding wavelength is busy until it has been completely transmitted.

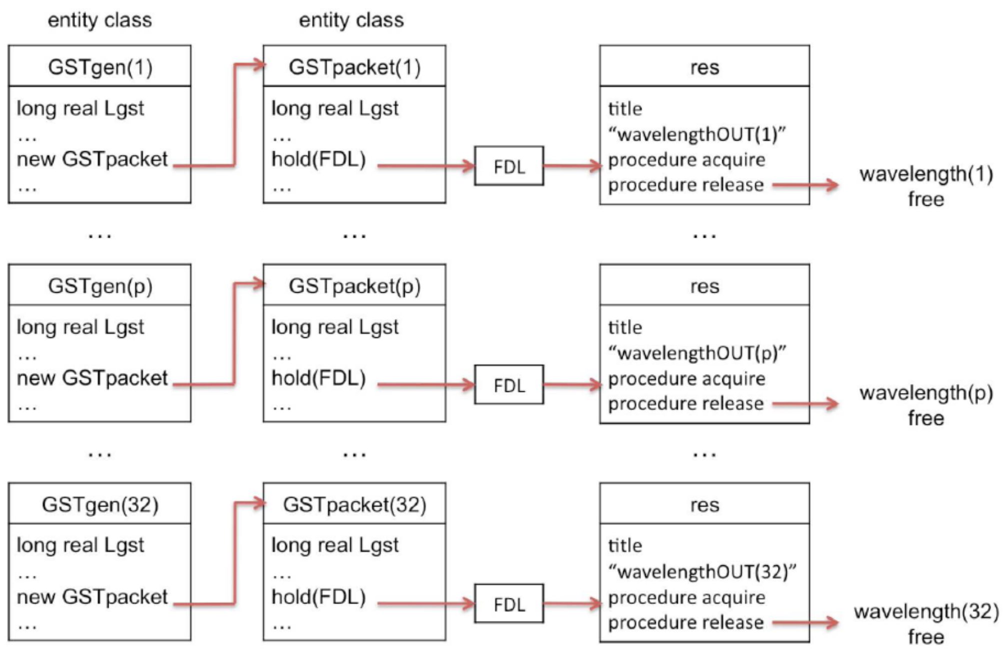

Figure 2.1: GST traffic implementation [3]

- **GST packet**

  Each GST source is directly linked with its own output wavelength. As soon as a GST packet is generated and then scheduled, the corresponding FDL is marked as "busy" using a boolean variable and the corresponding flag gets value 1.
  There is a flag for each wavelength and they are globally visible. When they have value 1 it means that the current wavelength is being used by a GST packet and thus it can not be used by other types of traffic.
  The FDL has a length equal to the maximum SM/RT packet length, that is, in the most cases, shorter than the GST packet. When the packet reaches the output wavelength, the resource is acquired and held until necessary.
  Once the FDL has become available again, the boolean variable becomes false, the flag is set to 0 and an appropriate message is

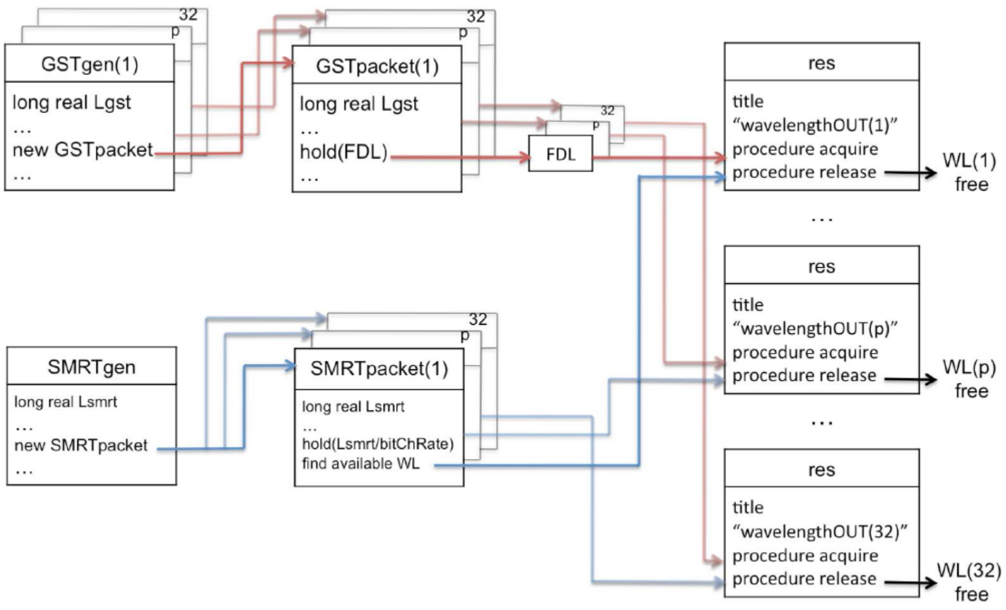

Figure 2.2: SM/RT traffic implementation [3]

sent to the BE server, signalling that the current wavelength is now free.

2.4.2 Statistically Multiplexed Real Time (SM/RT) traffic description

Figure 2.2 shows how SM/RT traffic has been implemented. Two new entities has been introduced:

- **RT generator**

  SM/RT traffic is generated by only one source, that creates packets according to the specifics in chapter 2.3.
  In order to guarantee collision avoidance, any SM/RT packet having length greater than the maximum specified must be cut off to the maximum length value.

- **RT packet**

SM/RT packets length distribution follows what stated in chapter 2.4.
In order to send a SM/RT packet, either a free wavelength can be used or a resource transmitting SM/BE packets, since they can be interrupted by SM/RT traffic.
First we start looking for a free output wavelength, beginning from the first one. This will lead to a higher utilization of the first wavelengths than the others. If a flag with value 0 is detected, this means that wavelength is available.
If all of them are busy, we start looking for a resource which is transmitting SM/BE traffic, since it has lower priority than SM/RT packets and thus it can be interrupted. If a flag with value 3 is found, it means SM/BE traffic is being carried by that wavelength and it is allowed to interrupt it and use that resource to send SM/RT packets.
Every time a SM/BE packet is interrupted this event is logged, an interrupt signal with value 2 is sent to the SM/BE packet and the wavelength carrying traffic is registered as available for by SM/RT packets.
Once a wavelength has been found, the selected resource is acquired, held for all the necessary time and the corresponding flag is set to 2. During the transmission, neither GST traffic nor SM/BE traffic can interrupt it.
Once the packet has been transmitted, the flag value is set back to 0, the corresponding resource is released and the BE server is informed of a free wavelength by sending it a proper message.
It is important to point up that, in case of high load, all the output wavelengths may be busy carrying GST or other SM/RT packets, thus it may happen that none of them is available to send SM/RT traffic (because we are looking for either a free resource or one carrying SM/BE traffic).

2.4.3 Statistically Multiplexed Best Effort (SM/BE) traffic description

SM/BE traffic for the standard 3LIHON architecture has been implemented in DEMOS as shown in Figure 2.3. Three new entities have

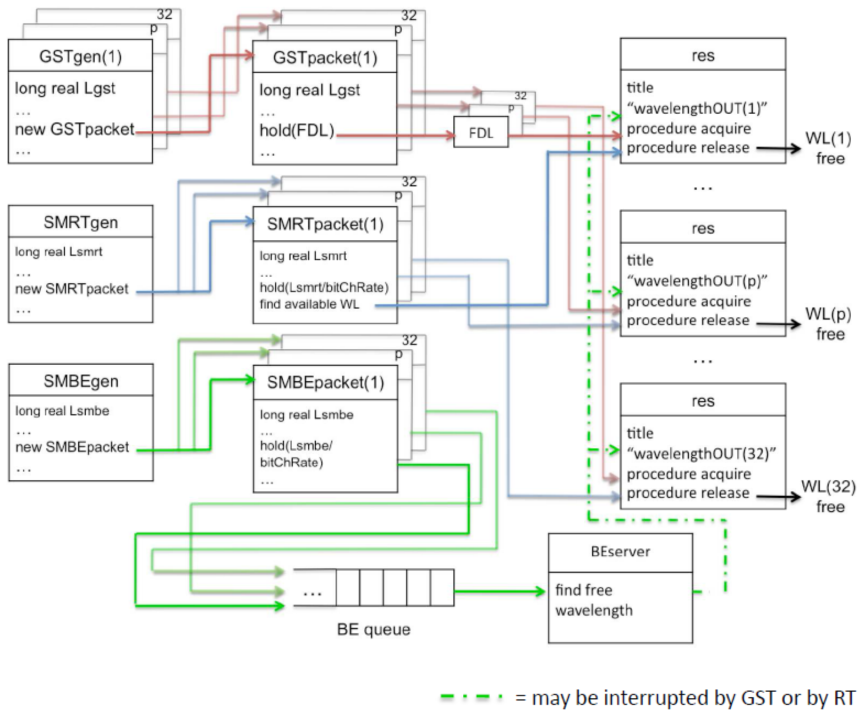

Figure 2.3: SM/BE traffic implementation [3]

been introduced.
Modifications studied in this work will be explained in section 2.6.

- **BE generator**

  A single generator is used to create SM/BE packets, with arrival times and length following the distributions showed in sections 2.3 and 2.4.

- **BE packet**

  This entity works together with the entity *BE server* shown later in order to manage the transmission of SM/BE packets in the system.
  This type of traffic has the lowest priority, so it must be implemented the preemptiveness of GST and SM/RT traffic over

SM/BE packets.
All the generated packets go into a queue *BE_Q* of class *waitq*.
As soon as a packet is scheduled, it sends a *give* message to the bin object *serverBEwaiting_wl*, to notify the server a SM/BE packet is waiting in the queue.
After having waited in the queue, the packet is served by the server, it acquires a wavelength, it marks the flag of the currently used wavelength with a value equal to 3, it holds the acquired resource until necessary and then releases it. While holding the resource, it may be interrupted by GST or SM/RT packets.
Once the transmission is completed, another *give* message is sent to *serverBEwaiting_wl*, to notify that the wavelength is now available again. Moreover, the flag value is set back to 0.
If, during the *hold* time, a GST packet requires the wavelength, it interrupts the SM/BE packet with *interrupted* value 1; if it is the case of a SM/RT packet, *interrupted* is set to 2.

- **BE server**

  This entity takes packets from the queue and forwards them into a free output wavelength.
  It starts looking for an available wavelength, beginning from the first one. If a free resource is found, then it checks if there are any packets waiting in the queue. If both conditions are true, the first packet in the queue is taken (following a FIFO policy) and forwarded through the chosen output wavelength. Finally, a *take* message is sent to the bin object *serverBEwaiting_wl*, to notify that an object from the queue has been taken.
  Now the server is ready again and can manage the forwarding of a new SM/BE packet.

2.4.4 The bin-server system

The insertion of the *bin* object *serverBEwaiting_wl* avoids the server to get stuck waiting for packets filling the queue. The bin-server system is show in Figure 2.4.

Without a *bin*, it may happen that, when the queue is empty, the

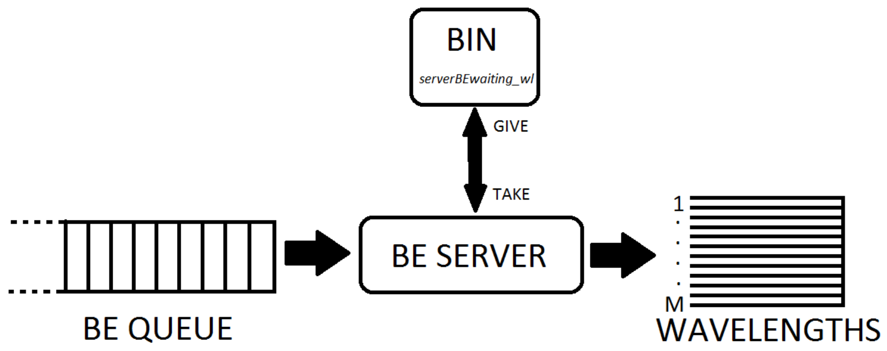

Figure 2.4: Graphical representation of the bin-server system

server keeps waiting indefinitely for an incoming packet. The *bin* has a specific role in managing the wavelengths: whenever a resource gets free, a ”token” is sent to the *bin* with the instruction *serverBEwaiting_wl.give(1)* by the GST-SM/RT-SM/BE packet which has just been transmitted (or maybe interrupted, if it is a SM/BE packet).
This signal tells the server to check the queue for some pending packets. There may also not be any packet in queue waiting to be served and, in this case, the server goes back to the *ready* state, otherwise the pending SM/BE packet is sent through the available wavelength.
A ”token” is sent to the server also whenever a SM/BE packet is scheduled. This means that there is a SM/BE packet waiting and so the queue must be checked, in order to send it.
Whenever a SM/BE packet is taken from the queue, a *serverBEwaiting_wl.take(1)* message is sent, that means that an output wavelength is now being used to transmit the SM/BE packet.

2.5 Input file

All the relevant input data needed for the simulator to run correctly are inserted through an input file. This allows us to run the same code or any of the versions presented in 2.6, while simply changing only the necessary information, thus performing a different simulation, reducing the probability of mistakes and allowing the user to keep track easily of the corresponding results and output files. E.g. the input file includes:

- the seeds array (ten seeds are used to feed the internal random number generator for ten independent sub-simulations);
- the transient and simulation time;
- the packet lengths for all the three traffic classes;
- the channel rate;
- the FDL length;
- the number of output wavelengths;
- the total load and the relative loads for each type of traffic.

An example of input file can be found in Appendix A.

2.6 DEMOS implementation of 3LIHON modified versions

We now illustrate the main differences between the three systems. Most of the code is the same for each of them, they differ only in the SM/BE management part (SM/BE server and packet behaviour).
Since 3LIHON-2R involves both retransmission and resuming, its code is thoroughly included at the end of this work, whereas for the two other types only the different parts are attached.

2.6.1 3LIHON-RS

To implement resuming, only the code regarding the SM/BE packet and server has been changed w.r.t. the standard 3LIHON node.
A new variable that keeps track of the remaining time necessary for the SM/BE packet to be completely transmitted has been introduced. If it is positive, it means the packet needs to be resumed, if it is zero it means the packet has been sent.
When a SM/BE packet is interrupted by traffic with higher priority,

it enters in an infinite hold and the time left is updated. In the meanwhile, the GST or SM/RT packet is sent.
The server checks if there is a free wavelength and if there is a packet, previously using that wavelength, that needs to be resumed. If this is the case, the server interrupts the SM/BE packet, which was in an infinite hold, with power value 3. If in that wavelength no packets need resuming, the server takes out a new packet from the queue, as usual.
The DEMOS code can be found in Appendix C.

2.6.2 3LIHON-RT

According to this scheduling algorithm, when a SM/BE packet is interrupted by GST or SM/RT traffic, it is put back in the waiting queue and then retransmitted, when the BE server takes it out.
For statistical purpose, we also keep track of the number of retransmission attempts of each single packet before its complete transmission.
The DEMOS code can be found in Appendix D.

2.6.3 3LIHON-2R

This version requires to distinguish which type of packet interrupted the SM/BE transmission. In order to take the appropriate decision, the power value of the packet causing the interruption is checked: if it is 1, it means it is a GST packet; if it is 2, it means it is a SM/RT packet.
The BE server, as in 3LIHON-RS, is the entity in charge of checking if there are some free wavelengths and packets which need to be resumed.
The DEMOS code can be found in Appendix E.

Chapter 3

Simulations and results

All the three new versions of the 3LIHON (3LIHON-RS, 3LIHON-RT and 3LIHON 2R) node have been studied.
Simulations consist in raising the system load until the highest value granting system stability, which is different for each version.
The starting load is always 50%, since lower values would probably be of little interest in real scenarios.
To establish if at a certain load the system is stable or not, we perform a simulation with two different durations. If the average length of the SM/BE server queue turns out to be approximately the same, it is possible to state that it is time-invariant and the system is proven to be stable at that load.
All simulations have some common factors:

- the seeds array, starting with the standard seed 907;
- the mean SM/RT and SM/BE packet length, 40 bytes and 1600 bytes respectively;
- the GST packet length, 40000 bytes;
- the maximum SM/RT packet length and the FDL length, 200 bytes;
- the channel rate, 1 Gbit/s;
- the number of output wavelengths, 32.

3.1 Results and discussion

Results presented below are not available for the same load values for every system. Due to the different scheduling algorithms, each system has a different maximum load, beyond which it becomes unstable.
Results at these loads are not of interest, since they are not trustworthy and not realistic. In our simulations, the SM/BE server has an infinite queue length, that means that it can potentially grow forever. Obviously, this would not be possible in a real scenario. Furthermore, some output data would have ”wrong” values, e.g. the mean delay of SM/BE packets. With and endless queue, SM/BE packets would experience higher and higher delay before being served, thus increasing the delay mean value.
For 3LIHON-RT, results show that its instability load point is slightly over 0.8, at about 0.83, so we decided to include data until load 0.8.
For 3LIHON-RS and 3LIHON-2R, the instability point is much higher: good results have been obtained up to a load of 0.92.

3.1.1 SM/BE packets mean delay

The new scheduling algorithms introduced in this work are intended to improve the performances of the standard 3LIHON node. This means mainly reducing packet loss for SM/BE packets, their delay and the utilization of the output wavelengths to achieve these results.
Delay and packet loss can be important if we think that SM/BE traffic is intended to support general data transfer and, sometimes, interactive messaging QoS [6]. This services can be loss sensitive (thus may exploit retransmission) and have some light real-time demands.

In this section we analyse how the introduced output scheduling techniques affect delay for SM/BE traffic.
Every sub-simulation returns a mean value for delay of SM/BE packets, obtained by averaging all the delays experienced by every single SM/BE packet during the sub-simulation. All these ten mean values are then averaged again, in order to get a single mean value, which has been plotted.

Figure 3.1 shows SM/BE packets delay with 95% confidence interval

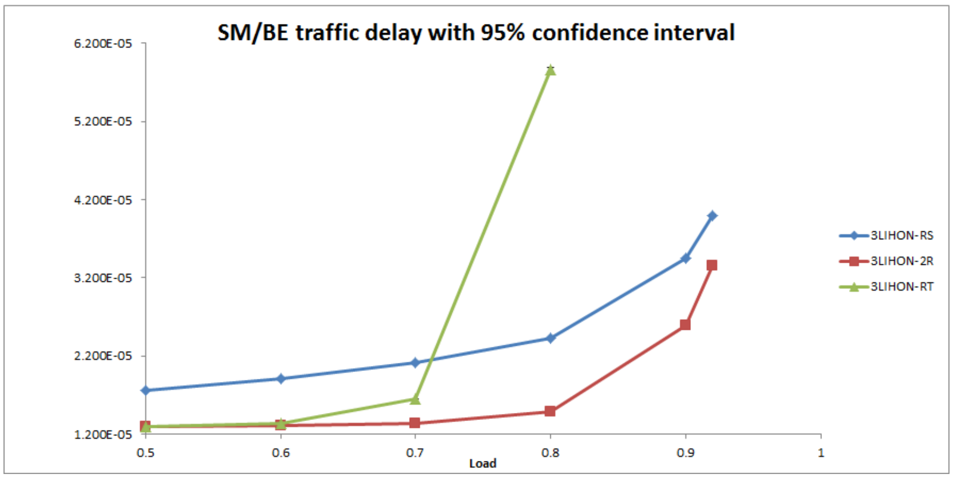

Figure 3.1: SM/BE packets delay with 95% confidence interval

for every system as a function of system load.
Confidence interval bars are not visible since its values are too small. In order to get accurate results, different simulation and transient times have been used for different loads and systems: the higher the load, the longer the time. Since 3LIHON-RT simulations usually take more time, in order to get results as accurate as for the two other systems, 3LIHON-RT transient and simulation times have been set longer.

Figure 3.1 shows that 3LIHON-RS has the highest delay until a load of 0.7 or slightly higher. This means that, at low loads, retransmitting is the best solution. GST packets are much longer than SM/RT ones. In 3LIHON-RS, even if a SM/BE packet is quickly scheduled, when an interruption by a GST packet occurs it has to wait until all the GST packet is sent. This leads to a high delay value, even in low load conditions. 3LIHON-RT and 3LIHON-2R have the possibility to retransmit the interrupted SM/BE packet through another free wavelength, which is likely to be found, since the load is low.
When the load increases, interruptions by SM/RT packets tend to be more and more frequent, causing every time a retransmission in

3LIHON-RT. Its SM/BE queue starts filling up and being longer, so SM/BE packets experience a higher delay.
3LIHON-2R is an intermediate solution between 3LIHON-RS and 3LIHON-RT. Figure 3.1 shows that its delay is always the lowest: at low loads it can compete with 3LIHON-RT, whereas at high loads its delay is much lower than 3LIHON-RT and definitely lower than 3LIHON-RS. At low loads, when interruptions are still quite rare and most of them are caused by GST packets, SM/BE packets do not experience a high delay: if the interruption is caused by a SM/RT packet, which is very short, the SM/BE interrupted packet is resumed and thus do not considerably perceive the wait; on the other hand, if the interruption is caused by a GST packet, the SM/BE packet is retransmitted and easily find a free wavelength, so, again, it does not wait for a long time.

Since 95% confidence intervals are too small to be seen on Figure 3.1, they are presented in Figure 3.2.

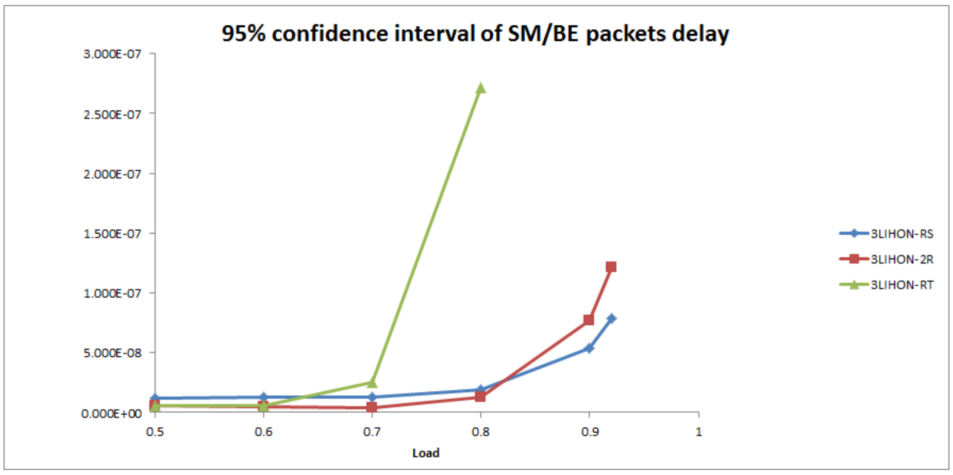

Figure 3.2: 95% confidence interval of delay for SM/BE packets

3.1.2 SM/BE queues

A closer look at the behaviour of the SM/BE queue for each system will help to better understand what stated in section 3.1.1.

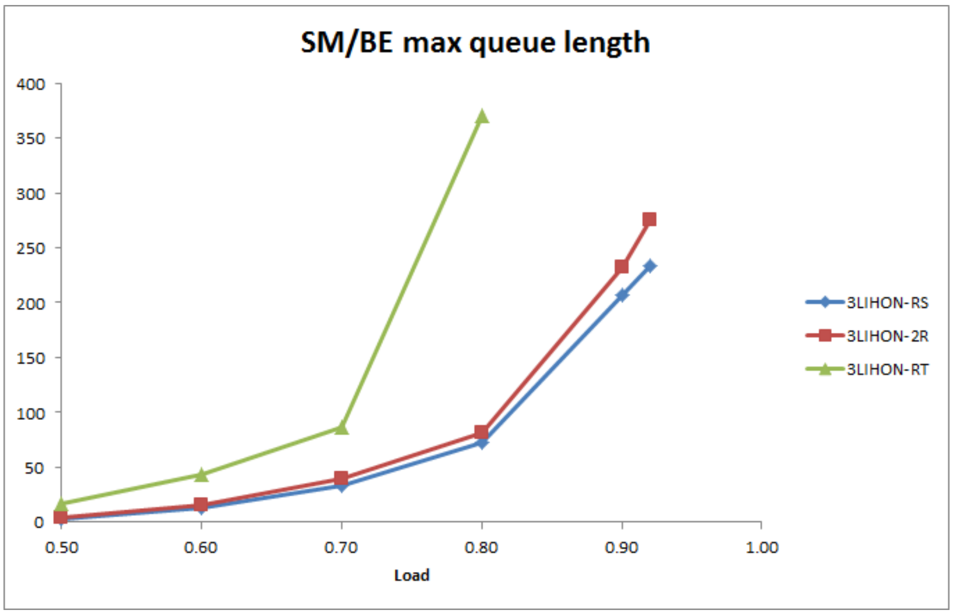

Figure 3.3: Maximum queue length for SM/BE packets

In Figure 3.3 is plotted the maximum queue length for each system, obtained with seed 234.
3LIHON-RS always has the shortest queue, because it does not implement any retransmission, so every packet is served only once. 3LIHON-RT always has the longest queue, since it retransmits every interrupted SM/BE packet. More in specific, after load 0.7 its queue length increases rapidly, because interruptions by SM/RT traffic become much more frequent (see section 3.1.5).
3LIHON-2R performance is slightly worse than 3LIHON-RS, because after load 0.8 interruptions by GST packets are just a little bit more than ones in 3LIHON-RS, as explained in section 3.1.5 and showed in Figure 3.12.
This explains why 3LIHON-RS has always the best wavelengths utilization (see section 3.1.4) but not always the shortest delay: SM/BE packets are quickly served, since the queue is short, but they suffer from interruptions by GST packets, which are long, so waiting for them for resuming the transmission has a noticeable impact on delay.

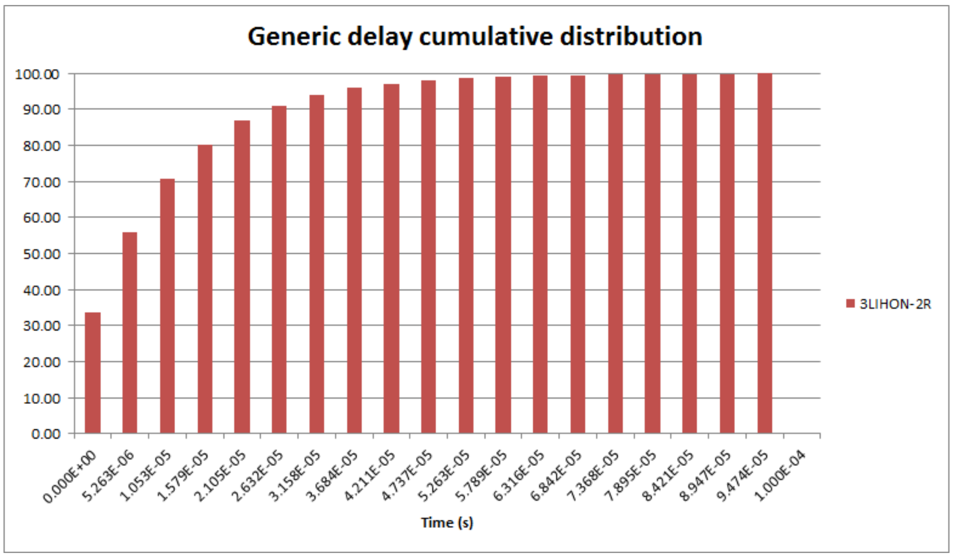

Figure 3.4: Generic cumulative distribution of delay for SM/BE traffic

We can then state that 3LIHON-2R is a good compromise between all the systems: it does not have the best performances at every load, but it is very well balanced.

3.1.3 Delay distribution of SM/BE traffic

It is useful to see how delay of SM/BE packets is distributed and how the mean value is affected by high values of delay at high loads.

Figure 3.4 shows a generic distribution of delay for SM/BE packets in a low load situation. Time is split into intervals, each represented by a bar. Values in the X-axis represent the starting time of the intervals. The last bar includes delays with a value of $1 \cdot 10^-4$ s or higher.
Only a very little amount of packets has the highest delay, because the system is not under heavy load and most of the packets find a free resource. But how does the chart change at high loads?

In figure 3.5 is showed the cumulative distribution of delay for SM/BE traffic of a 3LIHON-RS node at load 0.92. We can see that the step between the last bar and the previous is much more clear than in figure 3.4 and it is ca. 7%. The higher the difference between these two bars, the more SM/BE packets have high delay.

Figure 3.6 represents the same situation for 3LIHON-2R. In this

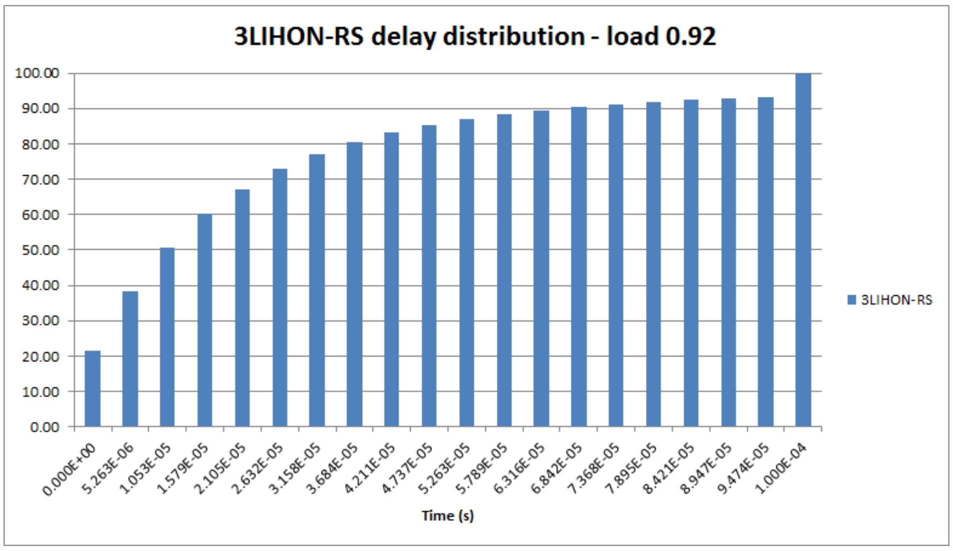

Figure 3.5: 3LIHON-RS cumulative distribution of delay for SM/BE traffic - load 0.92

case the difference is ca. 6%. This means that 1% less SM/BE packets than 3LIHON-RS has high delay at the same load, proving that 3LIHON-2R has better delay performances for SM/BE traffic than 3LIHON-RS at high loads. In other words, we can say that, at high loads, retransmitting SM/BE packets interrupted by GST traffic takes a little less time than waiting and resuming. SM/RT traffic does not influence the results, since both systems resume SM/BE packets interrupted by SM/RT packets.

In figure 3.7 is showed the cumulative distribution of delay for SM/BE traffic of a 3LIHON-RT node at load 0.8. The difference between the last two bars is clearly higher than in the two systems above, this time it is ca. 16%. This confirms the poor performances of 3LIHON-RT at high loads, even at load 0.8. It is due to its retransmission policy, which is not optimal at high loads, because retransmitted packets wait a long time in queue. The more packets waiting in queue, the

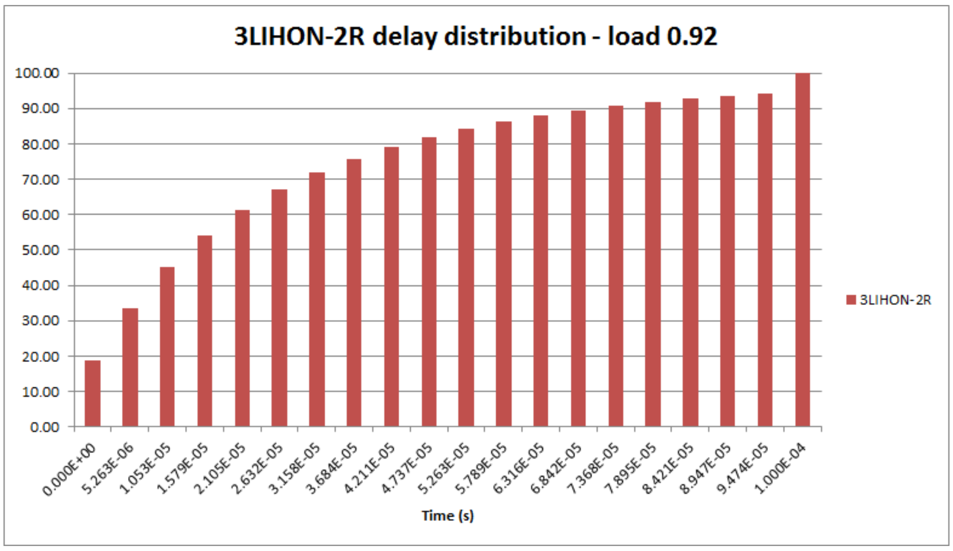

Figure 3.6: 3LIHON-2R cumulative distribution of delay for SM/BE traffic - load 0.92

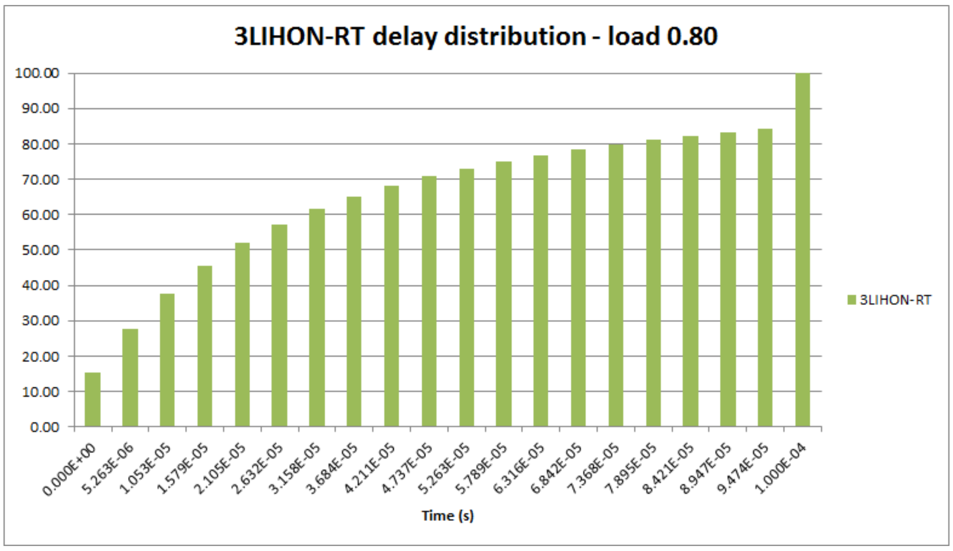

Figure 3.7: 3LIHON-RT cumulative distribution of delay for SM/BE traffic - load 0.80

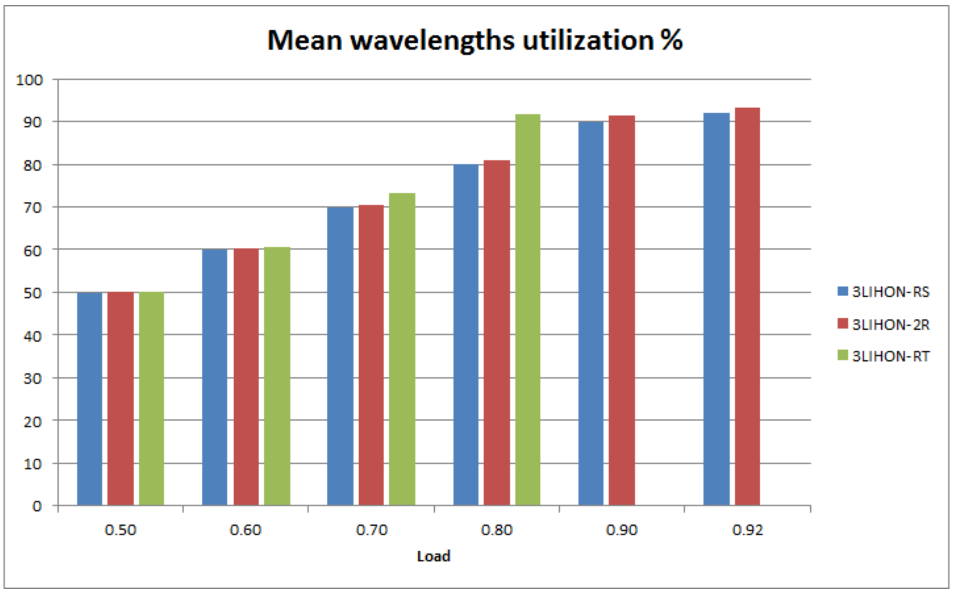

Figure 3.8: Average wavelengths utilization as a function of system load

more packets needing to be served and transmitted, the less available wavelengths, thus preventing incoming packets to be served quickly.

3.1.4 Wavelengths utilization

Figure 3.8 shows how the wavelengths are exploited in each system by considering all three types of traffic together.
Since all the considered nodes use 32 wavelengths and every simulation is made of ten sub-simulations (one per each seed), showing the detailed results would have been impractical. Results for one seed (234) have been chosen for each system and for each simulation. Then the 32 utilization values, one for each wavelength, have been averaged. This work has been done for each load.
The figure shows that wavelengths utilization values for 3LIHON-RS and 3LIHON-2R are very close to each other, but 3LIHON-2R has always a little bit higher utilization. This is due to its retransmission policy for SM/BE packets interrupted by GST traffic. This difference has its highest value of 1,32% at load 0.9. This worse performance is

balanced by its lower delay values (see section 3.1.1).
3LIHON-RT has always the highest utilization, because retransmitting a packet means wasting all the transmission time spent until the interruption occurred. At load 0.8, which is quite close to its instability point, its utilization value is 10,91% higher than 3LIHON-2R and 11,83% higher than 3LIHON-RS.
We can also observe that, until a load of 0.7, wavelengths utilization of 3LIHON-RS and 3LIHON-2R is quite close to the load value, that means that there is not a high overhead. 3LIHON-2R has a slightly higher utilization than 3LIHON-RS due to its retransmission policy. 3LIHON-RT starts having an utilization noticeably higher than the load value.
At load 0.8, the difference between 3LIHON-RS and 3LIHON-2R is more noticeable and 3LIHON-RT utilization is much higher than the load, because it is approaching its instability load, which is at about 0.84.
At load 0.92, wavelengths utilization of 3LIHON-RS is still very close to the load value, whereas 3LIHON-2R utilizes 93.34% of its resources because of retransmission of some packets.
It is also interesting to compare how every system allocates resources to the different types of traffic. Figures 3.9 and 3.10 show the percentage of time, over the total simulation time, which has been allocated to transmit respectively GST and SM/RT traffic. In the last figure, 3.11, it is showed the percentage of time used to retransmit interrupted SM/BE packets. Since only 3LIHON-2R and 3LIHON-RT implement retransmission, 3LIHON-RS is not present in this plot.

From Figure 3.9 it is possible to see the highest priority granted to GST traffic. Relative load for GST traffic is 0.6 in all simulations, so, e.g. when the total load is 0.8, GST traffic is supposed to use $0.6 * 0.8 = 48\%$ of the available resources. Figure 3.9 shows that this rule is always followed, for every system and at every load, thus proving that QoS for GST traffic is guaranteed.

Figure 3.10 shows that also for SM/RT traffic the expected wavelengths availability is assured. At higher loads, e.g 0.9 or 0.92, it is possible to see a small packet loss (see section 3.1.7 for further details).

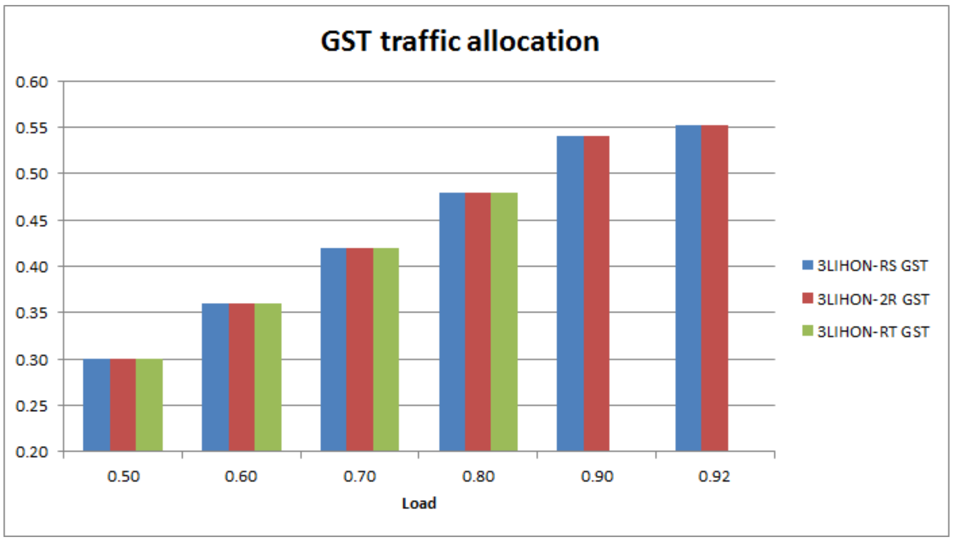

Figure 3.9: Percentage of time allocation for GST traffic

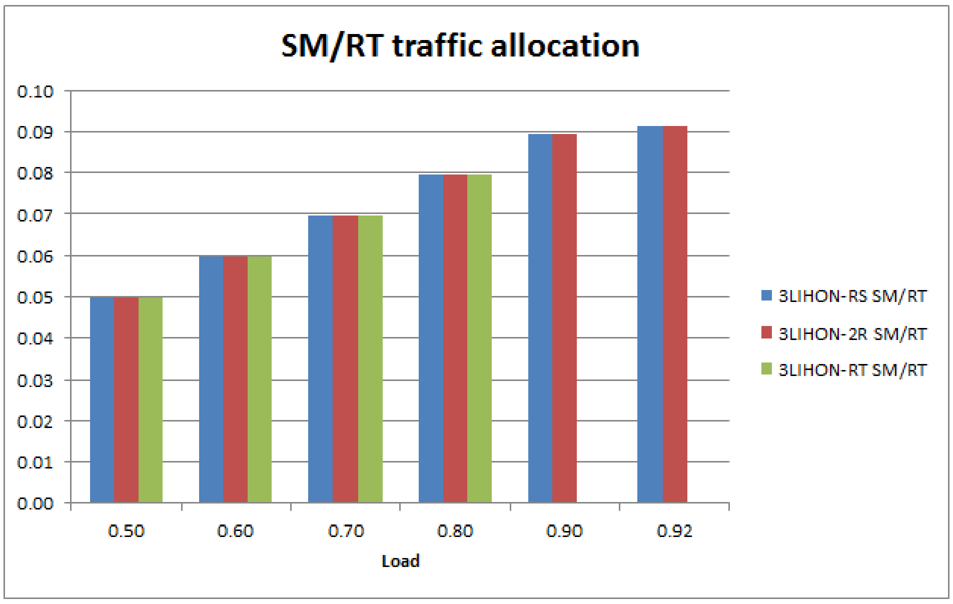

Figure 3.10: Percentage of time allocation for SM/RT traffic

Figure 3.11 shows very well the performance difference between 3LIHON-

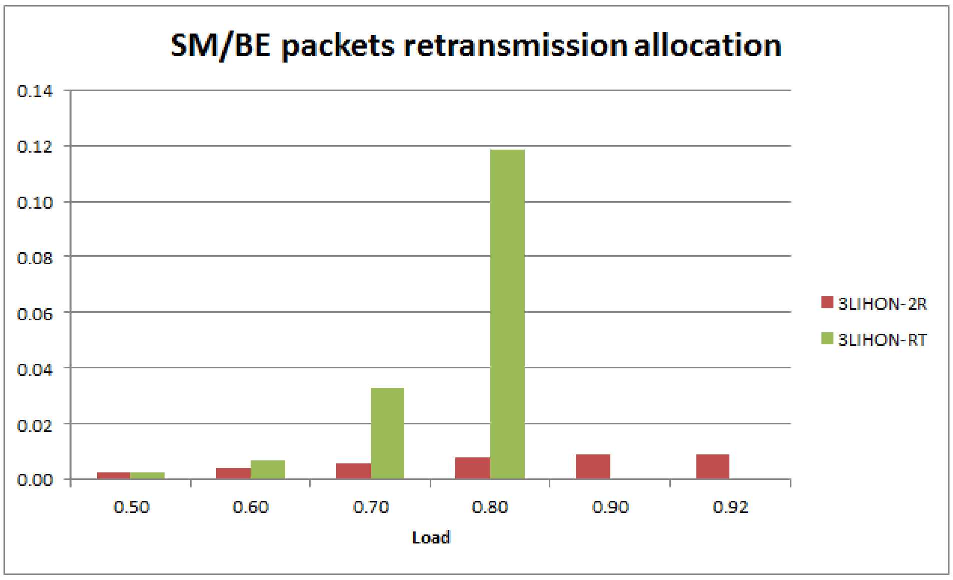

Figure 3.11: Percentage of time allocation to retransmission of interrupted SM/BE packets

RT and 3LIHON-2R, which comes from their different architectures. It shows the percentage of wavelengths utilization used to retransmit interrupted SM/BE packets. Only unsuccessful retransmissions (SM/BE packets which have been interrupted, retransmitted and then interrupted again) are taken into account. Since 3LIHON-2R retransmits only SM/BE packets interrupted by GST packets, fewer resources are necessary w.r.t. 3LIHON-RT. We can state that time spent retransmitting without success is wasted time, because all data sent during this time will be dropped.
It is possible to see that at low loads, until e.g 0.5, the two architectures achieve the same results. There are multiple reasons to explain that. The first is that GST interruptions are still relatively rare, due to the low load. Furthermore, many wavelengths are available to send retransmitted packets, so an interrupted packet easily find a new free resource where to be successfully transmitted. Interruptions caused by SM/RT traffic are a small fraction of the ones caused by GST traffic, which are the most frequent, so their impact in 3LIHON-RT on total retransmissions is very low.

When increasing the load, SM/RT packets gradually become the main cause of interruption. Since 3LIHON-RT retransmits all the interrupted SM/BE packets, its retransmission attempts increase consequently, whereas 3LIHON-2R does not suffer from this problem. On the contrary, it experiences a linear amount of time spent retransmitting, according to the linear increase of interruptions by GST packets.

3.1.5 GST and SM/RT interruptions over SM/BE traffic

The trend of interruptions by GST and SM/RT traffic over SM/BE packets is important to figure out the behaviour of each system at different loads.

Figures 3.12 and 3.13 show the percentage of interruptions by GST and SM/RT traffic respectively.
Interrupting GST or SM/RT packets are the ones which find a wave-

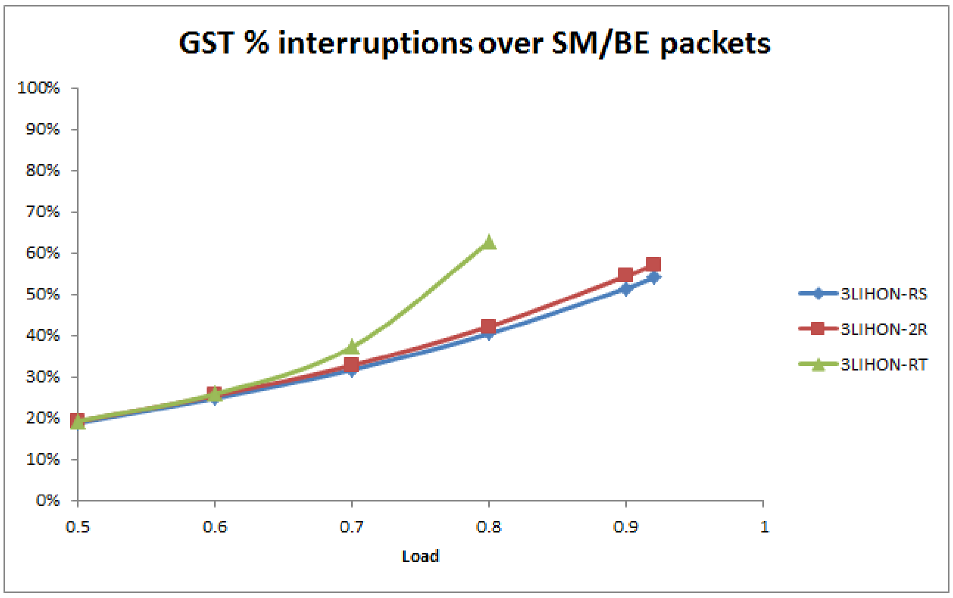

Figure 3.12: Interruptions by GST packets over SM/BE traffic

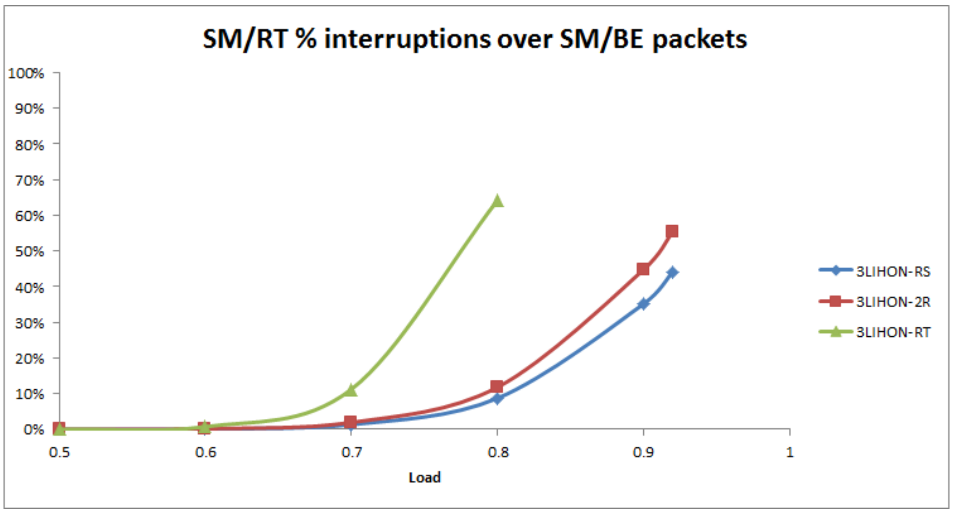

Figure 3.13: Interruptions by SM/RT packets over SM/BE traffic

length busy carrying SM/BE traffic. The number of interrupting packets is then divided by the number of generated packets of the same type to obtain the percentage.
What comes out from both plots is that interruptions increase with the increasing load, since more packets need to be transmitted and they preempt SM/BE traffic.
GST interruptions increase quite linearly, except for 3LIHON-RT, because repeated retransmissions of SM/BE packets lead to have more SM/BE packets inside the system that can be interrupted, so the probability if an interruption is higher. For the same reason, 3LIHON-2R interruption percentage is slightly over the 3LIHON-RS one.
Similar considerations can be done for interruptions caused by SM/RT packets.

3.1.6 SM/BE delay and standard deviation

Standard deviation is useful to figure out how much delay values are spread around the mean value.

Figure 3.14 shows standard deviation trend for all the 3LIHON versions as a function of load. They refer to seed 234.

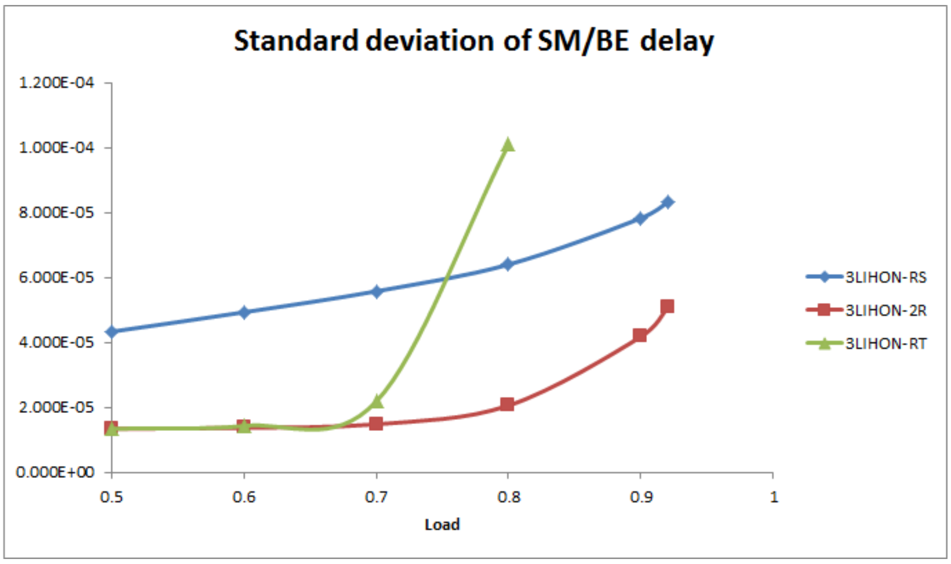

Figure 3.14: Standard deviation as a function of system load

We can see that both 3LIHON-RS and 3LIHON-2R standard deviation values have a linear trend until a load value of 0.8 and then they rapidly increase. Starting from a load of 0.8, interruptions caused by SM/RT traffic become much more frequent, whereas those caused by GST traffic increase linearly, as Figure 3.12 and Figure 3.13 show. In both systems, interruptions caused by SM/RT traffic are managed by resuming the interrupted SM/BE packet.
This let us state that increased SM/RT interruptions lead to higher delays for some SM/BE packets and this increases standard deviation as well. The implemented scheduling algorithm always attempt to use the first available wavelength, so SM/BE packets using the latest wavelengths are less prone to interruptions and preserve small delays, whereas SM/BE packets using the first wavelengths are frequently interrupted, thus having longer delays. This difference in delays duration is what lead to higher standard deviation values.

Figure 3.14 shows also that, between load values 0.8 and 0.9, 3LIHON-2R standard deviation increases more rapidly than 3LIHON-RS standard deviation. From figures 3.12 and 3.13 we can also observe that, until a load value of 0.8, the percentage of GST and SM/RT interruptions is fairly the same for 3LIHON-RS and 3LIHON-2R, but then it starts to increase more quickly for 3LIHON-2R. Since 3LIHON-2R treats SM/BE packets interrupted by GST traffic by retransmitting them (they are put back at the tail of the queue), delay for these packets will increase. Furthermore, 10% more interruptions by SM/RT packets sum up its effect to the one given by GST traffic, thus further increasing the standard deviation.

By observing figure 3.14 we can draw another interesting conclusion. 3LIHON-RS standard deviation is clearly much higher than 3LIHON-2R standard deviation, but as the load increases, they tend to be closer. When the load is not very high, SM/BE packets interrupted by GST traffic in 3LIHON-2R find a new free wavelength rather easily. This does not affect their delay very much, which remains closer to the mean value than for SM/BE packets interrupted by GST traffic and then resumed in 3LIHON-RS. GST packets are very long, so their interruption will cause the resumed SM/BE packets to have a high delay, thus increasing the standard deviation. But when the load is high, more wavelengths are busy and the waiting queue gets longer, so retransmitted SM/BE packets have a longer wait and the time difference between retransmission and resuming gets narrower.
In 3LIHON-RT, Figure 3.12 and Figure 3.13 show that interruptions by both GST and SM/RT traffic start to increase positively from a load value of 0.7 and all the SM/BE interrupted packets are retransmitted. This causes their delay to increase. Since utilization of the latest wavelengths is still relatively low, SM/BE packets being sent through these wavelength does not suffer frequent interruptions, thus experiencing much lower delay. The great difference in delay between interrupted and not interrupted SM/BE packets is what makes 3LIHON-RT standard deviation grow so quickly.

As illustrated above, the way how traffic is distributed among the wavelengths directly influences standard deviation. This lead us to think that introducing more fairness in the algorithm (e.g. using a Round-robin or a random algorithm) would probably lower standard deviation values, thus achieving better performances, especially for

3LIHON-RT scheduling. This option is discussed in section 4.2.

3.1.7 SM/RT packet loss

Since SM/RT packets have priority over SM/BE traffic, the type of output scheduling adopted does not influence their loss.
A SM/RT packet may be lost only if all the wavelengths are busy carrying other SM/RT packets or GST packets.

The results presented in Figure 3.15 have been obtained with 3LIHON-RS, seed 234. SM/RT packet loss is presented as a percentage of total generated SM/RT packets, since this number highly depends on system load. Loads over 0.92 are not present, since the system would be unstable.
For loads up to 0.6 there is no loss, then it starts increasing, but at

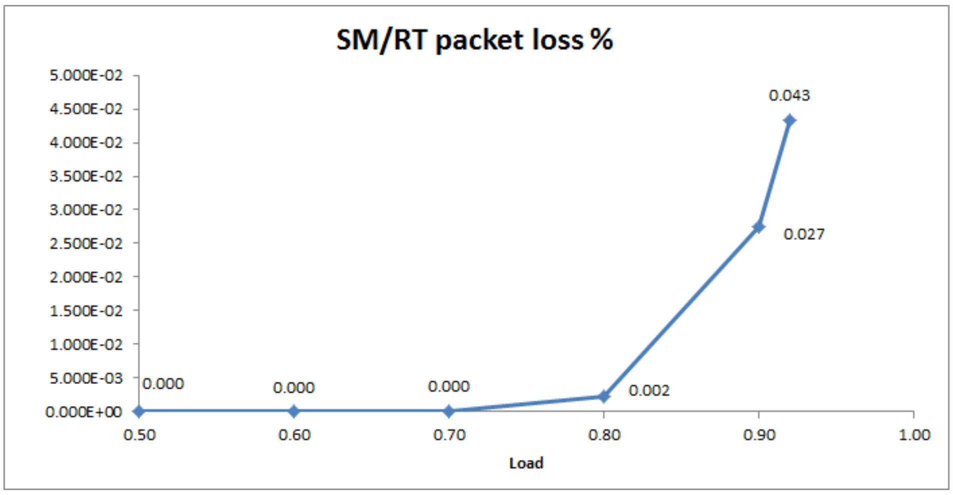

Figure 3.15: SM/RT percentage packet loss as a function of system load

high loads the percentage of lost packets is still very low.

Chapter 4

Conclusions and further works

4.1 Conclusions

In this work, three new different versions of the standard 3LIHON node have been studied.
Each of them has distinct output scheduling algorithms, which modify the SM/BE packet processing model of the original one. This also leads to new node architectures.

3LIHON-RS always resumes the transmission of SM/BE packets interrupted by GST or SM/RT traffic.
3LIHON-RT always retransmits interrupted SM/BE packets. Finally, 3LIHON-2R retransmits SM/BE packets interrupted by GST traffic and resumes the ones interrupted by SM/RT traffic.

Three different simulation models have been implemented using *Simula* programming language and its *DEMOS* context class, specifically designed for discrete event simulations. These systems have been tested under different traffic loads, starting from 0.5 up to 0.8 for 3LIHON-RT and 0.92 for 3LIHON-RS and 3LIHON-2R.

Performances of these new architectures have been analysed, paying attention especially to:

- SM/BE packets mean delay. 3LIHON-2R has proved to have the lowest mean delay for SM/BE packets. 3LIHON-RT can compete with it at low loads (below 0.6), but its performances decline very quickly after that point. 3LIHON-RS has much higher delay than 3LIHON-2R, but at very high loads this difference becomes smaller.
- SM/BE queues. 3LIHON-RS has showed to have always the shortest queue, but 3LIHON-2R can get very close to its performances for loads below 0.8. 3LIHON-RT queue is the longest and increases rapidly from load 0.7.
- delay distribution of SM/BE traffic. This time, 3LIHON-RS and 3LIHON-2R have similar results at a very high load, but the latter is still slightly better. On the other hand, 3LIHON-RT is the best at low loads, but its retransmission policy worsen its performances very soon.
- wavelengths utilization. 3LIHON-RS has the lowest resource usage at all. 3LIHON-2R gets very close to it, while 3LIHON-RT starts wasting capacity noticeably from load 0.7. It also allocates much more resources to retransmission than 3LIHON-2R from that load value.
- GST and SM/RT interruptions over SM/BE traffic. Again, SM/BE traffic in 3LIHON-RS suffers less from interruptions than in the other architectures. SM/BE packets in 3LIHON-RT start being interrupted frequently from load 0.7, whereas 3LIHON-2R performances are in-between, but closer to 3LIHON-RS ones.
- standard deviation of SM/BE delay. 3LIHON-2R has the lowest standard deviation values for SM/BE delay, very similar to 3LIHON-RT ones until load 0.7. After this point, 3LIHON-RT achieves the worst results. 3LIHON-RS starts with quite high values, but they do not increase a lot with load, so from load 0.7 its performances are better than 3LIHON-RT.
- SM/RT packet loss. Since SM/RT traffic has priority over SM/BE, the scheduling algorithm adopted does not affect this result and

it has been studied only for one system. The packet loss increases noticeably after load 0.8 but even at high loads it is very small.

The conclusion is that 3LIHON-2R is a very good trade-off between pure retransmission or resuming. It does not always have the best results, but it has a balanced behaviour as the load increases.

4.2 Further works

These new scheduling techniques can be further developed and studied.
For example, retransmission can be implemented by putting the SM/BE packets at the head of the queue, instead of at the tail. If it is a short packet that has been interrupted, it has good chances to be sent again without any other interruption, so putting it at the back of the queue will increase its delay. On the other hand, putting a long SM/BE packet at the beginning of the queue could lead to many interruptions and thus blocking all the other packets.
Another option could be deciding dynamically according to the packet length.
The drawback of these two possible developments is that reordering packets inside the queue would result in a more challenging and complex architecture.
It can also be further investigated how SM/BE packets length affects the queue length, thus changing the mean SM/BE packet length.

According to what stated in section 1.4.1, a new architecture can be implemented, using a store-and-forward version instead of the current cut-through. This should lead to simpler logics.

Another improvement could be tested, following the results illustrated in section 3.1.6: the implementation of a new algorithm which equally distributes traffic among the wavelengths. Instead of looking for the first free resource, a Round-robin algorithm could store the value of the last used wavelength and then starting its search from the next one. Another option, perhaps more challenging, could be choosing a resource randomly. These versions should result in lower standard deviation values.

List of Figures

Appendix A: Input file example

```
Seeds array:
907 234 326 104 711 523 883 113 417 656
Transient time (seconds):
0.75
Simulation time (seconds):
7.5
Mean RT packets length (bit) [40x8]:
320
Mean BE packets length (bit) [multiplier*mean_length_RT]:
40
GST packets length (bit) [multiplier*mean_length_RT]:
1000
Maximum RT packets length [multiplier*mean_length_RT]:
5
Channel rate (bit/s):
1000000000
FDL length (bit) [multiplier*mean_length_RT]:
5
FDL length (seconds) [multiplier*mean_length_RT/bitChannelRate]:
5
Number of output wavelengths:
32
Total load:
0.5
relative percentage of GST packets:
```

```
0.6
relative percentage of RT packets:
0.1
relative percentage of BE packets:
0.3
```

Appendix B: Confidence interval

It is necessary to prove and estimate the good quality of the results obtained by each simulation.
The estimated parameters are random variables, each characterized by mean value and variance. By repeating n (n=10 in this work) times each sub-simulation, we obtain n independent and identically distributed observations $\mathbf{X} = (X_1, X_2, ..., X_n)$. From this sample, we want to obtain a confidence interval for the observed parameter, α [4].
An unbiased estimate for the mean value is the sample mean, $\bar{X} = (X_1 + X_2 + ... + X_n)/n$.
Two real-valued functions of the sample, $A_1(\mathbf{X})$ and $A_2(\mathbf{X})$, define the confidence interval:

$$P\{A_1(\mathbf{X}) < \alpha \leq A_2(\mathbf{X})\} = 1 - \beta, \quad 0 < \beta < 1. \tag{1}$$

What (1) means is that a proportion $1-\beta$ of the computed confidence intervals contains the true value of α. $1 - \beta$ is called "confidence coefficient"of the confidence interval.
We now suppose the existence of a random variable $Z(\mathbf{X}, \alpha)$ having fixed and known distribution. We can find two numbers, z_1 and z_2, such that

$$P(z_1 < Z(\mathbf{X}, \alpha) \leq z_2) = 1 - \beta, \quad 0 < \beta < 1. \tag{2}$$

The observations X_i obtained from the simulation tend to follow a normal distribution. So, exploiting the central limit theorem, we can say that each observation in the sample follows a normal distribution, with mean value α and variance σ^2, both unknown.

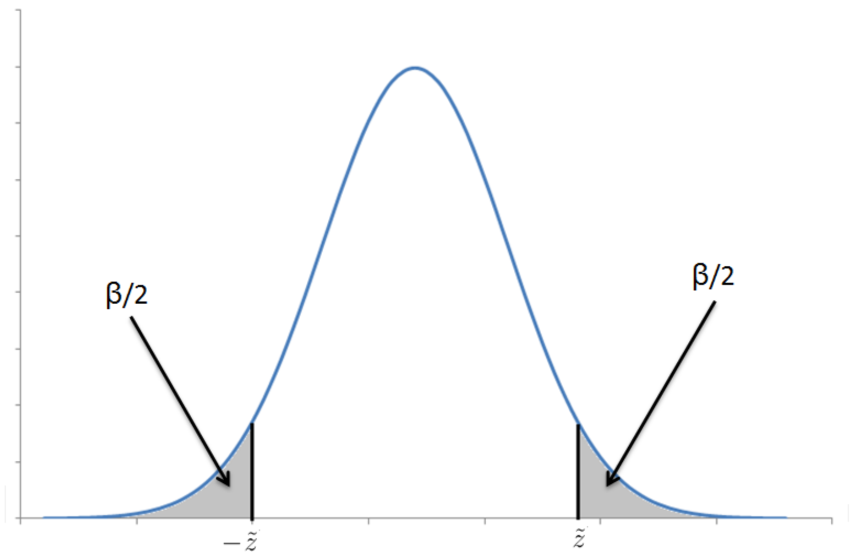

Figure 1: Gaussian distribution of $\tilde{Z}$

The arithmetic mean $\bar{X}$ follows a normal distribution too (the sum of normally distributed random variables is normally distributed), with mean value α and variance σ^2/n. We can state then that the random variable

$$\tilde{Z}(\mathbf{X}, \alpha) = \frac{\bar{X} - \alpha}{\sigma/n^{1/2}} \tag{3}$$

has the standard normal distribution, fixed and known, with mean 0 and variance 1.
By using a table of the standard normal distribution and choosing a confidence coefficient $1 - \beta$, we can obtain $\tilde{z}_1$ and $\tilde{z}_2$ such that

$$P(\tilde{z}_1 < Z \leq \tilde{z}_2) = 1 - \beta \tag{4}$$

We can choose $\tilde{z}$ so that $P(\tilde{Z} \leq \tilde{z}) = 1 - (\beta/2)$ and then $\tilde{z}_1 = -\tilde{z}$, $\tilde{z}_2 = \tilde{z}$, as shown in Figure 1. Substituting (3) into (4) we obtain:

$$P\left\{\bar{X} - (\tilde{z}\sigma/n^{1/2}) < \alpha \leq \bar{X} + (\tilde{z}\sigma/n^{1/2})\right\} = 1 - \beta \tag{5}$$

Now we have to determine σ, because its value is unknown, in order to set out the upper and lower limits in (5). It is possible to obtain σ

from the observations.
Called S^2 the sample variance, it is an unbiased estimator for σ^2:

$$S^2 = \left[\sum_{i=1}^{n}(X_i - \bar{X})^2\right] / (n-1) \tag{6}$$

Adding and subtracting α in (6) in each bracketed term, we obtain

$$E[S^2] = \sigma^2 \tag{7}$$

The random variable

$$Z(\mathbf{X}, \alpha) = \frac{\bar{X} - \alpha}{S/n^{1/2}} \tag{8}$$

has Student's distribution, if the observations X_i are normally distributed.
This variable has fixed and known distribution and it is function only of the sample of observations and α. It also has $n-1$ degrees of freedom.
Called t_n the density function, using the tables it is possible to find a number z such that $P(t_{n-1} \leq z) = 1 - (\beta/2)$. Hence

$$P(-z < Z \leq z) = 1 - \beta \tag{9}$$

Finally, we obtain the confidence interval by substituting (8) into (9):

$$P\left\{\bar{X} - (zS/n^{1/2}) < \alpha \leq \bar{X} + (zS/n^{1/2})\right\} = 1 - \beta \tag{10}$$

In this work, we choose a confidence interval of 95% ($1 - \beta = 0.95$) for the mean value estimator and $n = 10$ observations.

Appendix C: 3LIHON-RS code

Since the three simulators share great part of their code, here only relevant differences with 3LIHON-2R are showed. For full code, see Appendix E.

```
!----SMBE PACKET----;
entity class SMBEpacket;
begin

integer actual_wl;
long real Lsmbe;
long real start_BEp, time_left, start_transmission,
resume_transmission;

  serverBEwaiting_wl.give(1); !tell the server a BE packet
is waiting in the queue - invio un segnale di "presenza di
un pacchetto BE in coda" al server;
time_left:= Lsmbe/bitChannelRate;
start_BEp:= time;
BE_Q.wait;
delayBE := time - start_BEp;
be_insec := Lsmbe/bitChannelRate;
be_insec_tot := be_insec_tot + be_insec;
while time_left > 0.0 do
begin

wavelengthOUT(actual_wl).acquire(1); !transmission starts;
```

```
flag(actual_wl) := 3;
resume_BE_pointer(actual_wl) :- NONE;
SMBEpacket_pointer(actual_wl) :- this SMBEpacket;
start_transmission:= time;
if not BE_resumed(actual_wl) then
begin
BE_in_sec_offerti_al_canale:= Lsmbe/bitChannelRate;
BE_in_sec_offerti_al_canale_TOT:= BE_in_sec_offerti_al_canale_TOT+
BE_in_sec_offerti_al_canale;
BEp_intoWL := BEp_intoWL + 1;
end;
hold(time_left);
wl_BE_util(actual_wl):= wl_BE_util(actual_wl) + (time - start_transmission);
SMBEpacket_pointer(actual_wl) :- NONE;
wavelengthOUT(actual_wl).release(1); !packet transmitted or
interrupted;
if flag(actual_wl) = 3 then flag(actual_wl):= 0;
if not BE_resumed(actual_wl) then BEp_outtoWL := BEp_outtoWL
+ 1;
if interrupted = 1 then counter_BEpacket_interrupted_by_GSTp
:= counter_BEpacket_interrupted_by_GSTp + 1;
if interrupted = 2 then counter_BEpacket_interrupted_by_RTp
 := counter_BEpacket_interrupted_by_RTp + 1;


if interrupted = 0 then
begin
time_left:= 0.0;!whole packet transmitted (no interruptions);
BE_resumed(actual_wl):= false;
resume_BE_pointer(actual_wl) :- NONE; !if I'm here, it should
already be "none";
!serverBEwaiting_wl.give(1);!packet transmitted --> signal
to BE server that wl is now available;
counter_BEpacket_successful:= counter_BEpacket_successful +
1;
BEsuccessfull_in_sec := Lsmbe/bitChannelRate;
BEsuccessfull_in_sec_TOT := BEsuccessfull_in_sec_TOT + BEsuccessfull_in_sec;
```

```
BEdelay_hist.update(time - start_BEp);
BE_lifetime:= time - start_BEp; !delay of the BE packet, from
generation to the end of transmission;
total_BE_lifetime:= total_BE_lifetime + BE_lifetime;
total_BE_lifetime2:= total_BE_lifetime2 + BE_lifetime**2;
end;

if interrupted > 0 then
begin

!interrupt for statistical;
if not BE_resumed(actual_wl) then counter_BEpacket_resumed
 := counter_BEpacket_resumed + 1; !count  interrupted packets
but multiple interruptions of the same packet are not counted;
if not BE_resumed(actual_wl) then if interrupted = 1 then
interruptedByGST:=interruptedByGST+1;
if not BE_resumed(actual_wl) then if interrupted = 2 then
interruptedByRT:=interruptedByRT+1;
BE_resumed(actual_wl):= true;
counter_BEpacket_interrupted := counter_BEpacket_interrupted
+ 1;
BEinterrupted_in_sec := Lsmbe/bitChannelRate;
     BEinterrupted_in_sec_TOT := BEinterrupted_in_sec_TOT +
BEinterrupted_in_sec;
     total_retransmission(actual_wl):=total_retransmission(actual_wl)
     +1;
resume_BE_pointer(actual_wl) :- this SMBEpacket;
time_left:= time_left - (time - start_transmission);
hold(sim_time+1);!infinite hold: the interrupted packet waits
to be scheduled again by the server;
interrupted:=0;
end;

end***while***;


end***SMBEpacket***;
```

```
!----SMBE SERVER----;
entity class BEserver;
begin
integer i;
ref(SMBEpacket) p_SMBEpacket;

START:
serverBEwaiting_wl.take(1);

for i:= 1 step 1 until OUTPUT_WL do
begin


!wavelength free? Some packet to resume in this wavelength?;
if flag(i) = 0 and resume_BE_pointer(i) =/= none then
begin
p_SMBEpacket :- resume_BE_pointer(i);
p_SMBEpacket.interrupt(3);!I use interrupt (instead of schedule)
because the packet is in an endless  hold;
goto START;
end;

if flag(i) = 0 and BE_Q.length > 0 then
begin

p_SMBEpacket :- BE_Q.coopt;
p_SMBEpacket.actual_wl := i;
p_SMBEpacket.schedule(0.0);
BEp_inQ:= BEp_inQ + 1;
goto START;
end;


end***for***;

goto START;
```

```
end***BEserver***;
```

Appendix D: 3LIHON-RT code

Since the three simulators share great part of their code, here only relevant differences with 3LIHON-2R are showed. For full code, see Appendix E.

```
!----SMBE PACKET----;
entity class SMBEpacket;
begin

integer actual_wl;
integer retransmission_attempt; !Keeps track of how many times
a BE packet is retransmitted;
long real Lsmbe;
long real start_BEp, start_BEp2, time_left, start_transmission,
resume_transmission;

be_insec := Lsmbe/bitChannelRate;
be_insec_tot := be_insec_tot + be_insec;
RETRANSMIT:
  serverBEwaiting_wl.give(1); !tell the server a BE packet
is waiting in the queue - invio un segnale di "presenza di
un pacchetto BE in coda" al server;
start_BEp:= time;
if retransmission_attempt=0 then start_BEP2:= time; !used only
to compute BE lifetime;
BE_Q.wait;
delayBE := time - start_BEp;
```

```
flag(actual_wl) := 3;
SMBEpacket_pointer(actual_wl) :- this SMBEpacket;
wavelengthOUT(actual_wl).acquire(1); !transmission starts;
if retransmission_attempt>0 then count_retransmissions:=count_retransmissions+1;
start_transmission:= time;
BE_in_sec_offerti_al_canale:= Lsmbe/bitChannelRate;
BE_in_sec_offerti_al_canale_TOT:= BE_in_sec_offerti_al_canale_TOT+
BE_in_sec_offerti_al_canale;
if retransmission_attempt=0 then BEp_intoWL := BEp_intoWL +
1;
hold(Lsmbe/bitChannelRate);
wavelengthOUT(actual_wl).release(1); !packet transmitted or
interrupted;
SMBEpacket_pointer(actual_wl) :- NONE;
if retransmission_attempt=0 then BEp_outtoWL := BEp_outtoWL
+ 1;
if interrupted = 1 then counter_BEpacket_interrupted_by_GSTp
:= counter_BEpacket_interrupted_by_GSTp + 1;
if interrupted = 2 then counter_BEpacket_interrupted_by_RTp
 := counter_BEpacket_interrupted_by_RTp + 1;


if interrupted = 0 then
begin

serverBEwaiting_wl.give(1);!packet transmitted --> signal to
BE server that wl is now available;
counter_BEpacket_successful:= counter_BEpacket_successful +
1;
wl_BE_util(actual_wl):= wl_BE_util(actual_wl) + Lsmbe/bitChannelRate;
BEsuccessfull_in_sec := Lsmbe/bitChannelRate;
BEsuccessfull_in_sec_TOT := BEsuccessfull_in_sec_TOT + BEsuccessfull_in_sec;
BEdelay_hist.update(time - start_BEp2);
BE_lifetime:= time - start_BEp2; !delay of the BE packet, from
generation to the end of transmission;
total_BE_lifetime:= total_BE_lifetime + BE_lifetime;
total_BE_lifetime2:= total_BE_lifetime2 + BE_lifetime**2;
```

```
if flag(actual_wl) = 3 then flag(actual_wl):= 0;
end;

if interrupted > 0 then
begin
!interrupt for statistical;
if retransmission_attempt=0 then counter_BEpacket_resumed
:= counter_BEpacket_resumed + 1; !count  interrupted packets,
but multiple interruptions of the same packet are not counted;
if retransmission_attempt=0 then if interrupted = 1 then
interruptedByGST:=interruptedByGST+1;
if retransmission_attempt=0 then if interrupted = 2 then
interruptedByRT:=interruptedByRT+1;
counter_BEpacket_interrupted := counter_BEpacket_interrupted
+ 1;
BEinterrupted_in_sec := Lsmbe/bitChannelRate;
BEinterrupted_in_sec_TOT := BEinterrupted_in_sec_TOT +
BEinterrupted_in_sec;
total_retransmission(actual_wl):=total_retransmission(actual_wl)+1;
retransmission_attempt:= retransmission_attempt + 1;
interrupted:=0;
time_wasted(actual_wl):= time_wasted(actual_wl) + (time -
start_transmission);
if flag(actual_wl) = 3 then flag(actual_wl):= 0;
GOTO retransmit;
end;



end***SMBEpacket***;

!----SMBE SERVER----;
entity class BEserver;
begin
integer i;
ref(SMBEpacket) p_SMBEpacket;
```

```
START:

for i:= 1 step 1 until OUTPUT_WL do
begin


!wavelength free? Some packet to resume in this wavelength?;
!if flag(i) = 0 and resume_BE_pointer(i) =/= none then
begin
p_SMBEpacket :- resume_BE_pointer(i);

if flag(i) = 0 then if BE_Q.length > 0 then
begin

p_SMBEpacket :- BE_Q.coopt;
p_SMBEpacket.actual_wl := i;
p_SMBEpacket.schedule(0.0);
BEp_inQ:= BEp_inQ + 1;
end;


end***for***;

serverBEwaiting_wl.take(1);
goto START;

end***BEserver***;
```

Appendix E: 3LIHON-2R code

```
BEGIN

integer count_observations, i;
long real array PLPrt(0:9);
long real array PLPrt2(0:9);
long real array PLPbe(0:9);
long real array PLPbe2(0:9);
long real PLP_RT, PLP_RT2;
long real ao_be_element, ao_be2_element;
long real as_be_element, as_be2_element;
long real pigreco_s_element, pigreco_s2_element;
long real PLP_BE, PLP_BE2; !rapporto tra quanti pacchetti BE
sono stati interrotti e quanti si sono messi in coda, quindi
hanno avuto accesso alla WL;
long real PLP_resumingBE;
long real msum, m2sum; !used to calculate confidence interval
for the mean value of delay of BE packets;
long real vsum, v2sum;
long real SUM_PLPrt, sum_meanPLPrt;
long real SUM_PLPrt2, sum_meanPLPrt2;
long real SUM_PLPbe, sum_meanPLPbe;
long real SUM_PLPbe2, sum_meanPLPbe2;
long real array vsum_vettore(0:9);
long real array v2sum_vettore(0:9);
long real array msum_vettore(0:9);
long real array m2sum_vettore(0:9);
long real sum_vsum_vettore, sum_v2sum_vettore;
long real sum_msum_vettore, sum_m2sum_vettore;
long real deviazione_std_rt;
long real deviazione_std_be;
long real deviazione_std_delay;
long real ConfInter95rt;
long real ConfInter95be;
long real ConfInter95_delay;
long real lower_rt;
long real lower_be;
long real lower_delay;
long real upper_rt;
long real upper_be;
long real upper_delay;
long real adjustDELTA_rt;
long real adjustDELTA_be;
long real adjustDELTA_delay;


for count_observations:= 0 step 1 until 9 do
begin
!Input data;
integer array my_seeds(0:9);
long real length_GST, mean_length_RT, mean_length_BE;
integer OUTPUT_WL, h; !number of fiber;
long real max_mean_length_RT, max_mean_length_BE;
long real bitChannelRate; ![bit/s];
long real rho_gst, rho_rt, rho_be, total_load;
        long real A, B;
        long real sim_time, transient_time; !il transient time
serve ad es a riempire le code coi BE e le wl coi GST (che
sono i  pacchetti piu' lunghi) e a mettere a regime il sistema.
Trascorso questo tempo,resetto SOLO le statistiche, perche'
ovviamente all'inizio sono "traviate" da valori che sono ancora
parzialmente a zero;
        long real FDL, FDLinBIT;
        long real num_pRT_sec, num_pBE_sec;

          ref (infile) InputFile;

InputFile:- new infile ("input.txt");
InputFile.Open(Blanks(180));


outtext("                                                ");
outimage;
outtext("******************INPUT DATA*******************");
outimage;

!Initialization: seeds array;
InputFile.InImage;
OutText(InputFile.InText(80));
for h:=0 step 1 until 9 do
begin
my_seeds(h):=InputFile.InInt;
OutInt(my_seeds(h),4);
OutText("   ");
end***for***;
OutImage;

!Transient time;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
transient_time:=InputFile.InReal;
OutFix(transient_time, 3, 6);
OutImage;

!Simulation time;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
sim_time:=InputFile.InReal;
```

```
OutFix(sim_time, 3, 6);
OutImage;OutImage;

!Mean RT packets length;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
mean_length_RT :=InputFile.InReal;
OutFix(mean_length_RT, 5, 15);
OutImage;

!Mean BE packets length;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
mean_length_BE :=InputFile.InReal*mean_length_RT;
OutFix(mean_length_BE, 2, 15);
OutImage;

!GST packets length;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
length_GST :=InputFile.InReal*mean_length_RT;
OutFix(length_GST, 5, 15);
OutImage;

!Maximum RT packets length;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
max_mean_length_RT :=InputFile.InReal*mean_length_RT;
OutFix(max_mean_length_RT, 2, 15);
OutImage;OutImage;

outtext("Mean RT packets length (bytes):"); outimage;
outfix(mean_length_RT/8,2,8); outimage;
```

```
outtext("Mean BE packets length (bytes):"); outimage;
outfix(mean_length_BE/8,2,8); outimage;
outtext("Mean GST packets length (bytes):"); outimage;
outfix(length_GST/8,2,8); outimage;OutImage;

!Channel rate;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
bitChannelRate :=InputFile.InReal;
OutFix(bitChannelRate, 3, 20);
OutImage;
outtext("Channel rate (Gb/s):"); outimage;
outfix(bitChannelRate/1000000000,2,5); outimage;OutImage;

!FDL length (bit);
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
FDLinBIT :=InputFile.InReal*mean_length_RT;
OutFix(FDLinBIT, 3, 10);
OutImage;

!FDL length (sec);
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
FDL :=InputFile.InReal*mean_length_RT/bitChannelRate;
OutFix(FDL, 10, 12);
OutImage;OutImage;

!Number of output wavelengths;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
OUTPUT_WL :=InputFile.InInt;
OutInt(OUTPUT_WL, 2);
```

```
OutImage;OutImage;

!Total load;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
total_load :=InputFile.InReal;
OutFix(total_load, 2, 5);
OutImage;

!relative percentage of GST packets;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
rho_gst :=InputFile.InReal;
OutFix(rho_gst, 3, 6);
OutImage;

!relative percentage of RT packets;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
rho_rt :=InputFile.InReal;
OutFix(rho_rt, 3, 6);
OutImage;

!relative percentage of BE packets;
InputFile.InImage;
OutText(InputFile.InText(80));
InputFile.InImage;
rho_be :=InputFile.InReal;
OutFix(rho_be, 3, 6);
OutImage;

         InputFile.Close;
```

```
A:= (rho_GST*total_load*bitChannelRate)/length_GST;![Rate,
pkt/sec];
B:= (1/A-length_GST/bitChannelRate); !B[sec] is the mean value
of a negexp distribution;

num_pRT_sec:= (rho_RT*total_load*bitChannelRate)/mean_length_RT;
 ![Rate, pkt/sec];
          num_pBE_sec:=
          (rho_BE*total_load*bitChannelRate)/mean_length_BE;
 ![Rate, pkt/sec];



!outtext("            SIMULATION TIME [s]");
!outfix(sim_time,2,20);
!outimage;
!outtext("            TRANSIENT TIME [s]");
!outfix(transient_time,2,20);
!outimage;
!outtext("            NUMBER OF OUTPUT WAVELENGTHS");
!outfix(OUTPUT_WL,0,10);
!outimage;
!outtext("            BIT CHANNEL RATE [bit/s]");
!outfix(bitChannelRate,0,20);
!outimage;

!outtext("            GST [bytes]");
!outfix(length_GST/8,0,20);
!outimage;
!outtext("            RT [bytes]");
!outfix(mean_length_RT/8,0,20);
!outimage;
!outtext("            BE [bytes]");
!outfix(mean_length_BE/8,0,20);
!outimage;

outtext("                                                ");
```

```
outimage;




          BEGIN

EXTERNAL CLASS DEMOS="demos.atr";
DEMOS
BEGIN

integer p, k, i, u;
integer cont_res; !reset the arrays after the transient period;

!Statistic parameters for GST traffic;
integer array counter_GSTpacket_generate(1:OUTPUT_WL);!counts
generated GST packets for each wavelength;
integer array counter_GSTpacket_successful(1:OUTPUT_WL);!GST
packets successfully transmitted;
integer GSTp_intoWL, GSTp_outtoWL, GST_tot, GST_successful,
GST_wl_busy;
long real gst_insec, gst_insec_tot, gst_total_time;

!Statistic parameters for RT traffic;
integer counter_RTpacket_generate, counter_RTpacket_successful,
counter_RTpacket_lost, RT_wl_directly_free,  RT_wl_busy, RTp_intoWL,
RTp_outtoWL;
long real rt_insec, rt_insec_tot, rt_total_time;

!Statistic parameters for BE traffic;
integer counter_BEpacket, counter_BEpacket_generate, counter_BEpacket_successful,
BEp_intoWL, BEp_outtoWL;
integer counter_BEpacket_interrupted, counter_BEpacket_interrupted_by_GSTp,
counter_BEpacket_interrupted_by_RTp;
long real be_insec, be_insec_tot, BE_lifetime, total_BE_lifetime,
total_BE_lifetime2;
integer BEp_inQ, counter_BEpacket_resumed;
```

```
boolean array BE_resumed(1:OUTPUT_WL); !true if the packet
will be resumed, false if the packet has never been  interrupted;
integer interruptedByRT, interruptedByGST; !count how many
BE packets are interrupted. Multiple interruptions of  the
same packet count as one;
long real BEgen_in_sec, BEgen_in_sec_TOT; !duration of the
generated BE packet in seconds;
long real BEsuccessfull_in_sec, BEsuccessfull_in_sec_TOT;
long real BE_in_sec_offerti_al_canale,
BE_in_sec_offerti_al_canale_TOT;
long real BEinterrupted_in_sec, BEinterrupted_in_sec_TOT,
be_total_time;
integer array total_retransmission(1:OUTPUT_WL); !counts total
number of retransmission for each wl;
long real array time_wasted(1:OUTPUT_WL);

long real sum_counter_GSTpacket_generate,
sum_counter_GSTpacket_successful, sum_wl_gst_util, sum_wl_rt_util,
sum_wl_be_util, sum_time_wasted, sum_total_retransmission;

integer TOTAL_P_GENERATED, TOTAL_P_intoWL; !total generated
packets - packets using a certain resource;
long real somma, somma2; !total BE packets delay and its square;
long real delayBE;
boolean array FDL_in_use(1:OUTPUT_WL);!is the considered wl
busy?;
integer array flag(1:OUTPUT_WL);!one flag for each wl. Values:
1->GST, 2-> RT, 3-> BE;
ref(SMBEpacket) array    SMBEpacket_pointer(1:OUTPUT_WL);!is
the wl transmitting a BE packet? Used for GST  packets;
ref(SMBEpacket) array    resume_BE_pointer(1:OUTPUT_WL);


!tracks utilization for each wl;
long real array wl_GST_util(1:OUTPUT_WL);
long real array wl_RT_util(1:OUTPUT_WL);
long real array wl_BE_util(1:OUTPUT_WL);
```

```
!resources;
ref(res) array          wavelengthOUT(1:OUTPUT_WL);
ref(bin)                serverBEwaiting_wl;
ref(waitQ)  BE_Q;



!GST traffic;
ref(rdist) array nextGST(1:OUTPUT_WL); !array of GST packets;
ref(rdist) array lengthGST(1:OUTPUT_WL); !array containing
length of GST packets;

!RT traffic;
ref(rdist) nextSMRT;
ref(rdist) lengthSMRT;

!SMBEtraffic;
ref(rdist) nextSMBE;
ref(rdist) lengthSMBE;
ref(histogram) BEdelay_hist;

ref (outfile) transient;


!----GST PACKET GENERATOR----;
entity class GSTgen(p); integer p;!p is the wavelength;
begin

long real Lgst;!GST packet length;
ref(GSTpacket) p_GSTp; !the GST packet;

LOOP:
counter_GSTpacket_generate(p):= counter_GSTpacket_generate(p)
+ 1;!update generated GST packet counter;
p_GSTp:- new GSTpacket("GSTpacket ", p);!create a new GST packet;
```

```
Lgst:= lengthGST(p).sample;!get a random length;
p_GSTp.Lgst:= Lgst;!set the GST packet length;
p_GSTp.schedule(0.0);!generate it now!;
!two GST packets can't overlap, so, before the next generation
of a packet,
I must be sure the current one has been transmitted
(Lgst/bitChannelRate).Between two packets, there is
a negexp time interval nextGST(p).sample).
Remember: there is a GST generator for each
wavelength; hold(Lgst/bitChannelRate +
nextGST(p).sample);
goto LOOP;

end***GSTgen(p)***;


!----GST PACKET----;
entity class GSTpacket(p); integer p;
begin
long real Lgst;!GST packet length;
ref (SMBEpacket) BE_p;

flag(p):=1;!flag=1 means GST packet;
FDL_in_use(p):=true;!set the current wl to busy;

!this is the most frequent case, because FDL is equal to the
SM/RT max length
and usually RT packets are very short, so FDL is frequently
shorter than GST packets;
if Lgst/bitChannelRate >= FDL then
begin
hold(FDL);!FDL now full;
if SMBEpacket_pointer(p) =/= none then
begin
GST_wl_busy := GST_wl_busy + 1;
BE_P:- SMBEpacket_pointer(p);
BE_P.interrupt(1);!BE packet interrupted;
```

```
end;
wavelengthOUT(p).acquire(1);
GSTp_intoWL:= GSTp_intoWL + 1;
hold(Lgst/bitChannelRate-FDL);!first transmit the part of the
packet that overflows the FDL;
wl_GST_util(p):= wl_GST_util(p) + Lgst/bitChannelRate-FDL;
FDL_in_use(p):= false;!the beginning of the FDL is now free;
hold(FDL);!transmit the part of the packet that is as long
as the FDL;
wl_GST_util(p):= wl_GST_util(p) + FDL;
end
else
begin
hold(Lgst/bitChannelRate);!wait for the whole packet to enter
the FDL;
FDL_in_use(p):= false;!the beginning of the FDL is now free;
hold(FDL-Lgst/bitChannelRate);!wait for the packet to reach
the end of the FDL;
if SMBEpacket_pointer(p) =/= none then
begin
GST_wl_busy := GST_wl_busy + 1;
BE_P:- SMBEpacket_pointer(p);
BE_P.interrupt(1);!BE packet interrupted;
end;
wavelengthOUT(p).acquire(1);
GSTp_intoWL:= GSTp_intoWL + 1;
hold(Lgst/bitChannelRate);!send the packet;
wl_GST_util(p):= wl_GST_util(p) + Lgst/bitChannelRate;
end;
wavelengthOUT(p).release(1);
gst_insec:= Lgst/bitChannelRate;!duration of the GST packet;
gst_insec_tot:= gst_insec_tot+gst_insec;!total duration of
GST packets;
GSTp_outtoWL := GSTp_outtoWL  + 1;
if not FDL_in_use(p) then flag(p):=0;
serverBEwaiting_wl.give(1);  !packet transmitted --> signal
to BE server that wl is now available;
```

```
counter_GSTpacket_successful(p):=counter_GSTpacket_successful(p)+1;
end***GSTpacket(p)***;


!----SMRT PACKET GENERATOR----;
entity class SMRTgen;
begin
long real Lsmrt;
ref(SMRTpacket) p_SMRTp;

LOOP:
counter_RTpacket_generate := counter_RTpacket_generate + 1;
Lsmrt:= lengthSMRT.sample;
p_SMRTp:- new SMRTpacket("SMRTpacket ");
!NegExp can potentially generate very long packets:
I cut them off, because they must have a max length, used to
fix the FDL length;
if Lsmrt > max_mean_length_RT then Lsmrt := max_mean_length_RT;
p_SMRTp.Lsmrt := Lsmrt;
p_SMRTp.schedule(0.0);
hold(nextSMRT.sample);!now the gen waits some time, it's the
time between two following RT packets;
goto LOOP;
end***SMRTgen***;


!----SMRT PACKET----;
entity class SMRTpacket;
begin
long real Lsmrt;
integer interrupt_wl, i;
ref(SMBEpacket) BE_p;
counter_BEpacket:= counter_BEpacket+1; !IMO it should be RTpacket.
Anyway it is used correctly later;

for i:= 1 step 1 until OUTPUT_WL do
begin
```

```
if flag(i) = 0 then!the wavelength is free;
begin
RT_wl_directly_free := RT_wl_directly_free + 1;
goto USE_FREE_WL;
end;

!il controllo della presenza di pacchetti BE qui e' fatto diversamente
che dal caso GST
perche' il pacchetto GST non deve cercare una wl libera, ma
va dritto a quella che gli
e' stata assegnata. Invece un pacchetto RT deve prima cercare
una wl libera;

if (flag(i) = 3 and interrupt_wl = 0) then interrupt_wl :=
i;!the wl is currently used by BE traffic;


end***for***;
if interrupt_wl > 0 then !interrupt BE traffic;
begin
BE_P:- SMBEpacket_pointer(interrupt_wl);
BE_P.interrupt(2);
i:= interrupt_wl;!when i gets out of the "for" cycle above,
i is OUTPUT_WL and interrupt_wl is the first  interruptable
wl;
RT_wl_busy := RT_wl_busy + 1;
goto USE_FREE_WL;
end
else counter_RTpacket_lost:= counter_RTpacket_lost +1; !no
wl free-->RT packet dropped;
goto FINE;
USE_FREE_WL:
wavelengthOUT(i).acquire(1);
RTp_intoWL:= RTp_intoWL + 1;
flag(i) := 2;
hold(Lsmrt/bitChannelRate);!transmit RT packet;
wl_RT_util(i):= wl_RT_util(i) + Lsmrt/bitChannelRate;
```

```
wavelengthOUT(i).release(1);
rt_insec:= Lsmrt/bitChannelRate;
rt_insec_tot := rt_insec_tot+rt_insec;
RTp_outtoWL := RTp_outtoWL + 1;
if flag(i) = 2 then flag(i) := 0;
!send token to BE server;
serverBEwaiting_wl.give(1);   !packet transmitted --> signal
to BE server that wl is now available;
counter_RTpacket_successful:= counter_RTpacket_successful +
1;
FINE:
end***SMRTpacket***;


!----SMBE PACKET GENERATOR----;
entity class SMBEgen;
begin
ref(SMBEpacket) p_SMBEp;
long real Lsmbe;

LOOP:
counter_BEpacket_generate:= counter_BEpacket_generate + 1;
Lsmbe:=lengthSMBE.sample;
p_SMBEp:- new SMBEpacket("SMBEpacket ");
BEgen_in_sec := Lsmbe/bitChannelRate;
    BEgen_in_sec_TOT := BEgen_in_sec_TOT + BEgen_in_sec;
p_SMBEp.Lsmbe:=Lsmbe;
p_SMBEp.schedule(0.0);

hold(nextSMBE.sample);
goto LOOP;
end***SMBEgen***;


!----SMBE PACKET----;
entity class SMBEpacket;
begin
```

```
integer actual_wl;
integer retransmission_attempt; !Keeps track of how many times
a BE packet is retransmitted;
long real Lsmbe;
long real start_BEp, time_left, start_transmission, resume_transmission;

be_insec := Lsmbe/bitChannelRate;
be_insec_tot := be_insec_tot + be_insec;
RETRANSMIT:
  serverBEwaiting_wl.give(1); !tell the server a BE packet
is waiting in the queue - invio un segnale di "presenza di
un pacchetto BE in coda" al server. Vd. foglio per maggiori
spiegazioni;
time_left:= Lsmbe/bitChannelRate;
if retransmission_attempt=0 then start_BEP:= time; !used to
compute BE lifetime;
BE_Q.wait;
delayBE := time - start_BEp;

while time_left > 0.0 do
begin
wavelengthOUT(actual_wl).acquire(1); !transmission starts;
flag(actual_wl) := 3;
resume_BE_pointer(actual_wl) :- NONE;
SMBEpacket_pointer(actual_wl) :- this SMBEpacket;
start_transmission:= time;
if not BE_resumed(actual_wl) then
begin
BE_in_sec_offerti_al_canale:= Lsmbe/bitChannelRate;
BE_in_sec_offerti_al_canale_TOT:= BE_in_sec_offerti_al_canale_TOT+
BE_in_sec_offerti_al_canale;
BEp_intoWL := BEp_intoWL + 1;
end;
hold(time_left);
!wl_BE_util(actual_wl):= wl_BE_util(actual_wl) + (time - start_transmission);
SMBEpacket_pointer(actual_wl) :- NONE;
```

```
wavelengthOUT(actual_wl).release(1); !packet transmitted or
interrupted;
if flag(actual_wl) = 3 then flag(actual_wl):= 0;
if not BE_resumed(actual_wl) then BEp_outtoWL := BEp_outtoWL
+ 1;


if interrupted = 0 then
begin
time_left:= 0.0;!whole packet transmitted (no interruptions);
BE_resumed(actual_wl):= false;
wl_BE_util(actual_wl):= wl_BE_util(actual_wl) + Lsmbe/bitChannelRate;
resume_BE_pointer(actual_wl) :- NONE; !if I'm here, it should
already be "none";
serverBEwaiting_wl.give(1);!packet transmitted --> signal to
BE server that wl is now available;
counter_BEpacket_successful:= counter_BEpacket_successful +
1;
BEsuccessfull_in_sec := Lsmbe/bitChannelRate;
BEsuccessfull_in_sec_TOT := BEsuccessfull_in_sec_TOT +
BEsuccessfull_in_sec;
BEdelay_hist.update(time - start_BEp);
BE_lifetime:= time - start_BEp; !delay of the BE packet, from
generation to the end of transmission;
total_BE_lifetime:= total_BE_lifetime + BE_lifetime;
total_BE_lifetime2:= total_BE_lifetime2 + BE_lifetime**2;

end;

if interrupted = 1 then
begin

serverBEwaiting_wl.give(1);!packet interrupted and put back
in queue --> signal to BE server that wl is now available;
if retransmission_attempt=0 then if not BE_resumed(actual_wl)
then
begin
```

```
interruptedByGST:=interruptedByGST+1;
counter_BEpacket_resumed  := counter_BEpacket_resumed + 1;
!count interrupted packets but multiple interruptions of the
same packet are not counted;
end;
counter_BEpacket_interrupted_by_GSTp := counter_BEpacket_interrupted_by_GSTp
+ 1;
counter_BEpacket_interrupted := counter_BEpacket_interrupted
+ 1;
BEinterrupted_in_sec := Lsmbe/bitChannelRate;
BEinterrupted_in_sec_TOT := BEinterrupted_in_sec_TOT + BEinterrupted_in_sec;
total_retransmission(actual_wl):=total_retransmission(actual_wl)+1;
retransmission_attempt:= retransmission_attempt + 1;
interrupted:=0;
time_wasted(actual_wl):= time_wasted(actual_wl) + (time - start_transmission);
GOTO retransmit;
end;

if interrupted = 2 then
begin
!interrupt for statistical;
counter_BEpacket_interrupted_by_RTp  := counter_BEpacket_interrupted_by_RTp
+ 1;
if retransmission_attempt=0 then if not BE_resumed(actual_wl)
then
begin
counter_BEpacket_resumed  := counter_BEpacket_resumed + 1;
!count interrupted packets but multiple interruptions of the
same packet are not counted;
interruptedByRT:=interruptedByRT+1;
end;
BE_resumed(actual_wl):= true;
counter_BEpacket_interrupted := counter_BEpacket_interrupted
+ 1;
BEinterrupted_in_sec := Lsmbe/bitChannelRate;
     BEinterrupted_in_sec_TOT := BEinterrupted_in_sec_TOT +
BEinterrupted_in_sec;
```

```
     total_retransmission(actual_wl):=total_retransmission(actual_wl)
resume_BE_pointer(actual_wl) :- this SMBEpacket;
time_left:= time_left - (time - start_transmission);
hold(sim_time+1);!infinite hold: the interrupted packet waits
to be scheduled again by the server;
interrupted:=0;
end;

end***while***;
!if flag(actual_wl) = 3 then flag(actual_wl):= 0;


end***SMBEpacket***;

!----SMBE SERVER----;
entity class BEserver;
begin
integer i;
ref(SMBEpacket) p_SMBEpacket;

START:
serverBEwaiting_wl.take(1);

for i:= 1 step 1 until OUTPUT_WL do
begin


!wavelength free? Some packet to resume in this wavelength?;
if flag(i) = 0 and resume_BE_pointer(i) =/= none then
begin
p_SMBEpacket :- resume_BE_pointer(i);
p_SMBEpacket.interrupt(3);!I use interrupt (instead of schedule)
because the packet is in an endless hold;
goto START;
end;

if flag(i) = 0 and BE_Q.length > 0 then
```

```
begin

p_SMBEpacket :- BE_Q.coopt;
p_SMBEpacket.actual_wl := i;
p_SMBEpacket.schedule(0.0);
BEp_inQ:= BEp_inQ + 1;
goto START;
end;


end***for***;

goto START;

end***BEserver***;


!------------;
!----MAIN----;
!------------;

outf :- new outfile ("report.txt");
outf.setaccess("Append");
outf.open(blanks(180));


outtext("**************************************************");
outimage;
outtext("
"); outimage;
setseed(my_seeds(count_observations));
outtext("OBSERVATION N. [0 - 9]: ");
outfix(count_observations, 0, 3);
outimage;
outtext("SEED:");
outfix(my_seeds(count_observations), 0, 13);
outimage;
```

```
outtext("
"); outimage;
outtext("**************************************************");
outimage;

!GST PACKET LENGTH (constant distribution);
for k := 1 step 1 until OUTPUT_WL do
lengthGST(k)         :- new Constant("L GSTp", length_GST);
!NEXT GST PACKET (negexp distrib. for the arrival process);
for k := 1 step 1 until OUTPUT_WL do
nextGST(k)           :- new NegExp("NextGSTp", 1/B); !1/B is
lambda, the parameter of a negexp distribution;

!SMRT PACKET LENGTH (negexp distribution);
lengthSMRT           :- new NegExp("L SMRTp", 1/mean_length_RT);
!NEXT SMRT PACKET (negexp distrib. for the arrival process);
nextSMRT            :- new NegExp("NextSMRTp", num_pRT_sec*OUTPUT_WL)

!SMBE PACKET LENGTH (negexp distribution);
lengthSMBE          :- new NegExp("L SMBEp", 1/mean_length_BE);
!NEXT SMBE PACKET (negexp distrib. for the arrival process);
nextSMBE            :- new NegExp("NextSMBEp", num_pBE_sec*OUTPUT_WL)

outtext("LAMBDAS");
outimage;

outtext("LAMBDA GST");
outfix(1/B,4,20);
outimage;

outtext("LAMBDA RT");
outfix(num_pRT_sec*OUTPUT_WL,4,20);
outimage;

outtext("LAMBDA BE");
outfix(num_pBE_sec*OUTPUT_WL,4,20);
outimage;
```

```
!OUTPUT WAVELENGTHS;
for k := 1 step 1 until OUTPUT_WL do
wavelengthOUT(k) :- new res(edit("wlOUT  ", k), 1);

!the global text procedure edit accepts a text t and an integer
n as actual
parameters and combines them into a single text (e.g. edit("aisle",
17) returns "aisle17".
If the text t is more than 10 characters long, then it is stripped
down to the first 10.
If the integer value of N is not in the range 0 through 99
then abs(n)//100 is accepted. It
is commonly used with res etc. arrays which share the same
text as title;

!FLAG;
for k := 1 step 1 until OUTPUT_WL do flag(k) := 0;

BE_Q :- new waitQ("BEtQueue");
serverBEwaiting_wl :- new bin("ServBE wait", 0);!first parameter
used in reports and
in traces. Second parameter = initial size of the pool;
!trace;

!GENERATORS;
for k := 1 step 1 until OUTPUT_WL do
new GSTgen("GSTgen", k).schedule(nextGST(k).sample);

new SMRTgen("RTgen").schedule(nextSMRT.sample);

new SMBEgen("BEgen").schedule(nextSMBE.sample);

new BEserver("BEserver").schedule(0.0);

BEdelay_hist :- new histogram ("delay_hist", 0, 0.000100, 19);
```

```
hold(transient_time);!a sort of warm up period;
reset;! "resets all Demos facilities created by the user, so
that they now collect afresh over the next time period";

!----reset----;

!reset statistic parameters for GST traffic;

for cont_res:= 1 step 1 until OUTPUT_WL do
begin
counter_GSTpacket_generate(cont_res):=0;
counter_GSTpacket_successful(cont_res):=0;
wl_GST_util(cont_res):=0;
end;
GSTp_intoWL:=0;
GSTp_outtoWL:=0;
GST_tot:=0;
GST_successful:=0;
gst_insec:=0.0;
gst_insec_tot:=0.0;
GST_wl_busy:=0;
gst_total_time:= 0.0;

!reset statistic parameters for RT traffic;
counter_RTpacket_generate:=0;
counter_RTpacket_successful:=0;
RT_wl_directly_free:=0;
RT_wl_busy:=0;
counter_RTpacket_lost:=0;
RTp_intoWL:=0;
RTp_outtoWL:=0;
rt_insec:=0.0;
rt_insec_tot:=0.0;
PLP_RT:=0.0;
PLP_RT2:=0.0;
rt_total_time:= 0.0;
```

```
for cont_res:= 1 step 1 until OUTPUT_WL do wl_RT_util(cont_res):=0;


!reset statistic parameters for BE traffic;

counter_BEpacket_generate:=0;
counter_BEpacket_successful:=0;
counter_BEpacket_interrupted:=0;
counter_BEpacket_interrupted_by_GSTp:=0;
counter_BEpacket_interrupted_by_RTp:=0;
counter_BEpacket_resumed:=0;
BEp_inQ:=0;
BEp_intoWL:=0;
BEp_outtoWL:=0;
be_insec:=0.0;
be_insec_tot:=0.0;
counter_BEpacket:=0;
somma:=0.0;
somma2:=0.0;
delayBE:=0.0;
PLP_BE:=0.0;
PLP_BE2:=0.0;
PLP_resumingBE:= 0.0;
vsum:=0.0;
v2sum:=0.0;
interruptedByGST:=0;
interruptedByRT:=0;
BE_lifetime:= 0.0;
total_BE_lifetime:= 0.0;
for cont_res:= 1 step 1 until OUTPUT_WL do
begin
wl_BE_util(cont_res):=0;
BE_resumed(cont_res):=false;
total_retransmission(cont_res):=0;
time_wasted(cont_res):=0;
end;
be_total_time:= 0.0;
```

```
BEgen_in_sec:=0.0;
BEgen_in_sec_TOT :=0.0;
BEsuccessfull_in_sec:=0.0;
BEsuccessfull_in_sec_TOT:=0.0;
BE_in_sec_offerti_al_canale:=0.0;
BE_in_sec_offerti_al_canale_TOT:=0.0;
BEinterrupted_in_sec := 0.0;
BEinterrupted_in_sec_TOT:= 0.0;
BEdelay_hist.reset;

!total statistic parameters;
TOTAL_P_GENERATED:=0;
TOTAL_P_intoWL:=0;

!-----end reset-----;

transient :- new outfile ("transient.txt");
transient.setaccess("Append");
transient.open(blanks(180));
transient.outimage;
transient.outtext("--------------------------------------------------
transient.outimage;

!by steps of 1% I can check (print out) the simulation progress;
for u:=1 step 1 until 100 do
begin
hold(sim_time/100.0);
!print out the % of simulation ;
outtext("Simulation in progress: ");
outint(u, 3);
outtext(" % complete");
!outtext(" % of total simulation time");
outimage;
if u<4 then
begin
transient.outfix(total_BE_lifetime, 16, 20);
transient.outimage;
transient.outfix(counter_BEpacket_successful, 2, 20);
transient.outimage;
end;
transient.outfix(total_BE_lifetime/counter_BEpacket_successful,
20, 22);
transient.outimage;
end;


transient.close;

for i:=1 step 1 until OUTPUT_WL do
begin
GST_tot:= GST_tot + counter_GSTpacket_generate(i);
end;

for i:=1 step 1 until OUTPUT_WL do
begin
GST_successful:= GST_successful + counter_GSTpacket_successful(i);
end;

PLP_RT:= counter_RTpacket_lost/counter_BEpacket;
!square value of PLP rt;
PLP_RT2:=PLP_RT**2;

PLP_resumingBE:= counter_BEpacket_resumed/BEp_inQ; !my PLP;
PLP_BE:= counter_BEpacket_interrupted/BEp_inQ; !Gaia's;
!square value of PLP be;
PLP_BE2:= PLP_BE**2;

TOTAL_P_GENERATED:= GST_TOT + counter_BEpacket_generate + counter_RTpacket_generate
TOTAL_P_intoWL:= GSTp_intoWL + counter_BEpacket + BEp_inQ;


!***************************PRINT****************************************;
!                          GST traffic
                ;
```

```
!****************************************************************
outtext("
"); outimage;
outtext("
"); outimage;
outtext("*********************GST**********************");
outimage;

!stampo a video quanti pacchetti GST sono stati generati da
ciascuna sorgente GST p-esima;
outtext("Column 1: source or wl number"); outimage;
outtext("Column 2: GST packets generated by each GST source");
outimage;
outtext("Column 3: GST packets successfully forwarded by each
GST source"); outimage;
outtext("Column 4: time used by each wl for GST packets");
outimage;
outtext("Column 5: time used by each wl for RT packets"); outimage;
outtext("Column 6: time used by each wl for BE packets"); outimage;
outtext("Column 7: time wasted by each wl for retransmission");
outimage;
outtext("Column 8: total number of retransmission for each
wl"); outimage;
outimage;
for i:= 1 step 1 until OUTPUT_WL do
begin
outint(i,2); outtext("  |"); outint(counter_GSTpacket_generate(i),
7); outint(counter_GSTpacket_successful(i), 10); outtext("
   |"); outfix(wl_gst_util(i), 5,12); outfix(wl_rt_util(i),
5,12); outfix(wl_be_util(i), 5,12); outfix(time_wasted(i),5,9);
outint (total_retransmission(i), 8);
outimage;
end;
outimage;

!media dei valori sopra indicati;
for i:= 1 step 1 until OUTPUT_WL do
```

```
begin
sum_counter_GSTpacket_generate:= counter_GSTpacket_generate(i)
+ sum_counter_GSTpacket_generate;
sum_counter_GSTpacket_successful:= counter_GSTpacket_successful(i)
+ sum_counter_GSTpacket_successful;
sum_wl_gst_util:= wl_gst_util(i) + sum_wl_gst_util;
sum_wl_rt_util := sum_wl_rt_util + wl_rt_util(i);
sum_wl_be_util:= sum_wl_be_util + wl_be_util(i);
sum_time_wasted:= time_wasted(i) + sum_time_wasted;
sum_total_retransmission:= sum_total_retransmission + total_retransmission(i);

end;
outtext("Avg: "); outfix(sum_counter_GSTpacket_generate/OUTPUT_WL,
1, 7); outfix(sum_counter_GSTpacket_successful/OUTPUT_WL, 1,
10); outtext("    |"); outfix (sum_wl_gst_util/OUTPUT_WL,5,12);
outfix (sum_wl_rt_util/OUTPUT_WL, 5,12); outfix(sum_wl_be_util/OUTPUT_WL,
5,12); outfix(sum_time_wasted/OUTPUT_WL,5,9); outtext(" ");
outfix(sum_total_retransmission/OUTPUT_WL, 2, 8);
outimage;


!stampo a video quanti paccheti GST TOTALI sono stati generati;
outtext("Total GST packets generated: ");
outfix(GST_tot, 0, 20);
outimage;

!stampo a video quanti paccheti GST TOTALI arrivano a destinazione;
outtext("GST packets successful: ");
outfix(GST_successful, 0, 20);
outimage;

outtext("Generated unsuccessful GST packets: ");
outfix(GST_tot-GST_successful, 0, 20);
outimage;

!how many packets are now in the system?;
outtext("GST packets in the FDL: ");
```

```
outfix(GST_tot-GSTp_intoWL, 0, 20);
outimage;

outtext("GST packets actively using wavelengths when simulation
stops: ");
outfix(GSTp_intoWL-GSTp_outtoWL, 0, 10);
outimage;

!pacchetti che prendono la risorsa;
outtext("GST packets inside WLs: ");
outfix(GSTp_intoWL, 0, 20);
outimage;

!pacchetti che rilasciano la risorsa;
outtext("GST packets releasing WL = GST packets successful:");
outfix(GSTp_outtoWL, 0, 20);
outimage;

!stampo a video quanti pacchetti BE vegono interrotti per poter
trasmettere un pacchetto GST;
outtext("GST packets finding a busy WL = BEp interrupted by
GST: ");
outint(GST_wl_busy, 10);
outimage;

!tempo totale impiegato da tutte le wl per trasmettere pacchetti
GST (dovrebbe coincidere con gst_insec_tot);
outtext("Total time used by all wl to transmit GST packets:");
for i:= 1 step 1 until OUTPUT_WL do gst_total_time:= gst_total_time
+ wl_GST_util(i);
outfix(gst_total_time,16,20);
outimage;

!quanti secondi sono impiegati per trasmettere pacchetti di
tipo GST;
outtext("sec to transmit all GST packets:");
outfix(gst_insec_tot,16,20);
```

```
outimage;
outtext("avg time needed by a WL to transmit GST packets over
tot sim time:");
outfix((gst_insec_tot/OUTPUT_WL)/sim_time, 16, 20);
outimage;


!****************************PRINT******************************************;
!                          RT traffic
              ;
!***************************************************************************;
outtext("
"); outimage;
outtext("
"); outimage;
outtext("***********************RT*************************");
outimage;

!stampo a video quanti paccheti RT sono stati generati;
outtext("RT packets generated: ");
outfix(counter_RTpacket_generate, 0, 20);
outimage;

!stampo a video quanti paccheti RT arrivano a destinazione;
outtext("RT packets successful: ");
outfix(counter_RTpacket_successful, 0, 20);
outimage;

outtext("RT packets unsuccessful: ");
outfix(counter_RTpacket_generate-counter_RTpacket_successful,
0, 20);
outimage;

!how many RT packets are now in the system?;
outtext("RT packets crossing the system: ");
outfix(counter_RTpacket_generate-RTp_intoWL, 0, 20);
outimage;
```

```
outtext("RT packets actively using wavelengths when the simulation
stops: ");
outfix(RTp_intoWL-RTp_outtoWL, 0, 10);
outimage;

!pacchetti RT che prendono la risorsa;
outtext("RT packets acquiring WL: ");
outfix(RTp_intoWL, 0, 20);
outimage;

!pacchetti RT che rilasciano la risorsa;
outtext("RT packets releasing WL = RT packets Successful: ");
outfix(RTp_outtoWL, 0, 20);
outimage;

!quanti pacchetti RT vanno persi;
outtext("RT packets lost (all wl busy): ");
outfix(counter_RTpacket_lost, 0, 20);
outimage;

!stampo a video i pacchetti RT che trovano subito libera la
wl;
outtext("RT packets that find a free WL: ");
outfix(RT_wl_directly_free, 0, 20);
outimage;

!stampo a video quanti pacchetti BE vegono interrotti per poter
trasmettere un pacchetto RT;
outtext("RT packets finding a busy WL = BEp interrupted by
RT: ");
outfix(RT_wl_busy, 0, 10);
outimage;

!stampo a video PLP RT;
outtext("PLP RT: ");
outfix(PLP_RT, 16, 20);
```

```
outimage;

!tempo totale impiegato da tutte le wl per trasmettere pacchetti
RT (dovrebbe coincidere con rt_insec_tot);
outtext("Total time used by all wl to transmit RT packets:");
for i:= 1 step 1 until OUTPUT_WL do rt_total_time:= rt_total_time
+ wl_RT_util(i);
outfix(rt_total_time,16,20);
outimage;

!quanti secondi sono impiegati per trasmettere pacchetti di
tipo RT;
outtext("sec to transmit all RTp: ");
outfix(rt_insec_tot,16,20);
outimage;
outtext("avg time needed by a WL to transmit RTp over tot sim
time :");
outfix((rt_insec_tot/OUTPUT_WL)/sim_time,16,20);
outimage;

!***************************PRINT***************************************;
!                          BE traffic
                ;
!**********************************************************************;
outtext("
"); outimage;
outtext("
"); outimage;
outtext("**********************BE***********************");
outimage;

!stampo a video quanti paccheti BE sono stati generati;
outtext("BEpackets generated: ");
outfix(counter_BEpacket_generate, 0, 20);
outimage;

!stampo a video quanti paccheti BE arrivano a destinazione;
```

```
outtext("BEpackets successful: ");
outfix(counter_BEpacket_successful, 0, 20);
outimage;

outtext("Generated unsuccessful BE packets: ");
outfix(counter_BEpacket_generate-counter_BEpacket_successful,
0, 20);
outimage;

!pacchetti BE che vanno in coda;
outtext("BEp_inQ = BEp queued = BEintoWL :");
outfix(BEp_inQ, 0,20);
outimage;

!pacchetti BE che prendono la risorsa;
outtext("BEp acquiring WL: ");
outfix(BEp_intoWL, 0, 20);
outimage;

!pacchetti BE che rilasciano la risorsa;
outtext("BEp releasing WL = BEp Successful: ");
outfix(BEp_outtoWL, 0, 20);
outimage;

outtext("BEps actively using wavelengths when simulation stops:
");
outfix(BEp_intoWL-BEp_outtoWL, 0, 20);
outimage;

!stampo a video quante volte vengono interrotti i pacchetti
BE;
outtext("Total interruptions = interruptions by GST + interruptions
by RT : ");
outfix(counter_BEpacket_interrupted, 0, 10); outimage;

!stampo a video quante volte vengono interrotti i pacchetti
BE da GST;
```

```
outtext("Total interruptions by GST: ");
outfix(counter_BEpacket_interrupted_by_GSTp, 0, 10);
outimage;

!stampo a video quante volte vengono interrotti i pacchetti
BE da RT;
outtext("Total interruptions by RT: ");
outfix(counter_BEpacket_interrupted_by_RTp, 0, 10);
outimage;

!stampo la somma dei 2 addendi, che deve essere uguale al numero
di volte in cui i pacchetti BE sono stati interrotti;
outtext("Total interruptions (GST + RT): ");
outfix(counter_BEpacket_interrupted_by_RTp+counter_BEpacket_interrupted_by_GSTp,
0, 10);
outimage;

!stampo quanti pacchetti sono stati ripresi (uno stesso pacchetto
interrotto/ripreso piu' volte viene contato una sola volta);
outtext("BEpackets interrupted: ");
outInt(counter_BEpacket_resumed,10);
outimage;

!stampo il numero di pacchetti BE interrotti da RT (uno stesso
pacchetto interrotto piu' volte viene contato una sola volta);
outtext("BEpackets interrupted by RT: ");
outInt(interruptedByRT,10);
outimage;

!stampo il numero di pacchetti BE interrotti da GST (uno stesso
pacchetto interrotto piu' volte viene contato una sola volta);
outtext("BEpackets interrupted by GST: ");
outInt(interruptedByGST,10);
outimage;

!stampo il numero di pacchetti BE interrotti (uno stesso pacchetto
interrotto piu' volte viene contato una sola volta);
```

```
!la somma deve essere uguale agli interrotti/ripresi poco sopra;
outtext("BEpackets interrupted: ");
outInt(interruptedByRT + interruptedByGST,10);
outimage;

!stampo una media delle interruzioni per un pacchetto BE;
outtext("Avg number of interruptions per BEpacket: ");
outfix(counter_BEpacket_interrupted/counter_BEpacket_successful,
4, 7);
outimage;

!tempo impiegato per trasmettere tutti i pacchetti BE (dalla
loro creazione a fine trasmissione);
outtext("Total lifetime of BE packets: ");
outfix(total_BE_lifetime, 16, 20);
outimage;

outtext("Square total lifetime of BE packets: ");
outfix(total_BE_lifetime2, 16, 20);
outimage;

!ritardo medio di un pacchetto BE (dalla creazione a fine trasmission
outtext("Avg lifetime of a BE packet: ");
outfix(total_BE_lifetime/counter_BEpacket_successful, 15, 17);
outimage;

!PLP: pacchetti interrotti/pacchetti serviti;
outtext("PLP resuming BE (pkt int/pkt served): ");
outfix(PLP_resumingBE,16,20);
outimage;

!PLP_BE (Gaia): interruzioni totali/pacchetti serviti;
outtext("PLP BE (Gaia: tot interr./pkt served): ");
outfix(PLP_BE, 16, 20);
outimage;

!stampo a video il rapporto tra quanti pacchetti BE sono stati
```

```
interrotti e quanti si sono messi in coda, quindi hanno avuto
accesso alla WL ;
outtext("BEinterrupted/BEinQueue ratio:");
outfix(counter_BEpacket_interrupted/BEp_inQ, 16, 20);
outimage;

!solo pacchetti BE interrotti da GST a numeratore;
outtext("BEinterrupted by GST/BEinQueue ratio:");
outfix(counter_BEpacket_interrupted_by_GSTp/BEp_inQ, 16, 20);
outimage;

!solo pacchetti BE interrotti da RT a numeratore;
outtext("BEinterrupted by RT/BEinQueue ratio:");
outfix(counter_BEpacket_interrupted_by_RTp/BEp_inQ, 16, 20);
outimage;

!tempo totale impiegato da tutte le wl per trasmettere pacchetti
BE (>= be_insec_tot a causa delle ritrasmissioni);
outtext("Total time used by all wl to transmit BE packets:
");
for i:= 1 step 1 until OUTPUT_WL do be_total_time:= be_total_time
+ wl_BE_util(i);
outfix(be_total_time,16,20);
outimage;

!quanti secondi sono impiegati per trasmettere pacchetti di
tipo BE;
outtext("sum of all BE packets (time):");
outfix(be_insec_tot,16,20);
outimage;
outtext("sum of all BE packets (time)/number of OUTPUT_WL ratio:");
outfix((be_insec_tot/OUTPUT_WL)/sim_time,16,20);
outimage;

msum:= total_BE_lifetime/counter_BEpacket_successful;
m2sum:= (total_BE_lifetime/counter_BEpacket_successful)**2;
```

```
vsum:=vsum+abs(BEp_inQ*somma2-somma**2.0)/(BEp_inQ*(BEp_inQ-1.0));
v2sum:= v2sum+(abs(BEp_inQ*somma2-somma**2.0)/(BEp_inq*(BEp_inQ-1.0))
outtext("
"); outimage;
outtext("**************************************************");
outimage;

!stampo a video quanti pacchetti in totale ho generato;
outtext("PACKETS GENERATED OVERALL: ");
outfix(TOTAL_P_GENERATED, 0, 20);
outimage;

!stampo a video quanti pacchetti cercano di utilizzare una
risorsa;
outtext("TOTAL PACKETS into the WL: ");
outfix(TOTAL_P_intoWL, 0, 20);
outimage;

!stampo a video rho GST a priori;
outtext("rho GST a priori: ");
outfix(rho_gst, 5, 10);
outimage;
!stampo a video la percentuale assoluta di GST calcolata a
posteriori;
outtext("Actual absolute percentage of GST packets: ");
outfix((gst_insec_tot/OUTPUT_WL)/sim_time, 16, 20);
outimage;

!stampo a video la percentuale assoluta di RT calcolata a priori;
outtext("rho RT a priori : ");
outfix(rho_RT, 5, 10);
outimage;
!stampo a video la percentuale assoluta di RT calcolata a posteriori;
outtext("Actual absolute percentage of RT packets: ");
outfix((rt_insec_tot/OUTPUT_WL)/sim_time, 16, 25);
outimage;
```

```
!stampo a video la percentuale assoluta di BE calcolata a priori;
outtext("rho BE a priori: ");
outfix(rho_BE, 5, 10);
outimage;
!stampo a video la percentuale assoluta di BE calcolata a posteriori;
outtext("Actual absolute percentage of BE packets: ");
outfix((be_insec_tot/OUTPUT_WL)/sim_time, 16, 25);
outimage;

END***DEMOS***;
END;

PLPrt(count_observations):= PLP_RT;
PLPrt2(count_observations):= PLP_RT2;
sum_PLPrt := sum_PLPrt+PLPrt(count_observations);
sum_PLPrt2 := sum_PLPrt2+PLPrt2(count_observations);

PLPbe(count_observations):= PLP_BE;
PLPbe2(count_observations):= PLP_BE2;
sum_PLPbe := sum_PLPbe+PLPbe(count_observations);
sum_PLPbe2 := sum_PLPbe2+PLPbe2(count_observations);
msum_vettore(count_observations):=msum;
m2sum_vettore(count_observations):=m2sum;
vsum_vettore(count_observations):=vsum;
v2sum_vettore(count_observations):=v2sum;
sum_msum_vettore:= sum_msum_vettore+msum_vettore(count_observations);
sum_m2sum_vettore:= sum_m2sum_vettore + m2sum_vettore(count_observations);
sum_vsum_vettore:= sum_vsum_vettore+vsum_vettore(count_observations);
sum_v2sum_vettore:= sum_v2sum_vettore+v2sum_vettore(count_observations);

outtext("
"); outimage;
outtext("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"); outimage;
outtext("
"); outimage;
outtext("OBSERVED VALUE = PLP of RT : ");
outint(count_observations,2); outfix(PLPrt(count_observations),
40, 45);
outimage;

outtext("SQUARE OBSERVED VALUE: PLP**2 of RT ");
outint(count_observations,2); outfix(PLPrt2(count_observations),
40, 45);
outimage;

outtext("OBSERVED VALUE = PLP of BE : ");
outint(count_observations,2); outfix(PLPbe(count_observations),
40, 45);
outimage;

outtext("SQUARE OBSERVED VALUE: PLP**2 of BE ");
outint(count_observations,2); outfix(PLPbe2(count_observations),
40, 45);
outimage;

outtext("OBSERVED VALUE=vsum : ");
outint(count_observations,2); outfix(vsum_vettore(count_observations)
40, 45);
outimage;

outtext("SQUARE OBSERVED VALUE=v2sum  ");
outint(count_observations,10); outfix(v2sum_vettore(count_observation
40, 45);
outimage;

end***for***;



sum_meanPLPrt:=sum_PLPrt/count_observations;
sum_meanPLPrt2:=sum_PLPrt2/count_observations;
deviazione_std_rt:= sqrt(abs(10.0*sum_PLPrt2-sum_PLPrt*sum_PLPrt)/90.
ConfInter95rt:= (deviazione_std_rt*2.262)/sqrt(10.0);
lower_rt:= sum_meanPLPrt-ConfInter95rt;
```

```
upper_rt:= sum_meanPLPrt+ConfInter95rt;
if lower_rt>0 then adjustDELTA_rt:=ConfInter95rt
    else adjustDELTA_rt:=ConfInter95rt+lower_rt;

    outtext("Sum of PLP RT: ");
outfix(sum_PLPrt, 40, 45);
outimage;

outtext("Mean value of sum of PLP RT : ");
outfix(sum_meanPLPrt, 40, 45);
outimage;

outtext("Sum of PLP**2 of: ");
outfix(sum_PLPrt2, 40, 45);
outimage;

outtext("Mean value of sum of PLP**2 of RT : ");
outfix(sum_meanPLPrt2, 40, 45);
outimage;

outtext("Standard deviation RT: ");
outfix(deviazione_std_rt, 40, 45);
outimage;

outtext("95% Confidential Intervall : ");
outfix(ConfInter95rt, 40, 45);
outimage;

outtext("LOWER of RT : ");
outfix(lower_rt, 40, 45);
outimage;

outtext("UPPER of RT : ");
outfix(upper_rt, 40, 45);
outimage;

outtext("Adjusted lower 95% Confidence interval of RT : ");
```

```
outfix(adjustDELTA_rt, 40, 45);
outimage;

outtext("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
outimage;
outtext("
"); outimage;

sum_meanPLPbe:=sum_PLPbe/count_observations;
sum_meanPLPbe2:=sum_PLPbe2/count_observations;
deviazione_std_be:= sqrt(abs(10.0*sum_PLPbe2-sum_PLPbe*sum_PLPbe)/90.
ConfInter95be:= (deviazione_std_be*2.262)/sqrt(10.0);
lower_be:= sum_meanPLPbe-ConfInter95be;
upper_be:= sum_meanPLPbe+ConfInter95be;
if lower_be>0 then adjustDELTA_be:=ConfInter95be
   else adjustDELTA_be:=ConfInter95be+lower_be;

outtext("Sum of PLP BE: ");
outfix(sum_PLPbe, 40, 45);
outimage;

outtext("Mean value of sum of PLP BE : ");
outfix(sum_meanPLPbe, 40, 45);
outimage;

outtext("Sum of PLP**2 of BE: ");
outfix(sum_PLPbe2, 40, 45);
outimage;

outtext("Mean value of sum of PLP**2 of BE : ");
outfix(sum_meanPLPbe2, 40, 45);
outimage;

outtext("Standard deviation BE: ");
outfix(deviazione_std_be, 40, 45);
outimage;
```

```
outtext("95% Confidential Interval of BE: ");
outfix(ConfInter95be, 40, 45);
outimage;

outtext("LOWER of BE : ");
outfix(lower_be, 40, 45);
outimage;

outtext("UPPER of BE : ");
outfix(upper_be, 40, 45);
outimage;

outtext("Adjusted lower 95% Confidence interval of BE : ");
outfix(adjustDELTA_be, 40, 45);
outimage;

outtext("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
outimage;
outtext("
"); outimage;

deviazione_std_delay:=sqrt(abs(10.0*sum_m2sum_vettore-sum_msum_vettore**2)/90.0);
ConfInter95_delay:= (deviazione_std_delay*2.262)/sqrt(10.0);
lower_delay:= sum_msum_vettore/10.0-ConfInter95_delay;
upper_delay:= sum_msum_vettore/10.0+ConfInter95_delay;
if lower_delay>0 then adjustDELTA_be:=ConfInter95_delay
   else adjustDELTA_delay:=ConfInter95_delay+lower_delay;

outtext("Mean value of delay: ");
outfix(sum_msum_vettore/10.0, 40, 45);
outimage;

outtext("Standard deviation of delay: ");
outfix(deviazione_std_delay, 40, 45);
outimage;

outtext("95% Confidence interval of delay: ");
```

```
outfix(ConfInter95_delay, 40, 45);
outimage;

outtext("Lower bound for delay: ");
outfix(lower_delay, 40, 45);
outimage;

outtext("Upper bound for delay: ");
outfix(upper_delay, 40, 45);
outimage;


outtext("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
outimage;
outtext("
"); outimage;


END;
```

Bibliography

[1] *DEMOS - A system for Discrete Event Modelling on Simula.* 2003.

[2] Jan Cheyns, Erik Van Breusegem, Didier Colle, Mario Pickavet, and Pie Demeester. ORION: a Novel Hybrid Network Concept: Overspill Routing in Optical Networks. *Proceedings of 2003 5th International Conference on Transparent Optical Networks*, 1:144 – 147, 2003.

[3] Gaia Leli. Performance study of the 3LIHON output scheduling part. Master's thesis, NTNU, February 2012.

[4] I. Mitrani. *Simulation techniques for discrete event systems.* Cambridge University Press, 1982.

[5] Carla Raffaelli, Slavisa Aleksic, Franco Callegati, Walter Cerroni, Guido Maier, Achille Pattavina, and Michele Savi. Optical Packet Switching. *Enabling Optical Internet with Advanced Network Technologies*, pages 31 – 85, 2009.

[6] Norvald Stol, Michele Savi, and Carla Raffaelli. 3-level integrated hybrid optical network (3LIHON) to meet future QoS requirements. *Global Telecommunications Conference*, pages 1 – 6, Dec 2011.

Ringraziamenti

I miei piu' sentiti ringraziamenti vanno al Prof. Walter Cerroni, per avermi dato la possibilita' di svolgere all'estero il lavoro di preparazione di tesi, presso una delle piu' importanti universita' della Norvegia, NTNU. Grazie al suo supporto e alla sua guida mi e' stato possibile inserirmi all'interno di un progetto di ricerca estremamente interessante e innovativo, trascorrendo in tale Paese sei mesi che hanno dato come frutto la presente tesi.

Un grazie speciale va al Prof. Norvald Stol, che con molta disponibilita' e dedizione ha supervisionato lo sviluppo del mio lavoro dall'inizio alla fine, aiutandomi ad avere una visione globale dell'argomento, prodigandosi in ottime spiegazioni ed approfondimenti e lasciandomi grande liberta' nell'indirizzo dei miei studi, per i quali l'aspetto creativo e' stata componente di grande motivazione ed entusiasmo.

Ringrazio i miei genitori per l'aiuto morale, economico e affettivo, non solo durante questi sei mesi, ma anche per tutti gli anni di studi. Perche' mi hanno lasciato sempre libero di scegliere, senza farmi mai mancare nulla.

Meritano la mia gratitudine anche la zia Nadia, soprattutto per essersi talvolta prestata ai "quiz notturni pre-esame" con grande spirito di sacrificio; mio cugino Marco, che e' sempre riuscito a trovare lo spunto per qualche battuta anche su materie niente affatto divertenti; la zia Mirella, per il continuo interessamento ai miei avanzamenti negli studi e per il fondamentale supporto gastronomico - rigorosamente romagnolo - durante i sei mesi a Trondheim; la zia Dalide, per aver accolto sempre con entusiasmo le mie novita', per lo sforzo di "tec-

nologizzazione"pur di superare i 2600 km di distanza e perche', ormai raggiunto il secolo, le sue maggiori preoccupazioni sono ancora che "*l'inzignir*"mangi a sufficienza e non prenda freddo.

Grazie a Vincenzo, non solo come compagno di studi, ma come Amico, perche' parlando con lui ogni esame e' sembrato piu' facile (almeno per un momento!), perche' nei momenti di bisogno c'e' sempre stato, perche' c'e' sempre stato anche nei momenti di solitudine, per il piacere di vedermi e non solo per necessita', per avermi infuso parte della sua fermezza e determinazione, per avermi dedicato varie ore del proprio tempo libero a fine giornata e nei weekend ad aiutarmi con solerzia e passione nei miei esami, per essere stato compagno di tante serate.
Insieme a lui, Giovanni ha contribuito a rendere piena la mia vita extra-universitaria, compensando le ore sui libri con le nostre uscite, risate, prese in giro e tanti ritorni in bici all'alba con i merli che cantano.
Grazie a Roberto, persona che posso fortunatamente ed orgogliosamente annoverare tra i veri Amici, il quale e' stato l'unico a rinunciare a due settimane di caldo mediterraneo per venirmi a trovare in Norvegia e fare un indimenticabile esperienza al di la' del Circolo Polare: ripercorrendo questi ultimi sedici anni, capisco che condividiamo molto di piu' dei nostri viaggi.

Sono sicuro che, dentro a quell'asettico numero che e' il voto di laurea, siano in realta' racchiusi anche tutti i vostri contributi. **Grazie.**